Yuehui Chen and Ajith Abraham

Intelligent Systems Reference Library

Intelligent Systems Reference Library, Volume 2

Further volumes of this series can be found on our homepage: springer.com

Vol. 1. Christine L. Mumford and Lakhmi C. Jain (Eds.)
Computational Intelligence, 2009
ISBN 978-3-642-01798-8

Vol. 2. Yuehui Chen and Ajith Abraham
Tree-Structure Based Hybrid Computational Intelligence, 2009
ISBN 978-3-642-04738-1

Yuehui Chen and Ajith Abraham

Tree-Structure Based Hybrid Computational Intelligence

Theoretical Foundations and Applications

Springer

Prof. Yuehui Chen
School of Information Science and Engineering
University of Jinan
Jiwei Road 106
Jinan 250022
P.R. China
E-mail: yhchen@ujn.edu.cn

Prof. Ajith Abraham
Machine Intelligence Research Labs (MIR Labs)
Scientific Network for Innovation and Research Excellence
P.O. Box 2259
Auburn, Washington 98071-2259
USA
E-mail: ajith.abraham@ieee.org

ISBN 978-3-642-04738-1 e-ISBN 978-3-642-04739-8

DOI 10.1007/978-3-642-04739-8

Intelligent Systems Reference Library ISSN 1868-4394

Library of Congress Control Number: 2009936208

Typeset & *Cover Design:* Scientific Publishing Services Pvt. Ltd., Chennai, India.

Printed in acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

This book is dedicated to all our family members and colleagues around the world who supported us during the last several years.

This work is supported by Publishing Foundation Project of University of Jinan.

Preface

Computational intelligence is a well-established paradigm, where new theories with a sound biological understanding have been evolving. The current experimental systems have many of the characteristics of biological computers (brains in other words) and are beginning to be built to perform a variety of tasks that are difficult or impossible to do with conventional computers. In a nutshell, which becomes quite apparent in the light of the current research pursuits, the area is heterogeneous as being dwelled on such technologies as neurocomputing, fuzzy inference systems, artificial life, probabilistic reasoning, evolutionary computation, swarm intelligence and intelligent agents and so on.

Research in computational intelligence is directed toward building thinking machines and improving our understanding of intelligence. As evident, the ultimate achievement in this field would be to mimic or exceed human cognitive capabilities including reasoning, recognition, creativity, emotions, understanding, learning and so on. Even though we are a long way from achieving this, some success has been achieved in mimicking specific areas of human mental activity.

Recent research in computational intelligence together with other branches of engineering and computer science has resulted in the development of several useful intelligent paradigms. The integration of different learning and adaptation techniques, to overcome individual limitations and achieve synergetic effects through hybridization or fusion of some of these techniques, has in recent years contributed to a large number of new hybrid intelligent system designs.

Learning methods and approximation algorithms are fundamental tools that deal with computationally hard problems, in which the input is gradually disclosed over time. Both kinds of problems have a large number of applications arising from a variety of fields, such as function approximation and classification, algorithmic game theory, coloring and partitioning, geometric problems, mechanism design, network design, scheduling, packing and covering and real-world applications such as medicine, computational finance, and so on.

In this book, we illustrate Hybrid Computational Intelligence (HCI) framework and it applications for various problem solving tasks. Based on tree-structure based encoding and the specific function operators, the models can be flexibly constructed and evolved by using simple computational intelligence techniques. The main idea behind this model is the flexible neural tree, which is very adaptive, accurate and efficient. Based on the pre-defined instruction/operator sets, a flexible neural tree model can be created and evolved. The flexible neural tree could be evolved by using tree-structure based evolutionary algorithms with specific instructions. The fine tuning of the parameters encoded in the structure could be accomplished by using parameter optimization algorithms. The flexible neural tree method interleaves both optimizations. Starting with random structures and corresponding parameters, it first tries to improve the structure and then as soon as an improved structure is found, it fine tunes its parameters. It then goes back to improving the structure again and, provided it finds a better structure, it again fine tunes the rules' parameters. This loop continues until a satisfactory solution is found or a time limit is reached.

This volume is organized into 6 Chapters and the main contributions are detailed below:

Chapter 1 provides a gentle introduction to some of the key paradigms in computational intelligence namely evolutionary algorithms and its variants, swarm intelligence, artificial neural networks, fuzzy expert systems, probabilistic computing and hybrid intelligent systems.

Chapter 2 exhibits the flexible neural tree algorithm development and is first illustrated in some function approximation problems and also in some real world problems like intrusion detection, exchange rate forecasting, face recognition, cancer detection and protein fold recognition. Further the multi-input multi-output flexible neural tree algorithm is introduced and is illustrated for some problem solving. Finally an ensemble of flexible neural trees is demonstrated for stock market prediction problem.

Chapter 3 depicts three different types of hierarchical architectures. First the design and implementation of hierarchical radial basis function networks are illustrated for breast cancer detection and face recognition. Further, the development of hierarchical B-spline networks is demonstrated for breast cancer detection and time series prediction. Finally, hierarchical wavelet neural networks are presented for several function approximation problems.

Building a hierarchical fuzzy system is a difficult task. This is because the user has to define the architecture of the system (the modules, the input variables of each module, and the interactions between modules), as well as the rules of each modules. **Chapter 4** demonstrates a new encoding and an automatic design method for the hierarchical Takagi-Sugeno fuzzy inference system with some simulation results related to system identification and time-series prediction problems.

Can we evolve a symbolic expression that can be represented as a meaningful expression, i.e., a differential equation or a transfer function and it can be easily addressed by using traditional techniques? **Chapter 5** exhibits a new representation scheme of the additive models, by which the linear and nonlinear system identification problems are addressed by using automatic evolutionary design procedure. First a gentle introduction to tree structural representation and calculation of the additive tree models is provided. Further an hybrid algorithm for evolving the additive tree models and some simulation results for the prediction of chaotic time series, the reconstruction of polynomials and the identification of the linear/nonlinear system is demonstrated.

Chapter 6 summarizes the concept of hierarchical hybrid computational intelligence framework introduced in this book and also provides some future research directions.

We are very much grateful to Dr. Thomas Ditzinger (Springer Engineering Inhouse Editor, Professor Janusz Kacprzyk (Editor- in-Chief, Springer *Intelligent Systems Reference Library* Series) and Ms. Heather King (Editorial Assistant, Springer Verlag, Heidelberg) for the editorial assistance and excellent cooperative collaboration to produce this important scientific work. We hope that the reader will share our joy and will find it useful!

Yuehui Chen and Ajith Abraham*
School of Information Science and Engineering,
University of Jinan, Jiwei Road 106, Jinan 250022,
Peoples Republic of China
http://cilab.ujn.edu.cn
Email: yhchen@ujn.edu.cn

*Machine Intelligence Research Labs (MIR Labs)
Scientific Network for Innovation and Research Excellence
P.O. Box 2259, Auburn, Washington 98071, USA
http://www.mirlabs.org
http://www.softcomputing.net
email: ajith.abraham@ieee.org

Contents

Part I: Foundations of Computational Intelligence

1 Foundations of Computational Intelligence . . . 3
1.1 Introduction . . . 3
1.2 Evolutionary Algorithms . . . 3
1.2.1 Genetic Programming . . . 7
1.2.2 Estimation of Distribution Algorithm . . . 11
1.2.3 Population-Based Incremental Learning . . . 13
1.2.4 Probabilistic Incremental Program Evolution . . . 14
1.3 Swarm Intelligence . . . 18
1.3.1 Particle Swarm Optimization . . . 19
1.3.2 Ant Colony Optimization . . . 21
1.4 Artificial Neural Networks . . . 23
1.4.1 Architecture and Learning Algorithm . . . 24
1.4.2 Multilayer Perceptron . . . 26
1.4.3 Back-Propagation Algorithm . . . 27
1.4.4 Evolutionary Algorithm Based Training . . . 28
1.4.5 Self Organizing Feature Maps . . . 29
1.4.6 Radial Basis Function . . . 30
1.4.7 Recurrent Neural Networks . . . 30
1.4.8 Adaptive Resonance Theory . . . 31
1.5 Fuzzy Systems . . . 31
1.5.1 The Definition of Fuzzy Sets . . . 32
1.6 Takagi-Sugeno Fuzzy Model . . . 32
1.6.1 Universal Approximation Property . . . 33
1.6.2 Fuzzy Expert Systems - Design Challenges . . . 34
1.7 Probabilistic Computing . . . 34
1.8 Hybrid Intelligent Systems . . . 35
1.9 Models of Hybrid Intelligent Systems . . . 36

Part II: Flexible Neural Trees

2 Flexible Neural Tree: Foundations and Applications 39
2.1 Introduction to Flexible Neural Tree 39
2.2 Flexible Neural Tree Algorithms 40
2.2.1 Encoding and Evaluation 40
2.2.2 Flexible Neuron Instructor 40
2.2.3 Fitness Function 42
2.2.4 Structure and Parameter Learning 42
2.2.5 Flexible Neural Tree Applications 44
2.2.6 Exchange Rate Forecasting 64
2.2.7 Face Recognition 69
2.2.8 Microarray-Based Cancer Classification 73
2.2.9 Protein Fold Recognition 76
2.3 Multi Input Multi Output Flexible Neural Tree 79
2.4 Representation and Calculation of the MIMO FNT 80
2.4.1 Hybrid Algorithm for Structure and Parameter Learning 82
2.4.2 Hybrid Algorithm for Flexible Neural Tree Model 84
2.4.3 Illustrative Examples 84
2.5 Ensemble of Flexible Neural Tree 89
2.5.1 The Basic Ensemble Method 90
2.5.2 The Generalized Ensemble Method 90
2.5.3 The LWPR Method 90
2.5.4 Stock Index Forecasting Problem 91
2.6 Stock Index Forecasting Experimental Illustrations 93

Part III: Hierarchical Neural Networks

3 Hierarchical Neural Networks 99
3.1 Hierarchical Radial Basis Function Neural Networks 99
3.1.1 The Radial Basis Function Network 100
3.1.2 Automatic Design of Hierarchical Radial Basis Function Network 101
3.1.3 Tree Structure Optimization by Extended Compact Genetic Programming (ECGP) 102
3.1.4 Parameter Optimization Using Differential Evolution Algorithm 102
3.1.5 Procedure of the General Learning Algorithm 103
3.1.6 Variable Selection in the HRBF Network Paradigms 103
3.1.7 Experimental Illustrations 104
3.1.8 Face Recognition 105

3.2 Hierarchical B-Spline Neural Networks 108
3.2.1 The B-Spline Network 108
3.3 Automatic Design of HB-Spline Network 109
3.3.1 Encode and Calculation for HB-Spline 109
3.3.2 Tree Structure and Parameter Optimization 110
3.3.3 Procedure of the General Learning Algorithm 111
3.3.4 Variable Selection in the Hierarchical B-Spline Network Paradigms 111
3.3.5 Experimental Illustrations 111
3.3.6 Wisconsin Breast Cancer Detection 111
3.3.7 Time-Series Forecasting 113
3.4 Hierarchical Wavelet Neural Networks 118
3.4.1 Wavelet Neural Network 118
3.5 Automatic Design of Hierarchical Wavelet Neural Network 119
3.5.1 Ant Programming for Evolving the Architecture of HWNN 119
3.5.2 Parameter Optimization Using Differential Evolution Algorithm 121
3.5.3 Procedure of the General Learning Algorithm for HWNN 121
3.5.4 Variable Selection Using HWNN Paradigms 122
3.5.5 Experimental Illustrations 122
3.5.6 Application to Jenkins-Box Time-Series 124

Part IV: Hierarchical Fuzzy Systems

4 Hierarchical Fuzzy Systems 129
4.1 Introduction 129
4.2 Takagi-Sugeno Fuzzy Inference System (TS-FS) 131
4.3 Hierarchical TS-FS: Encoding and Evaluation 131
4.3.1 Encoding 132
4.3.2 Evaluation 133
4.3.3 Objective Function 134
4.4 Evolutionary Design of Hierarchical TS-FS 135
4.4.1 Algorithm for Designing Hierarchical TS-FS Model 135
4.4.2 Feature/Input Selection with Hierarchical TS-FS 136
4.5 Experimental Illustrations 137
4.5.1 Systems Identification 138
4.5.2 Chaotic Time-Series of Mackey-Glass 139
4.5.3 Iris Data Classification 142
4.5.4 Wine Data Classification 144

Part V: Reverse Engineering of Dynamical Systems

5 Reverse Engineering of Dynamic Systems 151
5.1 Introduction 151
5.2 Calculation and Representation of Additive Models 152
5.3 Hybrid Algorithm 153
5.3.1 Tree-Structure Based Evolutionary Algorithm 153
5.3.2 Evolving an Optimal or Near-Optimal Structure of Additive Model 154
5.3.3 Parameter Optimization 156
5.3.4 Summary of General Learning Algorithm 158
5.3.5 Experimental Illustrations 159
5.3.6 Discussions 165
5.4 Inferring a System of Differential Equations 168
5.5 Inference of Differential Equation Models by Multi Expression Programming 169
5.5.1 Structure Optimization by the MEP 169
5.5.2 Parameter Optimization by Particle Swarm Optimization Algorithm 170
5.5.3 Fitness Definition 171
5.5.4 Summary of Algorithm 172
5.6 Modeling Chemical Reactions 172
5.6.1 Simple Chemical Reaction Model 173
5.6.2 Two-Species Lotka-Volterra Model 174
5.6.3 Bimolecular Reaction 175
5.7 Inferring Gene Regulatory Networks 176
5.7.1 The Small Artificial Gene Regulatory Network 178
5.7.2 The Large-Scale Artificial Gene Regulatory Network with Noisy Environment 181

Part VI: Conclusions and Future Research

6 Concluding Remarks and Further Research 185
6.1 Limitations of Conventional Computational Intelligence 185
6.2 Towards Tree-Structure Based Hierarchical Hybrid Computational Intelligence 186
6.2.1 Tree Structure Based Evolutionary Computation Models 186
6.2.2 Hierarchical Hybrid Computational Intelligence Framework 186
6.3 Static and Dynamical Models 190

References 191

Part I

Foundations of Computational Intelligence

1

Foundations of Computational Intelligence

1.1 Introduction

The field of computational intelligence has evolved with the objective of developing machines that can think like humans. Computational intelligence is a well-established paradigm, where new theories with a sound biological understanding have been evolving. The current experimental systems have many of the characteristics of biological computers (brains in other words) and are beginning to be built to perform a variety of tasks that are difficult or impossible to do with conventional computers. To name a few, we have microwave ovens, washing machines and digital camera that can figure out on their own what settings to use to perform their tasks optimally with reasoning capability, make intelligent decisions and learn from experience. As usual, defining computational intelligence is not an easy task. In a nutshell, which becomes quite apparent in light of the current research pursuits, the area is heterogeneous as being dwelled on such technologies as neural networks, fuzzy systems, evolutionary computation, artificial life, multi-agent systems and probabilistic reasoning. The recent trend is to integrate different components to take advantage of complementary features and to develop a synergistic system. Hybrid architectures like neuro-fuzzy systems, evolutionary-fuzzy systems, evolutionary-neural networks, evolutionary neuro-fuzzy systems etc. are widely applied for real world problem solving.

This Chapter provides a gentle introduction to some of the key paradigms in computational intelligence namely evolutionary algorithms and its variants, swarm intelligence, artificial neural networks, fuzzy expert systems, probabilistic computing and hybrid intelligent systems.

1.2 Evolutionary Algorithms

Evolution can be viewed as a search process capable of locating solutions to problems offered by an environment. Therefore, it is quite natural to look for

Y. Chen, A. Abraham.: Tree-Struc. Based Hybrid Com. Intelligence, ISRL 2, pp. 3–36.
springerlink.com

an algorithmic description of evolution that can be used for problem solving. Such an algorithmic view has been discussed even in philosophy. Those iterative (search and optimization) algorithms developed with the inspiration of the biological process of evolution are termed evolutionary algorithms (EAs). They are aimed basically at problem solving and can be applied to a wide range of domains, from planning to control. Evolutionary computation (EC) is the name used to describe the field of research that embraces all evolutionary algorithms. The basic idea of the field of evolutionary computation, which came onto the scene about the 1950s/1960s, has been to make use of the powerful process of natural evolution as a problem-solving paradigm, usually by simulating it on a computer. The original three mainstreams of EC are genetic algorithms (GAs), evolution strategies (ES), and evolutionary programming (EP) [1][2]. Despite some differences among these approaches, all of them present the basic features of an evolutionary process as proposed by the Darwinian theory of evolution.

A standard evolutionary algorithm is illustrated as follows:

- A population of individuals that reproduce with inheritance. Each individual represents or encodes a point in a search space of potential solutions to a problem. These individuals are allowed to reproduce (sexually or asexually), generating offspring that carry some resemblance with their parents;
- Genetic variation. Offspring are prone to genetic variation through mutation, which alters their genetic makeup;
- Natural selection. The evaluation of individuals in their environment results in a measure of adaptability, quality, or fitness value to be assigned to them. A comparison of individual fitnesses will lead to a competition for survival and reproduction in the environment, and there will be a selective advantage for those individuals of higher fitness [306].

The standard evolutionary algorithm is a generic, iterative and probabilistic algorithm that maintains a population P of N individuals, $P = x_1, x_2, , x_N$, at each iteration t (for simplicity of notation the iteration index t was suppressed). Each individual corresponds to (represents or encodes) a potential solution to a problem that has to be solved. An individual is represented using a data structure. The individuals x_i, $i = 1, , N$, are evaluated to give their measures of adaptability to the environment, or fitness. Then, a new population, at iteration $t + 1$, is generated by selecting some (usually the most fit) individuals from the current population and reproducing them, sexually or asexually. If employing sexual reproduction, a genetic recombination (crossover) operator may be used. Genetic variations through mutation may also affect some individuals of the population, and the process iterates. The completion of all these steps: reproduction, genetic variation, and selection, constitutes what is called a generation. An initialization procedure is used to generate the initial population of individuals. Two parameters p_c and p_m correspond to the genetic recombination and variation probabilities, and will be further discussed.

Note that all evolutionary algorithms involve the basic concepts common to every algorithmic approach to problem solving:

- representation (data structures);
- definition of an objective; and
- specification of an evaluation function (fitness function).

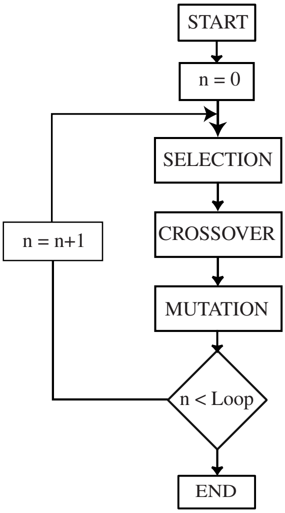

Fig. 1.1 A flowchart of simple genetic algorithm

Genetic algorithms (GAs) are globally stochastic search technique that emulates the laws of evolution and genetics to try to find optimal solutions to complex optimization problems. GAs are theoretically and empirically proven to provide to robust search in complex spaces, and they are widely applied in engineering, business and scientific circles. The general flowchart of GA is presented in Figure 1.1.

GAs are different from more normal optimization and search procedures in different ways:

- GAs work with a coding of the parameter set, not the parameter themselves.
- GAs search from a population of points, not a single point.
- GAs use objective function information, not derivatives or other auxiliary knowledge, but with modifications they can exploit analytical gradient information if it is available.
- GAs use probabilistic transition rules, not deterministic rules.

Coding and Decoding

Coding refers to the representation of the parameter used in the optimization problem. The usually used coding methods in GAs are base-2, base-10 and floating-point coding methods. In a base-2 representation, alleles (values in the position, genes on the chromosome) are 0 and 1. In base-10, the alleles take on integer values between 0 and 9. In floating-point representation, the alleles are real-valued number. In base-2 and base-10 representations, the relationship between the real value of a parameter and its integer representation can be expressed by:

$$x = a + \bar{x}\frac{range}{resolution} \tag{1.1}$$

where x is the real value of the parameter, $\bar{x}$ is the integer value corresponding to the x, a is the lowest value assumed by $\bar{x}$, *range* is the interval of definition of the parameters, and *resolution* is the number that take in account the number of bits used, i.e., $2^{number\ of\ bits} - 1$.

Genetic Operators

A simple genetic algorithm that yields good results in many practical problems consists of three genetic operators:

- *Reproduction* is a process in which individual strings are copied according to their objective or fitness function values. Fitness function can be imagined as some measure of profit, utility, or goodness to be optimized. For example, in curve fitting problem, the fitness function can be mean square error:

  $$Fitness = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(y_i, a_i))^2 \tag{1.2}$$

  where y_i is the experimental data, $f(y_i, a_i)$ is the function chosen as model and a_i are the model parameters to be optimized by GA. When GA is used to optimize an adaptive controller, the error and change in error information can be taken account for the designing of a proper fitness function. In general, reproduction operator guarantee survival of the better individual to the next generation with a higher probability, which is an artificial version of natural selection.
- *Crossover* is a partial exchange of the genetic content between couples of members of the population. This task can be done in several different ways and it also depends on the representation scheme chosen. In integer representation, the simple way to do it, is to choose a random value with a uniform distribution as $[1, length\ of\ chromosome]$. This number represents a marker inside the two strings of bits representing the couple of chromosomes. It cuts both the chromosomes into two parts. Then, the left

or the right parts of the two chromosomes are swapped. This occurs in such a way that both the two new chromosomes will contain a part of the genetic information of both the parents. In the floating-point representation, the crossover should be realized by:

$$new_1 = a \cdot old_1 + (1 - a) \cdot old_2, \tag{1.3}$$

$$new_2 = (1 - a) \cdot old_1 + a \cdot old_2, \tag{1.4}$$

where new_1 and new_2 are the chromosomes after the crossover, old_1 and old_2 are the chromosomes before the crossover, a is a random number with uniform distribution in [0,1].

- *Mutation* is needed because, even through reproduction and crossover effectively search and recombine extant notions, occasionally they may become overzealous and lose some potentially useful genetic materials. In GA, the mutation operator protects against such an irrecoverable loss. In other words, mutation tries to escape from a local maximum or minimum of the fitness function, and it seeks to explore other areas of the search space in order to find a global maximum or minimum of the fitness function. In integer representation, the mutation of gene in a position of the chromosome is randomly changed form one integer to another. In floating-point representation, mutation will randomly change the value of the chromosome within a range of definition.

1.2.1 Genetic Programming

Genetic Programming (GP) technique provides a framework for automatically creating a working computer program from a high-level problem statement of the problem [30]. Genetic programming achieves this goal of automatic programming by genetically breeding a population of computer programs using the principles of Darwinian natural selection and biologically inspired operations. The main difference between genetic programming and genetic algorithms is the representation of the solution. Genetic programming creates computer programs in the LISP or scheme computer languages as the solution. LISP is an acronym for LISt Processor and was developed by John McCarthy in the late 1950s. Unlike most languages, LISP is usually used as an interpreted language. This means that, unlike compiled languages, an interpreter can process and respond directly to programs written in LISP. The main reason for choosing LISP to implement GP is due to the advantage of having the programs and data have the same structure, which could provide easy means for manipulation and evaluation.

GP is the extension of evolutionary learning into the space of computer programs. In GP the individual population members are not fixed length character strings that encode possible solutions to the problem at hand, they are programs that, when executed, are the candidate solutions to the problem. These programs are expressed in genetic programming as parse trees,

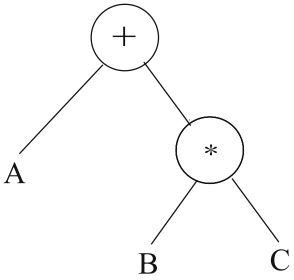

Fig. 1.2 A simple tree structure of GP

rather than as lines of code. Example, the simple program $a + b * c$ would be represented as shown in Figure 1.2. The terminal and function sets are also important components of genetic programming. The terminal and function sets are the alphabet of the programs to be made. The terminal set consists of the variables and constants of the programs (example, A,B and C in Figure 1.2).

The most common way of writing down a function with two arguments is the infix notation. That is, the two arguments are connected with the operation symbol between them as follows:

$$A + B$$

A different method is the prefix notation. Here the operation symbol is written down first, followed by its required arguments.

$$+AB$$

While this may be a bit more difficult or just unusual for human eyes, it opens some advantages for computational uses. The computer language LISP uses symbolic expressions (or S-expressions) composed in prefix notation. Then a simple S-expression could be

$$(\text{operator}, \text{argument})$$

where operator is the name of a function and argument can be either a constant or a variable or either another symbolic expression as shown below:

$$(\text{operator}, \text{argument}(\text{operator}, \text{argument})(\text{operator}, \text{argument}))$$

A parse tree (Figure 1.3) is a structure that develops the interpretation of a computer program. Functions are written down as nodes, their arguments as leaves. A subtree is the part of a tree that is under an inner node of this tree. If this tree is cut out from its parent, the inner node becomes a root node and the subtree is a valid tree of its own.

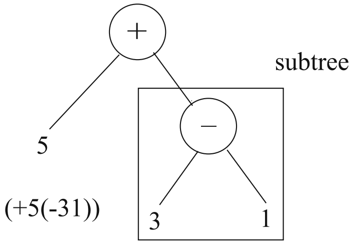

Fig. 1.3 Illustration of a parse tree and a subtree

There is a close relationship between these parse trees and S-expression; in fact these trees are just another way of writing down expressions. While functions will be the nodes of the trees (or the operators in the S-expressions) and can have other functions as their arguments, the leaves will be formed by terminals, that is symbols that may not be further expanded. Terminals can be variables, constants or specific actions that are to be performed. The process of selecting the functions and terminals that are needed or useful for finding a solution to a given problem is one of the key steps in GP.

Evaluation of these structures is straightforward. Beginning at the root node, the values of all sub-expressions (or subtrees) are computed, descending the tree down to the leaves. GP procedure could be summarized as follows:

- Generate an initial population of random compositions of the functions and terminals of the problem;
- Compute the fitness values of each individual in the population;
- Using some selection strategy and suitable reproduction operators produce offsprings;
- Procedure is iterated until the required solution is found or the termination conditions have reached (specified number of generations).

The creation of an offspring from the crossover operation is accomplished by deleting the crossover fragment of the first parent and then inserting the crossover fragment of the second parent. The second offspring is produced in a symmetric manner. A simple crossover operation is illustrated in Figure 1.4. In GP the crossover operation is implemented by taking randomly selected sub trees in the individuals and exchanging them.

Mutation is another important feature of genetic programming. Two types of mutations are commonly used. The simplest type is to replace a function or a terminal by a function or a terminal respectively. In the second kind an entire subtree can replace another subtree. Figure 1.5 explains the concept of mutation:

GP requires data structures that are easy to handle and evaluate and robust to structural manipulations. These are among the reasons why the class of S-expressions was chosen to implement GP. The set of functions and terminals that will be used in a specific problem has to be chosen carefully. If the set of

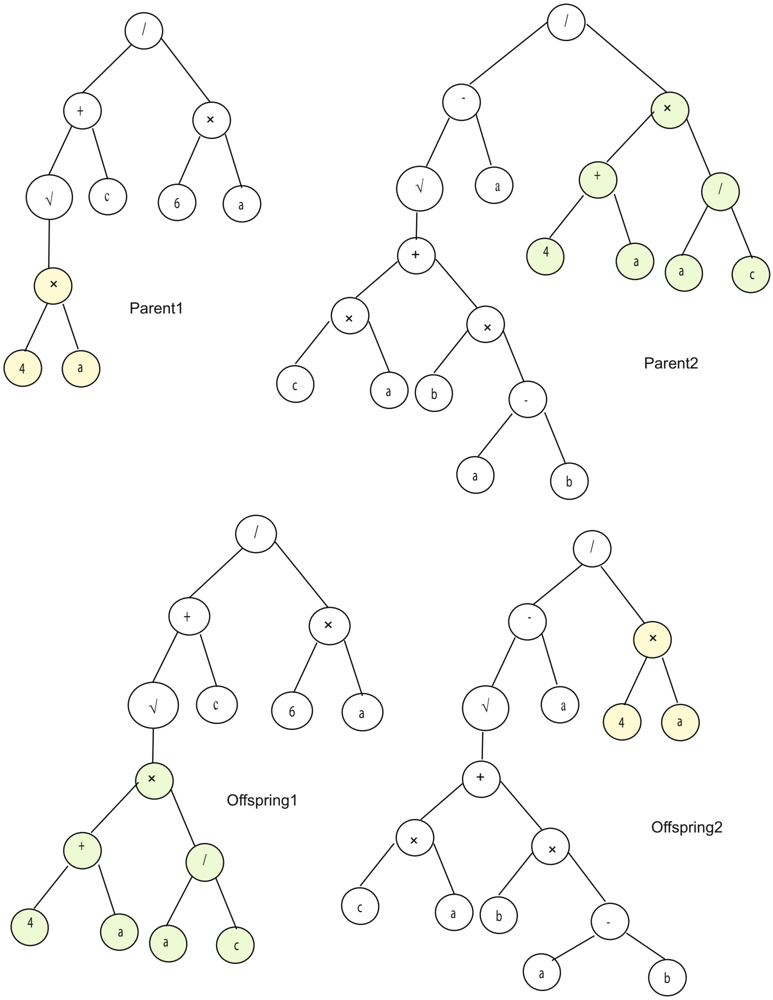

Fig. 1.4 Illustration of crossover operator

functions is not powerful enough, a solution may be very complex or not to be found at all. Like in any evolutionary computation technique, the generation of first population of individuals is important for successful implementation of GP. Some of the other factors that influence the performance of the algorithm are the size of the population, percentage of individuals that participate in the crossover/mutation, maximum depth for the initial individuals and the maximum allowed depth for the generated offspring etc. Some specific advantages of genetic programming are that no analytical knowledge is needed and still could get accurate results. GP approach does scale with the problem