Session : Genetic Programming

Course Title: Computational Intelligence
Course Code: 19CSE422A

Course Leader:
Dr. Vaishali R. Kulkarni
Assistant Professor, Department of Computer Science and Engineering
Faculty of Engineering and Technology
Ramaiah University of Applied Sciences, Bengaluru
Email: vaishali.cs.et@msruas.ac.in
Tel: +91-804-906-5555 Ext:2325
Website: www.msruas.ac.in/staff/fet_cse#Vaishali

Objectives of this Session

I wish to introduce:

1. Genetic Programming (GP)
2. Tree-based chromosome representation in GP
3. Fitness function evaluation in GP
4. Common crossover operators in GP
5. Common mutation operators in GP
6. Building block GP (BGP) and
7. Applications of GP

Intended Outcomes of this Session

At the end of this session, the student will be able to:

1. Distinguish GP from a GA
2. Represent a program code or a decision tree using a tree
3. Judge root and leaf nodes of a tree
4. Evaluate the fitness of a tree
5. Perform crossover and mutation on trees
6. Distinguish standard GP from BGP and
7. Summarize the application potential of GP

Recommended Resources for this Session

1. Engelbrecht, A. P. (2007). *Computational intelligence: An introduction*. Chichester, England, John Wiley & Sons.

2. De Jong, K. A. (2012). *Evolutionary Computation: A Unified Approach*. New York, USA, Bradford Books.

3. Konar, A. (2005). *Computational Intelligence: Principles, Techniques and Applications*. Secaucus, NJ, USA, Springer-Verlag New York, Inc.

Genetic Programming

- One of the central challenges of computer science is to get a computer to do what needs to be done, without telling it how to do it

Genetic Programming

- One of the central challenges of computer science is to get a computer to do what needs to be done, without telling it how to do it
- Genetic programming (GP) addresses this challenge by providing a method for automatically creating a working computer program from a high-level statement of the problem

Genetic Programming

- One of the central challenges of computer science is to get a computer to do what needs to be done, without telling it how to do it
- Genetic programming (GP) addresses this challenge by providing a method for automatically creating a working computer program from a high-level statement of the problem
- Automatic programming (a.k.a. program synthesis or program induction)

Genetic Programming

- One of the central challenges of computer science is to get a computer to do what needs to be done, without telling it how to do it
- Genetic programming (GP) addresses this challenge by providing a method for automatically creating a working computer program from a high-level statement of the problem
- Automatic programming (a.k.a. program synthesis or program induction)
- GP is a specialization of GAs. It concentrates on the evolution of genotypes

Genetic Programming

- One of the central challenges of computer science is to get a computer to do what needs to be done, without telling it how to do it
- Genetic programming (GP) addresses this challenge by providing a method for automatically creating a working computer program from a high-level statement of the problem
- Automatic programming (a.k.a. program synthesis or program induction)
- GP is a specialization of GAs. It concentrates on the evolution of genotypes
- GAs use string representations and GP uses a tree representation

Genetic Programming

- One of the central challenges of computer science is to get a computer to do what needs to be done, without telling it how to do it
- Genetic programming (GP) addresses this challenge by providing a method for automatically creating a working computer program from a high-level statement of the problem
- Automatic programming (a.k.a. program synthesis or program induction)
- GP is a specialization of GAs. It concentrates on the evolution of genotypes
- GAs use string representations and GP uses a tree representation

Genetic Programming

- One of the central challenges of computer science is to get a computer to do what needs to be done, without telling it how to do it
- Genetic programming (GP) addresses this challenge by providing a method for automatically creating a working computer program from a high-level statement of the problem
- Automatic programming (a.k.a. program synthesis or program induction)
- GP is a specialization of GAs. It concentrates on the evolution of genotypes
- GAs use string representations and GP uses a tree representation

Genetic Programming

- One of the central challenges of computer science is to get a computer to do what needs to be done, without telling it how to do it
- Genetic programming (GP) addresses this challenge by providing a method for automatically creating a working computer program from a high-level statement of the problem
- Automatic programming (a.k.a. program synthesis or program induction)
- GP is a specialization of GAs. It concentrates on the evolution of genotypes
- GAs use string representations and GP uses a tree representation

Tree-Based Representation

- GP was developed to evolve executable computer programs. Each chromosome represents a computer program, represented using a tree structure

Tree-Based Representation

- GP was developed to evolve executable computer programs. Each chromosome represents a computer program, represented using a tree structure

- **Adaptive individuals:** GP population usually has individuals of different size, shape and complexity (Size: tree depth, Shape: branching factor of nodes in the tree)

Tree-Based Representation

- GP was developed to evolve executable computer programs. Each chromosome represents a computer program, represented using a tree structure
- **Adaptive individuals:** GP population usually has individuals of different size, shape and complexity (Size: tree depth, Shape: branching factor of nodes in the tree)
- **Domain-specific grammar:** A grammar accurately reflects the problem to be solved. The defined grammar should be good enough to represent any possible solution

Tree Representation Grammar

- A terminal set, function set, and semantic rules are defined

Tree Representation Grammar

- A terminal set, function set, and semantic rules are defined
- The terminal set specifies all the variables and constants. The function set contains all the functions that can be applied to the elements of the terminal set

Tree Representation Grammar

- A terminal set, function set, and semantic rules are defined
- The terminal set specifies all the variables and constants. The function set contains all the functions that can be applied to the elements of the terminal set
- The functions may include mathematical, arithmetic and/or Boolean

Tree Representation Grammar

- A terminal set, function set, and semantic rules are defined
- The terminal set specifies all the variables and constants. The function set contains all the functions that can be applied to the elements of the terminal set
- The functions may include mathematical, arithmetic and/or Boolean
- Decision structures such as if-then-else and loops can also be included in the function set

Tree Representation Grammar

- A terminal set, function set, and semantic rules are defined
- The terminal set specifies all the variables and constants. The function set contains all the functions that can be applied to the elements of the terminal set
- The functions may include mathematical, arithmetic and/or Boolean
- Decision structures such as if-then-else and loops can also be included in the function set
- Elements of the terminal set form the leaf nodes of the evolved tree, and elements of the function set form the non-leaf nodes

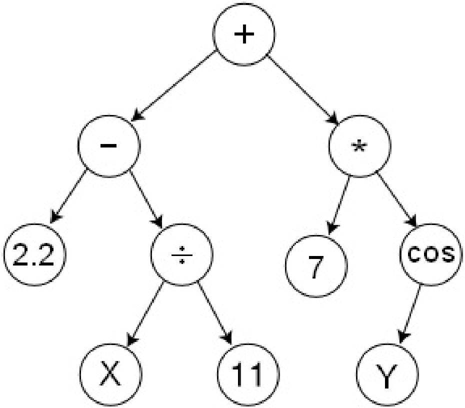
Tree Representation Grammar

- A terminal set, function set, and semantic rules are defined
- The terminal set specifies all the variables and constants. The function set contains all the functions that can be applied to the elements of the terminal set
- The functions may include mathematical, arithmetic and/or Boolean
- Decision structures such as if-then-else and loops can also be included in the function set
- Elements of the terminal set form the leaf nodes of the evolved tree, and elements of the function set form the non-leaf nodes
- For a problem, the search space consists of the set of all possible trees that can be constructed using the defined grammar

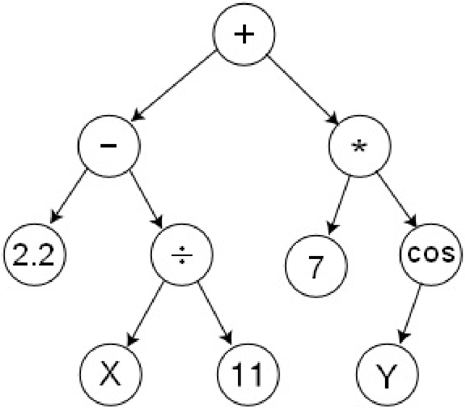
Tree Representation Grammar

- A terminal set, function set, and semantic rules are defined
- The terminal set specifies all the variables and constants. The function set contains all the functions that can be applied to the elements of the terminal set
- The functions may include mathematical, arithmetic and/or Boolean
- Decision structures such as if-then-else and loops can also be included in the function set
- Elements of the terminal set form the leaf nodes of the evolved tree, and elements of the function set form the non-leaf nodes
- For a problem, the search space consists of the set of all possible trees that can be constructed using the defined grammar

An Example of Tree Representation



An Example of Tree Representation



$$\left(2.2 - \frac{X}{11}\right) + \left(7 \times \cos Y\right)$$

An Example of Tree Representation

- In this example, the terminal set is specified as $\{2.2, 7, 11, X, Y\}$ with $X, Y \in \mathbb{R}$

An Example of Tree Representation

- In this example, the terminal set is specified as $\{2.2, 7, 11, X, Y\}$ with $X, Y \in \mathbb{R}$
- The minimal function set is given as $\{+, -, \times, \div, \cos\}$

An Example of Tree Representation

- In this example, the terminal set is specified as $\{2.2, 7, 11, X, Y\}$ with $X, Y \in \mathbb{R}$

- The minimal function set is given as $\{+, -, \times, \div, \cos\}$

Initial Population

- The initial population is generated randomly within the restrictions of a maximum depth and semantics of the grammar

Initial Population

- The initial population is generated randomly within the restrictions of a maximum depth and semantics of the grammar
- For each individual, a root is randomly selected from the set of function elements

Initial Population

- The initial population is generated randomly within the restrictions of a maximum depth and semantics of the grammar
- For each individual, a root is randomly selected from the set of function elements
- The branching factor of the root, and each non-terminal node, are determined by the arity of the selected function

Initial Population

- The initial population is generated randomly within the restrictions of a maximum depth and semantics of the grammar
- For each individual, a root is randomly selected from the set of function elements
- The branching factor of the root, and each non-terminal node, are determined by the arity of the selected function
- For each non-root node, an element is selected either from the terminal set or the function set

Initial Population

- The initial population is generated randomly within the restrictions of a maximum depth and semantics of the grammar
- For each individual, a root is randomly selected from the set of function elements
- The branching factor of the root, and each non-terminal node, are determined by the arity of the selected function
- For each non-root node, an element is selected either from the terminal set or the function set
- After an element from the terminal set is selected, the corresponding node becomes a leaf node and is no longer considered for expansion

Initial Population

- The initial population is generated randomly within the restrictions of a maximum depth and semantics of the grammar
- For each individual, a root is randomly selected from the set of function elements
- The branching factor of the root, and each non-terminal node, are determined by the arity of the selected function
- For each non-root node, an element is selected either from the terminal set or the function set
- After an element from the terminal set is selected, the corresponding node becomes a leaf node and is no longer considered for expansion
- Individuals are initialized to be simple. In the evolutionary process these grow if increased complexity is necessary

Initial Population

- The initial population is generated randomly within the restrictions of a maximum depth and semantics of the grammar
- For each individual, a root is randomly selected from the set of function elements
- The branching factor of the root, and each non-terminal node, are determined by the arity of the selected function
- For each non-root node, an element is selected either from the terminal set or the function set
- After an element from the terminal set is selected, the corresponding node becomes a leaf node and is no longer considered for expansion
- Individuals are initialized to be simple. In the evolutionary process these grow if increased complexity is necessary
- This facilitates creation of simple solutions

Initial Population

- The initial population is generated randomly within the restrictions of a maximum depth and semantics of the grammar
- For each individual, a root is randomly selected from the set of function elements
- The branching factor of the root, and each non-terminal node, are determined by the arity of the selected function
- For each non-root node, an element is selected either from the terminal set or the function set
- After an element from the terminal set is selected, the corresponding node becomes a leaf node and is no longer considered for expansion
- Individuals are initialized to be simple. In the evolutionary process these grow if increased complexity is necessary
- This facilitates creation of simple solutions

Fitness Function

- The fitness function is problem-dependent. If individuals represent a program, the fitness is obtained by program evaluation in a number of test cases

Fitness Function

- The fitness function is problem-dependent. If individuals represent a program, the fitness is obtained by program evaluation in a number of test cases
- For the mathematical expression, a set of sample input patterns and associated target outputs is needed

Fitness Function

- The fitness function is problem-dependent. If individuals represent a program, the fitness is obtained by program evaluation in a number of test cases
- For the mathematical expression, a set of sample input patterns and associated target outputs is needed
- For each pattern, the output of the expression represented by the individual is compared with the target to compute the error

Fitness Function

- The fitness function is problem-dependent. If individuals represent a program, the fitness is obtained by program evaluation in a number of test cases
- For the mathematical expression, a set of sample input patterns and associated target outputs is needed
- For each pattern, the output of the expression represented by the individual is compared with the target to compute the error
- The MSE over the errors for all the patterns gives the fitness of the individual

Fitness Function

- The fitness function is problem-dependent. If individuals represent a program, the fitness is obtained by program evaluation in a number of test cases
- For the mathematical expression, a set of sample input patterns and associated target outputs is needed
- For each pattern, the output of the expression represented by the individual is compared with the target to compute the error
- The MSE over the errors for all the patterns gives the fitness of the individual
- In some applications, individuals represents a decision tree. The fitness of individuals is calculated as the classification accuracy of the corresponding decision tree

Fitness Function

- If the objective is to evolve a game strategy in terms of a computer program, the fitness of an individual can be the number of times that the individual won the game

Fitness Function

- If the objective is to evolve a game strategy in terms of a computer program, the fitness of an individual can be the number of times that the individual won the game
- The fitness function can also be used to penalize individuals with undesirable structural properties

Fitness Function

- If the objective is to evolve a game strategy in terms of a computer program, the fitness of an individual can be the number of times that the individual won the game
- The fitness function can also be used to penalize individuals with undesirable structural properties
- For example, instead of having a predetermined depth limit, the depth of a tree can be penalized by adding an appropriate penalty term to the fitness function

Fitness Function

- If the objective is to evolve a game strategy in terms of a computer program, the fitness of an individual can be the number of times that the individual won the game
- The fitness function can also be used to penalize individuals with undesirable structural properties
- For example, instead of having a predetermined depth limit, the depth of a tree can be penalized by adding an appropriate penalty term to the fitness function
- Similarly, bushy trees can be penalized by adding a penalty term to the fitness function

Fitness Function

- If the objective is to evolve a game strategy in terms of a computer program, the fitness of an individual can be the number of times that the individual won the game
- The fitness function can also be used to penalize individuals with undesirable structural properties
- For example, instead of having a predetermined depth limit, the depth of a tree can be penalized by adding an appropriate penalty term to the fitness function
- Similarly, bushy trees can be penalized by adding a penalty term to the fitness function
- The fitness function can also be used to penalize semantically incorrect individuals

Crossover Operators

- Any selection operator can be used to select two parents to produce offspring. Two approaches can be used to generate offspring, each one differing in the number of offspring generated

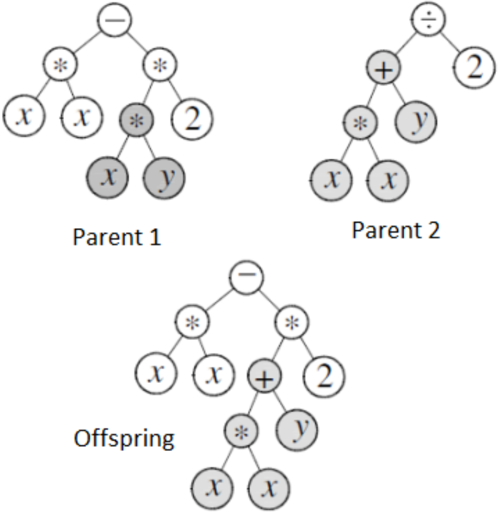
Crossover Operators

- Any selection operator can be used to select two parents to produce offspring. Two approaches can be used to generate offspring, each one differing in the number of offspring generated
- **Generating one offspring:** A random node is selected within each of the parents. Crossover then replaces the corresponding subtree in the one parent by that of the other parent

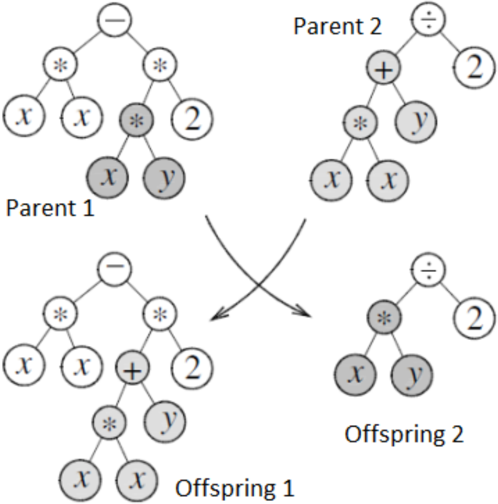
Crossover Operators

- Any selection operator can be used to select two parents to produce offspring. Two approaches can be used to generate offspring, each one differing in the number of offspring generated
- **Generating one offspring:** A random node is selected within each of the parents. Crossover then replaces the corresponding subtree in the one parent by that of the other parent
- **Generating two offspring:** A random node is selected in each of the two parents. The corresponding subtrees are swapped to create two offspring

One-Offspring Crossover



Two-Offspring Crossover



Mutation Operators

Typical mutation operators are:

- **Function node mutation:** A randomly chosen function node is replaced with another randomly chosen one with same arity

Mutation Operators

Typical mutation operators are:

- **Function node mutation:** A randomly chosen function node is replaced with another randomly chosen one with same arity
- **Terminal node mutation:** A randomly selected terminal node is replaced with another randomly selected one

Mutation Operators

Typical mutation operators are:

- **Function node mutation:** A randomly chosen function node is replaced with another randomly chosen one with same arity
- **Terminal node mutation:** A randomly selected terminal node is replaced with another randomly selected one
- **Swap mutation:** A function node is randomly selected and its arguments are swapped

Mutation Operators

Typical mutation operators are:

- **Function node mutation:** A randomly chosen function node is replaced with another randomly chosen one with same arity
- **Terminal node mutation:** A randomly selected terminal node is replaced with another randomly selected one
- **Swap mutation:** A function node is randomly selected and its arguments are swapped
- **Grow mutation:** A node is randomly selected and replaced by a randomly generated depth-restricted subtree

Mutation Operators

Typical mutation operators are:

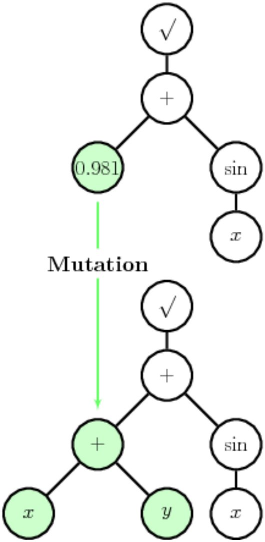
- **Function node mutation:** A randomly chosen function node is replaced with another randomly chosen one with same arity
- **Terminal node mutation:** A randomly selected terminal node is replaced with another randomly selected one
- **Swap mutation:** A function node is randomly selected and its arguments are swapped
- **Grow mutation:** A node is randomly selected and replaced by a randomly generated depth-restricted subtree
- **Gaussian mutation:** A terminal node that represents a constant is randomly selected and mutated by adding a Gaussian random value to that constant

Mutation Operators

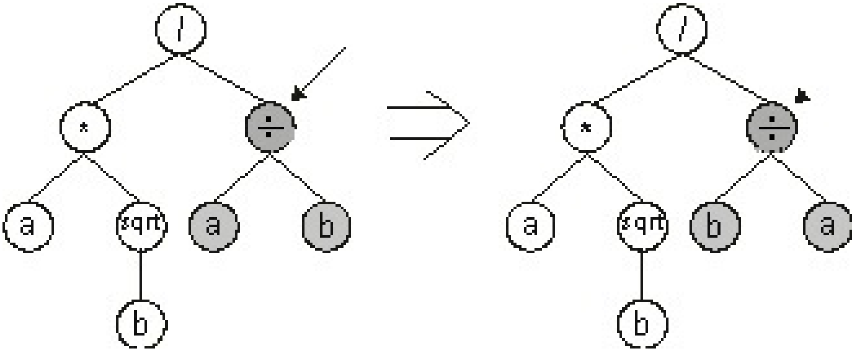
Typical mutation operators are:

- **Function node mutation:** A randomly chosen function node is replaced with another randomly chosen one with same arity
- **Terminal node mutation:** A randomly selected terminal node is replaced with another randomly selected one
- **Swap mutation:** A function node is randomly selected and its arguments are swapped
- **Grow mutation:** A node is randomly selected and replaced by a randomly generated depth-restricted subtree
- **Gaussian mutation:** A terminal node that represents a constant is randomly selected and mutated by adding a Gaussian random value to that constant
- **Trunc mutation:** A function node is randomly selected and replaced by a random terminal node

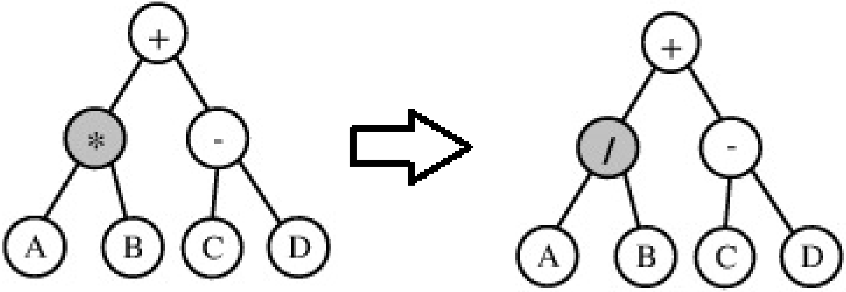
An Example of Mutation



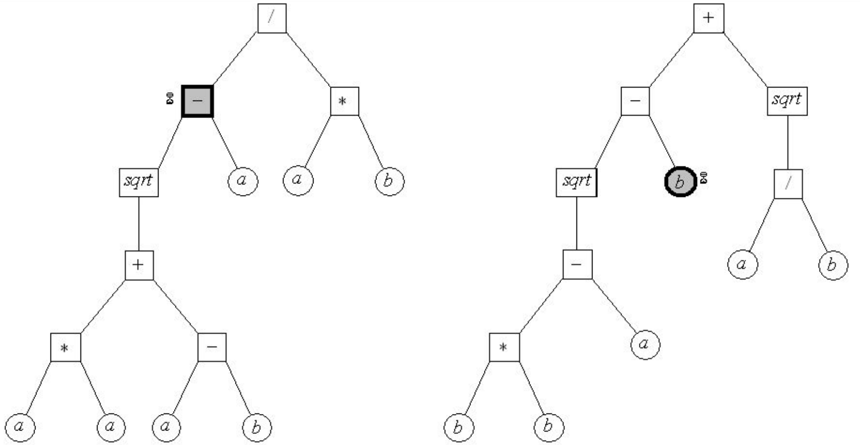
An Example of Mutation



An Example of Mutation



An Example of Mutation



Building Block GP (BGP)

- An alternative approach developed specifically for evolving decision trees

Building Block GP (BGP)

- An alternative approach developed specifically for evolving decision trees
- In BGP, initial individuals consist of only a root and the immediate children of that node

Building Block GP (BGP)

- An alternative approach developed specifically for evolving decision trees
- In BGP, initial individuals consist of only a root and the immediate children of that node
- Evolution starts on these small initial trees

Building Block GP (BGP)

- An alternative approach developed specifically for evolving decision trees
- In BGP, initial individuals consist of only a root and the immediate children of that node
- Evolution starts on these small initial trees
- When the simplicity of the populations individuals can no longer account for the complexity of the problem to be solved individuals are expanded

Building Block GP (BGP)

- An alternative approach developed specifically for evolving decision trees
- In BGP, initial individuals consist of only a root and the immediate children of that node
- Evolution starts on these small initial trees
- When the simplicity of the populations individuals can no longer account for the complexity of the problem to be solved individuals are expanded
- Expansion occurs by adding a randomly generated building block to individuals (grow mutation)

Building Block GP (BGP)

- An alternative approach developed specifically for evolving decision trees
- In BGP, initial individuals consist of only a root and the immediate children of that node
- Evolution starts on these small initial trees
- When the simplicity of the populations individuals can no longer account for the complexity of the problem to be solved individuals are expanded
- Expansion occurs by adding a randomly generated building block to individuals (grow mutation)
- This expansion occurs at a specified expansion probability p_e and, therefore, not all of the individuals are expanded

Building Block GP (BGP)

- An alternative approach developed specifically for evolving decision trees
- In BGP, initial individuals consist of only a root and the immediate children of that node
- Evolution starts on these small initial trees
- When the simplicity of the populations individuals can no longer account for the complexity of the problem to be solved individuals are expanded
- Expansion occurs by adding a randomly generated building block to individuals (grow mutation)
- This expansion occurs at a specified expansion probability p_e and, therefore, not all of the individuals are expanded
- This approach helps to reduce the computational complexity and helps to produce smaller individuals

Building Block GP (BGP)

- An alternative approach developed specifically for evolving decision trees
- In BGP, initial individuals consist of only a root and the immediate children of that node
- Evolution starts on these small initial trees
- When the simplicity of the populations individuals can no longer account for the complexity of the problem to be solved individuals are expanded
- Expansion occurs by adding a randomly generated building block to individuals (grow mutation)
- This expansion occurs at a specified expansion probability p_e and, therefore, not all of the individuals are expanded
- This approach helps to reduce the computational complexity and helps to produce smaller individuals

Applications of GP

- Decision trees
- Game-playing
- Bioinformatics
- Data mining
- Robotics

Session Summary

1. Genetic programming provides a method for automatically creating a working computer program from a high-level statement of the problem

Session Summary

1. Genetic programming provides a method for automatically creating a working computer program from a high-level statement of the problem
2. Computer programmes are represented by chromosomes that are structured as trees

Session Summary

1. Genetic programming provides a method for automatically creating a working computer program from a high-level statement of the problem
2. Computer programmes are represented by chromosomes that are structured as trees
3. Popular crossover operators: One-offspring and two-offspring

Session Summary

1. Genetic programming provides a method for automatically creating a working computer program from a high-level statement of the problem
2. Computer programmes are represented by chromosomes that are structured as trees
3. Popular crossover operators: One-offspring and two-offspring
4. Popular mutation operators: Function node, terminal node, Swap, Grow, Gaussian, and Trunc

Session Summary

1. Genetic programming provides a method for automatically creating a working computer program from a high-level statement of the problem
2. Computer programmes are represented by chromosomes that are structured as trees
3. Popular crossover operators: One-offspring and two-offspring
4. Popular mutation operators: Function node, terminal node, Swap, Grow, Gaussian, and Trunc
5. Applications of GPs include the following: Decision trees, Game-playing, Bioinformatics, Data mining and Robotics

Any Questions?

Thank You