AMO - Advanced Modeling and Optimization, Vol. 10, No. 1, 2008

Ordered Tree Alignment with Genetic Programming

S. Shafieian, H. Ahrabian *, A. Nowzari-Dalini
Center of Excellence in Biomathematics,
School of Mathematics, Statistics, and Computer Science,
University of Tehran, Tehran, Iran.
E-mail: {saeedsh,ahrabian,nowzari}@ut.ac.ir

Abstract

In this paper an algorithm for alignment of two ordered trees is presented. The algorithm is designed based on the genetic programming which is an extension of the genetic algorithms. In this approach, the two comparing trees are presented in parenthesis-form. Randomly, we create some pairs of trees based on these two trees as the initial population, and then by using a fitness function which is based on scoring the pairs of labels in tree nodes, the fitness of alignments of all pairs of trees is obtained. Trees with a better alignment are selected based on their fitness, then crossover and architecture-altering operations are performed on them to produce the new generation. These steps are performed until a predefined number of generations evolve. The crossover operator in our method is designed such that it replaces only similar sub-trees so that the resulting trees always represent correct alignments. Architecture-altering operator alters the architecture of a tree by increasing or decreasing the degree of a valid node in it.

Keywords: Tree, Alignment, Tree Alignment, Genetic Programming.

1 Introduction

Application of computer for solving problems has always been one of the major concerns for computer scientists. There are many problems in the field of computer science which are NP-hard or have a large polynomial time complexity. One of these problems is ordered tree alignment.

The tree is one of the major structures for representing data in computer science. A tree can be recursively defined as a finite set of some nodes in which a special node is root and the other nodes recursively denote other trees. An ordered tree is a rooted tree in which the children of each node are ordered. Since trees have many applications in representing and storage of data, comparing two trees has always been of great importance. As in the

*Corresponding author.
AMO - Advanced Modeling and Optimization. ISSN: 1841-4311

case of sequences comparisons, there are many ways to measure the similarity between two trees. For instance one could use tree edit distance, alignment distance, longest common sub-tree and smallest common super-tree [3, 6, 7, 12, 13].

Here, we consider the notion of alignment of trees as a measure of similarity between trees. This notion is a natural extension of alignment of sequences. In tree alignment, two given trees are first made isomorphic by inserting nodes labeled as $'-'$ and then the resulted trees are overlaid over each other.

Several researchers have studied comparing trees, some of them are as follow: In [7], Jiang et al. proposed the alignment of trees as a measure of the similarity between two labeled trees. Both ordered and unordered trees were considered in their approach. An algorithm was designed for comparing two ordered trees T_1 and T_2 with the time complexity of $O(|T_1|.|T_2|.(deg(T_1)+deg(T_2))^2)$, where $|T_i|$ is the number of nodes in T_i and $deg(T_i)$ is the degree of T_i, $i = 1, 2$. The algorithm is faster than the best known algorithm for tree edit when $deg(T_1)$ and $deg(T_2)$ are smaller than the depths of T_1 and T_2. For unordered trees, they showed that the alignment problem can be solved in polynomial time if the trees have a bounded degree and becomes MAX SNP-hard if one of the trees is allowed to have an arbitrary degree.

In [3], Hochsmann et al. presented a systematic treatment of alignment distance and local similarity algorithms on trees and forests built upon the tree alignment algorithm for ordered trees given by Jiang et. al [7] and extended it to calculate local forest alignments, which is essential for finding local similar regions in RNA secondary structures. Given two forests F1 and F2, the time complexity of their algorithm is $O(|F_1|.|F_2|.deg(F_1).deg(F_2).(deg(F_1)+deg(F_2))$ where $|F_i|$ is the number of nodes in forest F_i and $deg(F_i)$ is the degree of F_i.

In [6], Jansson et al. gave a fast algorithm for optimal alignment between two similar ordered trees with node labels. If there is an optimal alignment between the two input ordered trees which uses at most d blank symbols then their algorithm runs in $O(n \log n.(max\ \ deg)^4.d^2)$ time. In particular, if both trees are of bounded degree the running time reduces to $O(n \log n.d^2)$.

In [12], Wang et al. presented a dynamic programming algorithm for identifying the similar consensus (SC) (or the largest approximately common substructures) of two ordered labeled trees based on the alignment distance. They consider a substructure of a tree T to be a connected subgraph of T. Given two trees T_1, T_2 and an integer d, the SC problem is to find a substructure U_1 of T_1 and a substructure U_2 of T_2 such that U_1 is within distance d of U_2 and where there does not exist any other substructure V_1 of T_1 and V_2 of T_2 such that V_1 and V_2 satisfy the distance constraint and the sum of the sizes of V_1 and V_2 is greater than the sum of the sizes of U_1 and U_2. The proposed algorithm solves the SC problem in time $O(d^2.|T_1|.|T_2|.(G_1+G_2)^2)$, where $|T_i|$, $i = 1, 2$; is the size of tree T_i and G_i is the maximum degree of T_i.

The problem of ordered tree alignment if trees are large and have unbounded degrees, has a high computing time [7], so we proposed a genetic programming approach for solving the problem. In this paper our purpose is to find the optimal alignment of the trees i.e., the alignment in which maximum number of similar nodes overlay on each other.

Tree alignment has many applications that some of them include: Web data extraction, web pages comparison, programming languages codes comparison, tree comparisons in structured text databases, tree comparisons in image analysis, tree comparisons in automatic theory proving, tree comparisons in compiler optimization and comparison of RNA secondary structures [7].

In our approach we use genetic programming as a heuristic method for solving this problem. In this method a population of computer programs is generated using genetic rules then genetic operators are applied on the individuals of this population to generate the next population of programs. After producing many generations, the solution of the problem is obtained. In this approach which is a new method for tree alignment, an initial random population of trees is created based on the two initial trees. Then using genetic principles and crossover and architecture-altering operators, next generations which are evolved generations based on previous ones are created and finally one of the individuals is selected as the best alignment and solution of the problem. It is noted that, the main contribution of this paper is presenting a novel approach (genetic programming) for solving tree alignment problem. As mentioned, few polynomial solution are given for tree alignment problem, but for large size of trees, the time of the algorithm would be significantly much. For this reason, the approach of solving this problem by genetic programming can reduce this time.

The rest of this paper is organized as follows: In Section 2 an overview of the genetic programming is given, in Section 3 the problem of alignment of trees is presented, in Section 4 the proposed method for aligning trees is discussed and finally in Section 5 some experimental results of the method are presented. The conclusion of this work is given in Section 6.

2 Genetic programming

Genetic programming is in fact an extension of genetic algorithms which was first introduced by John Holland in 1975 [4]. Genetic programming is a domain-independent method that genetically breeds a population of computer programs to solve a problem. Specifically, genetic programming iteratively transforms a population of computer programs into a new generation of programs by applying analogs of naturally occurring genetic operations. As mentioned, the population individuals in genetic programming are programs which are each a candidate solution for the problem. In genetic programming the programs are expressed as syntax trees rather than as lines of code. Internal nodes of such a tree represent functions or instructions and leaves of the tree, also said terminals, represent independent variables of the problem or function arguments. Internal nodes can also be zero-argument functions and random constants [10].

The main difference between genetic programming and genetic algorithms is that population structures in this method are not fixed length characters like in genetic algorithms which encode problem solutions but are programs which are the problem solutions themselves. Genetic programming is capable of solving problems in many fields such as machine

learning, artificial intelligence, control, robotics, optimization, games theory, regression and conceptual learning [10].

2.1 Preparatory steps of genetic programming

Genetic programming solves a problem using programs which are in the form of trees (consisting operands and operators). Defining the problem in genetic programming is done by performing certain well-defined preparatory steps. The five major preparatory steps for the basic version of genetic programming are as follow [10]:

1. Defining the set of terminals (e.g., the independent variables of the problem, zero-argument functions and random constants).
2. Defining the set of primitive functions.
3. Defining the fitness measure (for explicitly or implicitly measuring the fitness of individuals in the population).
4. Defining certain parameters for controlling the run.
5. Defining the termination criterion and a method for designating the result of the run.

The first two preparatory steps specify the requirements to create the program. Each run of genetic programming is a competitive search among a diverse population of programs composed of the available functions and terminals. The identification of the function set and the terminal set for a particular problem (or category of problems) is usually a straightforward process. For some problems, the function set may consist of merely the arithmetic functions of addition, subtraction, multiplication, and division as well as a conditional branching operator. The terminal set may consist of the program's external inputs (independent variables) and numerical constants.

The third preparatory step specifies the fitness measure for the problem. The fitness measure defines what needs to be done. The fitness measure is the primary mechanism for specifying the problem's requirements to the genetic programming system.

The fourth and fifth preparatory steps are administrative. The fourth preparatory step specifies the control parameters for the run. The most important control parameter is the population size. Other control parameters include the probabilities of performing the genetic operations, the maximum size for programs, and other details of the run. The fifth preparatory step consists of specifying the termination criterion and the method of designating the result of the run. The termination criterion may include a maximum number of generations to be run as well as a problem-specific success predicate. The single best-so-far individual is then searched and designated as the result of the run.

2.2 Executional steps of genetic programming

After performing the preparatory steps for a problem, genetic programming can be run as a series of well-defined, problem-independent steps. Genetic programming typically starts with a population of randomly generated programs composed of the terminals and primitive functions provided in the first and second preparatory steps [10].

By applying operations analogous to naturally occurring genetic operations , genetic programming iteratively transforms a population of programs into a new generation of the population. These operations are applied to individual(s) selected from the population. The individuals are probabilistically selected to participate in the genetic operations based on their fitness, as measured by the fitness measure provided in the third preparatory step. The executional steps of genetic programming are as follow [10]:

1. Randomly creating an initial population (generation 0) of programs composed of available functions and terminals.

2. Iteratively performing the following sub-steps (called a generation) on the population until the termination criterion is satisfied:

   (a) Executing each program in the population and finding its fitness using the problem's fitness measure and performing the sub-step (b) for M times (M is the population size).

   (b) Creating new programs by applying following genetic operations with specified probabilities:

   Reproduction: Selecting one program based on fitness and copying it into the new population.

   Crossover: Selecting two programs based on fitness and creating new child programs for the new population by combining randomly chosen parts from two selected programs.

   Mutation: Selecting one program based on fitness and creating a child program for the new population by randomly mutating a randomly chosen part of the selected program.

   Architecture-altering operations: Selecting one architecture-altering operation from the available set of such operations and creating one new child program for the new population by applying the chosen architecture-altering operation to the selected program.

3. After the termination criterion is satisfied, the single best program in the population produced during the run (the best-so-far individual) is searched and designated as the result of the run. If the run is successful, the result may be a solution (or approximate solution) to the problem.

There are some methods for creating the initial population of the programs. In one of them that is called "Full" initialization method, program tree nodes are taken from the

function set until a maximum tree depth is reached. Beyond that depth only terminals can be chosen. The other method which is called "Grow" initialization method allows the selection of nodes from the whole primitive set until the depth limit is reached [10].

3 Tree alignment problem

In this section we formally define [3] the tree alignment problem. Suppose the possible set of labels of the two trees is denoted by Σ and is called the alphabet. Also suppose that there is a special symbol called space and denoted by $'-'$ which is not in the alphabet Σ. In order to include label $'-'$ in the alphabet we define Σ^- as $\Sigma \cup \{-\}$. Paired alphabet is also defined as $\Sigma^2 = \Sigma^- \times \Sigma^- \setminus \{(-,-)\}$ for excluding the paired label $(-,-)$ and $\Sigma^{-2} = \Sigma^- \times \Sigma^-$ for including the special pair $(-,-)$. The scoring function Ω is defined as $\Omega : \Sigma^{-2} \rightarrow \mathrm{R}$. So the score of an alignment A of the trees T_1 and T_2 can be defined as:

$$S(A) = \sum_{\text{for each node } v \text{ in } A} \Omega(label(v)),$$

in which $label(v)$ denotes the label of the node v.

An alignment A of two trees T_1 and T_2 is first making the trees isomorphic by inserting nodes labeled $'-'$ in each of them and then overlaying the resulting trees over each other. The goal of the alignment problem is to find the optimal alignment of the two trees in which $S(A)$ is maximum. It should be noticed that here, isomorphism is without considering node labels in the two trees.

As an example of tree alignment, consider the two trees in Fig. 1(a). The isomorphs of them are shown in Fig. 1(b). Fig. 1(c) shows the optimal alignment of the two trees.

In our new approach for aligning ordered labeled trees, genetic programming receives the two aligning trees as its input and after evolving the populations of program trees for several generations, produces the (near) optimal alignment of the two trees as its output.

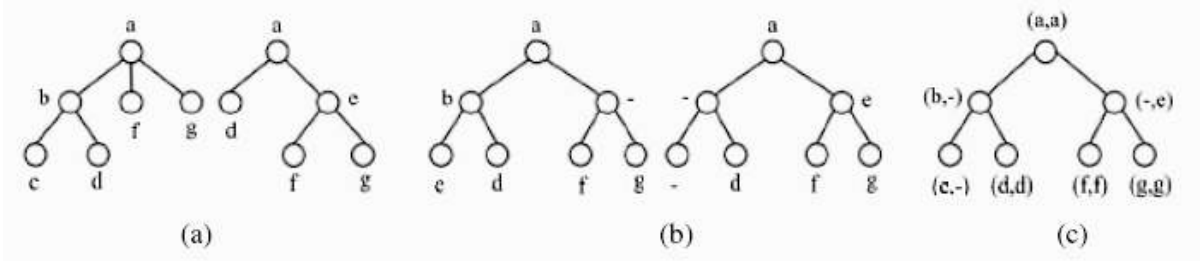


Figure 1: (a) Two sample trees. (b) Isomorphs of the two trees by inserting $'-'$ nodes. (c) The optimal alignment of the two trees.

4 Tree alignment with genetic programming

In this section the new proposed method for aligning two ordered trees with genetic programming is presented. The preparatory steps for genetic programming to solve the problem, crossover and architecture-altering operations and the other details of the proposed method are discussed.

4.1 Steps of genetic programming for tree alignment

In this sub section we discuss the preparatory steps that must be performed before the run of the genetic programming begins. As mentioned before, five steps must be performed in order to prepare the requirements of the program run. These steps are as follow:

1. **Specifying the terminals set:** The members of the terminals set are the members of the alphabet $\sum^{-2}$, i.e., the members of the terminals set are (α, β) such that $\alpha, \beta \in \sum^{-}$.
2. **Specifying the functions set:** In our method the functions set is the null set (), i.e., in generating the program trees in the proposed method the primitive functions are not used.
3. **Fitness measure:** The fitness measure in this method is alignment score, S.
4. **Parameters for controlling the run:** These parameters include population size, probability of the reproduction operator, probability of the crossover operator and the probability of the architecture-altering operations.
5. **Termination criterion and result of run:** The termination criterion for this problem is specifying the maximum number of generations the program must run. The best generated program through the run is designated as the result of the run (optimal alignment).

It is clear that the generated trees for solving this problem are only composed of terminal nodes or labels alphabet, i.e., they are trees with (α, β) labels. Programs in this method are alignments of the two trees and for determining the fitness of each one, its alignment score is computed. The higher the alignment score, the fitter is the program having it. The only possible termination criterion for this problem is to specify a maximum number of generations for the genetic programming to evolve. Since the score of the optimal alignment is not known and cannot be well estimated too, specifying any termination criterion based on the score of the generated alignments is impossible.

4.2 Initialization of the population

Initialization always plays a major role in evolving programs in genetic programming, since if good initialization is done, the convergence of the population programs to the problem solution will be accelerated.

The method of initialization (creating generation 0) for the alignment problem is that first M (M, population size) pairs of trees with one copy of T_1 and one copy of T_2 in each of them are created. Then in each of the paired trees, point(s) of inserting space(s) is randomly selected and the nodes labeled $'-'$ are inserted in them. After finishing this operation, the two trees in each M created pairs are made isomorphic by inserting certain additional $'-'$ labeled nodes and then overlaid on each other. In this way, each resulted tree becomes an alignment of the two first trees and these M alignments make the initial population of the problem. Fig. 2 shows a sample initialization for one of the M programs in generation 0. As it is shown in Fig. 2(b) two $'-'$ nodes are both inserted as the children of the node $'a'$. In these random insertions, in the first tree the node $'-'$ has become the parent of node $'f'$ and in the second one the parent of node $'d'$. Thus M numbers of Fig. 2(d) trees are created as our initial population.

4.3 Problem specific operators

In order to solve the problem of the alignment of two ordered trees with genetic programming approach, the defined operators in genetic programming cannot be utilized in their initial and simple form. Since the mutation operator replaces a randomly selected sub-tree by a randomly generated sub-tree in an alignment tree, it will certainly generate invalid alignment trees and therefore its infeasibility for use in this problem is clear. In the following section we will discuss the infeasibility of the original crossover operator and present the alternative operators for it.

4.3.1 Crossover operator

The crossover operator which is the main operator in genetic programming has the task of randomly exchanging the sub-trees of the generated trees in the population. This operator with its original and simple form cannot be used in solving the alignment problem because it violates the closure property of the operators in genetic programming. Closure property means that every operator which is performed on program(s) in the population must result in valid program(s) such that each represents a valid and possible solution for the problem. It can be easily shown that the crossover operator violates the closure property of the operators because it randomly exchanges any sub-tree of the first parent by any sub-tree of the second one so the resulting trees can represent the alignment of two trees other than the two of the problem. For solving this problem one way is to apply a constraint on crossover operator to keep the order of the nodes in the trees and always generate valid trees for the problem.

Definition 1 *Consider two trees T_1 and T_2 with paired labels (α, β). We call two sub-trees T'_1 and T'_2 of the two trees T_1 and T_2 "similar", if by (a) deleting the children labeled $(-,-)$ in the two sub-trees, (b) relabeling the children labeled $(-,\alpha)$ and $(\alpha,-)$ to α and labeled (α,β) to $\alpha\beta$ and (c) ignoring the root labels in the two sub-trees, the two sub-trees become isomorphic. We denote the root of the new sub-trees by R. For example, two similar sub-trees and the procedure for determining their similarity are shown in Fig. 3.*

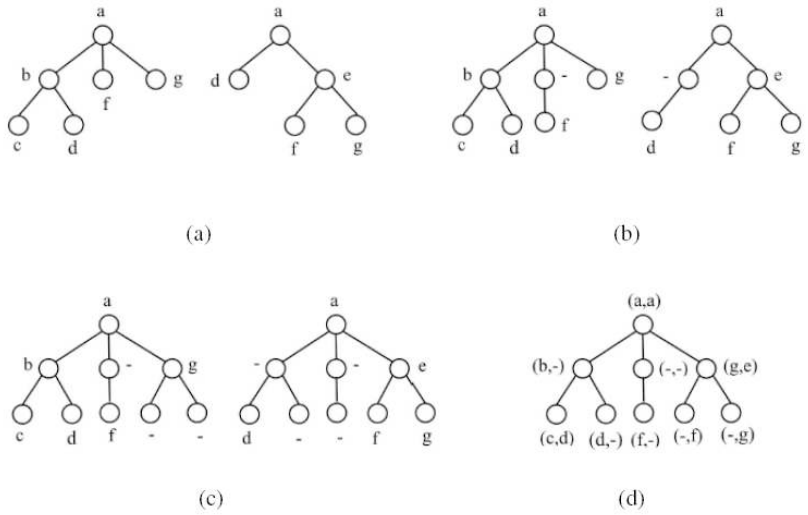


Figure 2: (a) The two first trees. (b) The two trees after inserting $'-'$ nodes. (c) The two isomorphic trees. (d) The resulting alignment.

As it is shown in Fig. 3, the two sub-trees are made isomorphic by the procedure defined in Definition 1. It is clear from this example that in two similar sub-trees the order of the nodes is equal, excluding the nodes labeled $(-)$. In fact the two strings obtained from level-order traverse of two similar sub-trees must be equal, excluding the $'-'$ characters.

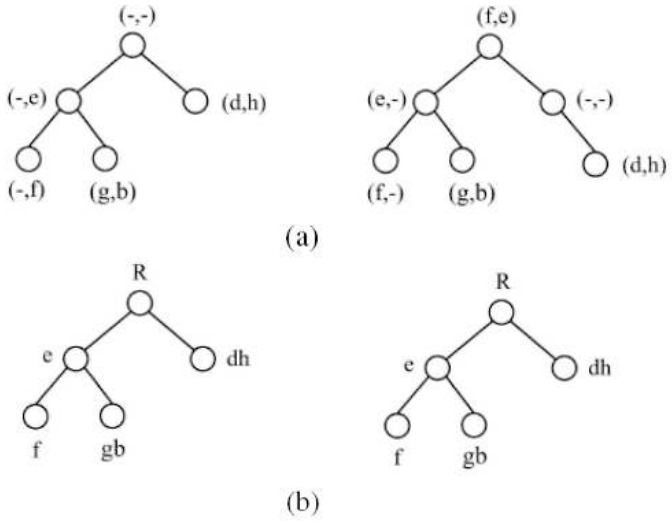


Figure 3: (a) Two sample sub-trees. (b) Converting them to verify their similarity.

In the proposed method for crossover, only similar sub-trees of the program trees are exchanged. In this way, the closure property of the operators is satisfied for the crossover operator and all the trees generated by this operator will represent possible and valid alignments for the problem. The reason for ignoring the roots of the two sub-trees in showing their similarity is that in this case the number of similar trees in the two trees is incremented. The results of the experimental tests show that this way of crossing over the trees well provides different combination of the trees for searching the optimal alignment.

4.3.2 Architecture-altering operators

The architecture-altering operator is the operator that only exists in genetic programming and not in genetic algorithms. This operator in accompany with crossover operator, can play an important role in generating the possible alignments and acts somehow like the mutation operator, even in the experimental results of this method there were cases in which the role of the architecture-altering operator was stronger than the crossover one. A possible reason for this can be the special nature of the trees in this problem.

The proposed architecture-altering for this problem can be categorized as two types: degree increasing operator and degree decreasing operator. Now we discuss these operators.

1. **Degree increasing architecture-altering operator**

   This operator randomly finds (if exists) a node i labeled $(-,-)$ in the tree and deletes it from the tree. By deleting this node, its children come up one level and become the children of the parent of the deleted node $(-,-)$. This operation increases the degree of the parent node by $deg(i) - 1$. Fig. 4 shows the function of this operator.

2. **Degree decreasing architecture-altering operator**

   This operator is like the previous one but inserts a new node in the tree. This operator randomly finds (if exists) a node i with a degree of three or more and then randomly selects m consecutive sibling of the node. After this, a new node labeled $(-,-)$ is inserted into the tree as the parent of these m siblings. This operation

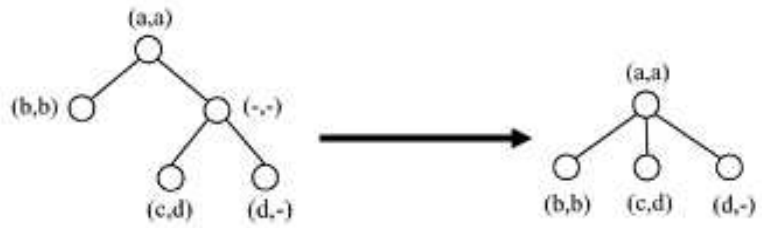


Figure 4: Degree increasing architecture-altering operator increases the degree of node (a,a) from 2 to 3.

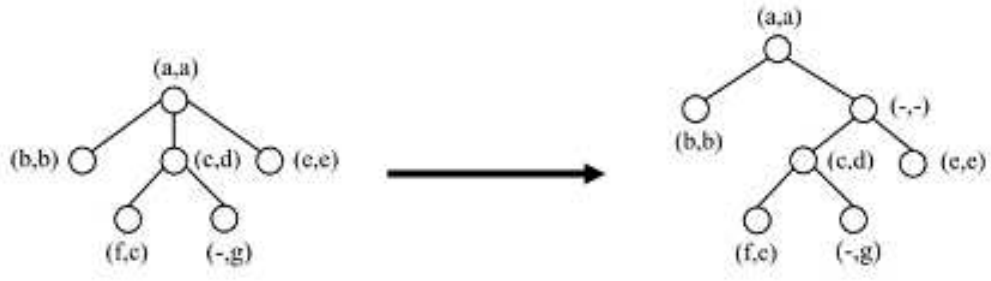


Figure 5: Degree decreasing architecture-altering operator decreases the degree of node (a,a) from 3 to 2.

decreases the degree of the node i by $m-1$. The function of this operator is shown in Fig. 5.

It is clear that the architecture-altering operations do not violate the closure property of the operators since they only insert/delete nodes labeled $(-,-)$ and these nodes do not have any effect in the order of alignment trees nodes.

4.4 Probabilities of the operators

As it is shown in the Fig. 1, each of the operators used in genetic programming has its own specific selection probability such that sum of the probabilities of all the operators must be 1. In the proposed method for aligning trees, since we have three operators reproduction, crossover and architecture-altering, therefore sum of the probabilities of these three operators must be equal to 1. Normally the probability of the crossover operator is set higher than other operators due to its major role, but generally, specifying the probabilities of the operators is an experimental work and cannot be done precisely for every problem. One possible solution for resolving this problem is to dynamically alter the probabilities of the operators during the run of genetic programming. In this method after evolving one or some generations, if the average fitness of the programs generated by an operator is increased, its probability will also increase and if the average fitness is decreased, its probability will decrease too. However the main drawback of this method is that it highly increases the run time of the program since in each generation it takes $O(M)$, (M is the number of programs in each generation) time to compare the programs generated in that generation with the previous generation and in total $O(G \times M)$, (G is the number of generations) time is added to the run time of the program. Considering the above argument and the experimental results obtained, there was no need to use dynamic probabilities, so static probabilities were used.

4.5 Formalization of the genetic programming method

As mentioned before, genetic programming is a search method that searches the space of the programs. If the state space of this search is illustrated, it can be seen that at first, the population of the programs is somewhat like a cloud of randomly distributed points but after evolving some generations the shape of this cloud is changed and moves in the search space following a well-defined trajectory [10].

Genetic programming schema is syntactically a tree in which there are some special nodes called "don't care". These nodes exactly specify a terminal node or a primitive function. Semantically, a schema represents all the programs that match its size, shape and defining (non-"don't care") nodes. For example the schema H = (DON'T CARE $\times$($+y$ DON'T CARE)) represents the programs $(+x(+yx)), (+x(+yy)), (\times x(+yx))$, etc [10].

Suppose H is a schema and $\delta(H,t)$ is the probability that a newly generated program at generation t belongs to H. Since there are three operators in our method, we have [10]:

$\delta(H,t) = P[A$ program in H is obtained via reproduction]
$+ P[$Two children matching H are produced by crossover]
$+ P[A$ program matching H is obtained via architecture-altering].

Then supposing that reproduction is performed with probability P_r, crossover with probability P_c and architecture-altering with probability P_a, $(P_r + P_c + P_a = 1)$, we will have [10]:

$\delta(H,t) = P_r \times P[A$ program in H is selected for cloning]
$+ P_c \times P[$Parents and crossover points are such that the two children match $H]$
$+ P_a \times P[$Parent and arch-altering operator are such that child matches $H]$.

Clearly, the first probability in this expression is simply the probability of selection for the members of the schema H by for instance fitness-proportionate selection. So,

$P[$Selecting a program in H for cloning$] = P(H,t)$ [10].

Now we must calculate the second probability in the expression. This is the probability that parents have the shapes and contents compatible with H and also the crossover points in the parents are selected such that they exactly provide the conditions for generating such children through exchanging two *similar* sub-trees.

A notion that helps simplify the calculation of this problem is that, although the probability of choosing a particular crossover point in a parent depends on the actual size and shape of such a parent, the process of crossover point selection is independent from the actual primitives present in the parent tree. Since in the proposed method for aligning two trees, all the generated trees in the population are isomorphic and have exactly the same structure, the probability of selecting a crossover point in one tree exactly equals the probability of selecting the point in any other tree [10]. By applying this observation we

can write:

P[Parents and crossover points are such that the resulting children match H] =
$\sum_{i,j}(P$[Sub-trees resulted from crossover points are similar]
$\times P$[Parents are such that if crossed over in points i, j produce two children in H]).

The first probability cannot be calculated easily because two similar sub-trees do not necessarily have the same structure and only by converting them by the procedure in Definition 1, similarity or dissimilarity of them can be recognized.

Now we must calculate the selection probability (for crossover) of the parents such that they have nodes that if crossed over in the specified cross over points, produce two children in schema H. The children produced via crossover are similar if and only if the two exchanged sub-trees along with the two parents have the structure and nodes compatible with schema H. This probability can be calculated but needs several other concepts and notations which are beyond the scope of this paper.

For the architecture-altering operation, we can express the probability that the resulted child matches the schema H as follows:

P[The parent and the architecture-altering operator are such that the produced child is in H] =
$\sum_{\text{For all the parents}}$([Selected tree can be degree increased/decreased]
$\times$[Degree increasing/decreasing is performed such that the resulting tree is in H]).

The first probability means that if the selected tree is supposed to increase, it must have at least one node labeled $(-,-)$ and if it wants degree decreasing it must have at least one node with a degree of three or higher. The second probability can also be calculated but is beyond the scope of this paper.

4.6 Alignment algorithm

In this section the general algorithm for alignment of two ordered trees using genetic programming is given. This algorithm is shown in Algorithm 1. As it is shown in this algorithm, initialization is done first and a number of random programs each representing a possible alignment are generated. Then until the number of evolved generations is less than or equal to the maximum number of generation specified, in each generation the reproduction, crossover and architecture-altering operators are performed and each of them generate one, two and one programs for the next generation respectively. This procedure lasts until the generated programs for the next generation are equal to the specified population size for each generation. At the end of each generation, the best generated program is saved and at the end of the run the best generated program is selected among these saved programs and is recognized as the best alignment and solution of the problem.

5 Experimental results

In this section some of the experimental results obtained from the implementation of the proposed method are presented. For displaying trees in the implementation, we have used the list-representation method of the trees. But for further readability, the trees are displayed in their original form here. In the Fig. 7 the optimal alignment obtained for some pair of sample trees along with their corresponding parameters are shown.

In the sample trees shown in Fig. 7 which are all classic and valid samples of the tree alignment, the alignments obtained via genetic programming method is exactly identical to the optimal alignments obtained for these trees via classical and non-heuristic methods and genetic programming has succeeded in producing the optimal alignment in all the cases.

6 Conclusion

In this paper a new method for aligning ordered trees by applying genetic programming was presented and discussed. First, Five preparatory steps for genetic programming were presented. The function set was null and the programs were the same as alignment trees. The fitness function in this method was a scoring mechanism over alignment trees nodes and the different parameters for controlling the run were also discussed. The termination criterion for this method was specifying a maximum number of generations to evolve the programs and the reason for such a termination criterion was analyzed. Next, the initialization method for the population was presented that consisted of paired isomorphic trees generated by inserting $'-'$ symbols in the trees. Then the most important part of the proposed method, i.e., the special operators designed for solving the alignment problem was discussed. These operators consisted of crossover which exchanged random similar sub-trees of two parent trees and the architecture-altering operators which had increasing/decreasing types, the former increased the degree of a tree by deleting a node labeled $(-,-)$ and the later decreased the degree by inserting a node labeled $(-,-)$.The best generated alignment tree in each generation was saved and finally the best alignment was searched through these saved alignments and was designated as (near) optimal alignment for the two input trees.

Some formalizations of the method were also presented using genetic programming theorem. At the end, the result of testing the method on several sample trees were showed in all of which obtained alignments were the optimal alignments for the trees.

For continuing this research, multiple alignments of ordered trees, dynamic changing of control parameters during the run and defining further operators for the genetic programming can be suggested.

Acknowledgements

This research was supported by a grant from Iran's Supreme Council of Information and Communication Technology (SCICT).

```
Algorithm Genetic - Alignment
Begin
    Gen := 0;
    Create initial random population of programs;
    while Gen ≤ number of maximum generations do begin
        Apply fitness measure on all the programs in the population;
        Gen := Gen + 1;
        Count := 0;
        while Count < Popsize do begin
            Select a genetic operation based on operations probabilities;
            case of reproduction do begin
            Select one program based on fitness;
            Perform reproduction;
            Copy selected program into new population;
            Count := Count + 1;
        end;
        case of Crossover do begin
            Select two programs based on fitness;
            Perform crossover;
            Insert two children into new population;
            Count := Count + 2;
        end;
        case of Architecture - Altering do begin
            Select Degree Increasing or Decreasing operation based on their
            probabilities;
            Select one program based on fitness;
            Perform Architecture - Altering operation;
            Count := Count + 1;
        end;
        Store the best program created in this generation;
        end;
    end;
    Select the best program among the stored programs as the optimal alignment;
End;
```

Figure 6: Genetic programming algorithm for aligning two ordered trees.

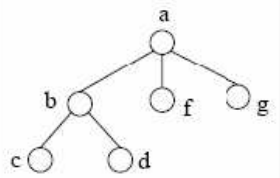
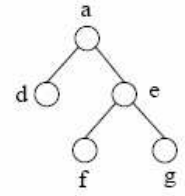
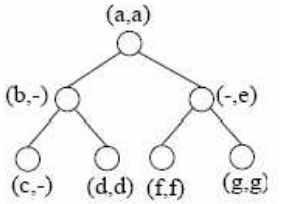
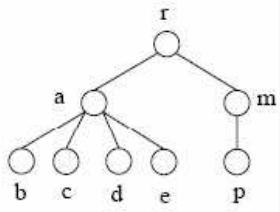
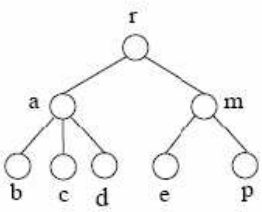
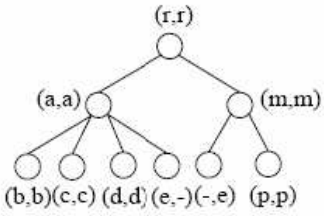
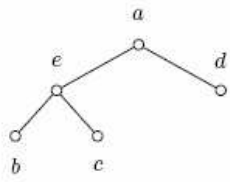
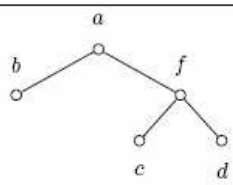
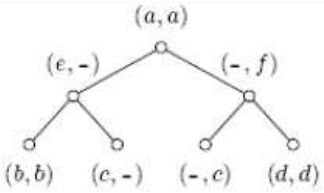
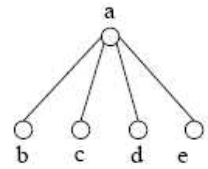
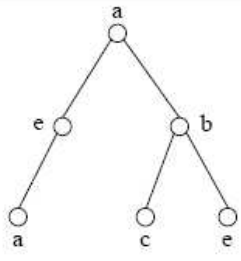
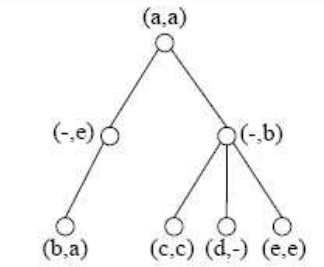
Tree 1	Tree 2	Best resulting alignment	Pop. Size	NO. Gen.	Rep. %	Cross. %	Arch. %
			80	50	5	90	5
			100	20	10	30	60
			70	50	10	70	20
			50	40	20	30	50

Figure 7: Some pairs of sample trees with their obtained alignments via genetic programming, the corresponding parameters are also shown.

References

[1] A. A. Freitas, A genetic programming framework for two data mining tasks: classification and generalized rule induction, in: *Proc. 2nd Annual Conference on Genetic Programming*, Morgan Kaufmann, San Francisco, 1997, pp. 96-101.

[2] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, 1989.

[3] M. Hochsmann, T. Toller, R. Giegerich, and S. Kurtz, Local similarity in RNA secondary structures, in: *Proc. IEEE Computational Systems Bioinformatics Conference*,

IEEE Computer Society Press, 2003, pp. 159- 168.

[4] J. H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, 2nd ed., University of Michigan Press, Cambridge, 1992.

[5] Y. J. Hu, GPRM: A genetic programming approach to finding common RNA secondary structure elements, *Nucleic Acids Res.* 31 (2003), 3446-3449.

[6] J. Jansson and A. Lingas, *A fast algorithm for optimal alignment between similar ordered trees*, Lecture Notes in Computer Science, vol. **2089**, pages 232-240, Springer-Verlog, Berlin, 2001.

[7] T. Jiang, L. Wang, and K. Zhang, Alignment of trees - an alternative to tree edit, *Theo. Comp. Sci.* **143** (1995), 137-148.

[8] J. K. Kishore, L. M. Patnaik, V. Mani, and V. K. Agrawal, Application of Genetic Programming for Multicategory Pattern Classification, *IEEE Trans. Evol. Comput.* **4** (2000), 242-258.

[9] J. R. Koza, F. H. Bennett, D. Andre, and M. A. Keane, *Genetic Programming III: Darwinian Invention and Problem Solving*, Morgan Kaufmann, San Francisco, 1999.

[10] J. R. Koza and R. Poli, A Genetic Programming Tutorial, in: *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society Press, San Francisco, 1996.

[11] C. Notredame and D. G. Higgins, SAGA: Sequence alignment by genetic algorithm, *Nucleic Acids Res.* **24** (1996), 1515-1524.

[12] J. T. L. Wang and K. Zhang, Identifying consensus of trees through alignment, *Inform. Sci.* **126** (2000), 165-189.

[13] J. T. L. Wang, K. Zhang, and C. Y. Chang, Identifying approximately common substructures in trees based on a restricted edit distance, *Inform. Sci.* **121** (1999), 367-386.