Two-Step Genetic Programming for Optimization of RNA Common-Structure

Jin-Wu Nam[1, 2], Je-Gun Joung[1, 2], Y.S. Ahn[4], and Byoung-Tak Zhang[1, 2, 3]

[1] Graduate Program in Bioinformatics
[2] Center for Bioinformation Technology (CBIT)
[3] Biointelligence Laboratory, School of Computer Science and Engineering
Seoul National University, Seoul 151-742, Korea
[4]Altenia Corporation, Korea
{jwnam, jgjoung, ysahn, btzhang}@bi.snu.ac.kr

Abstract. We present an algorithm for identifying putative non-coding RNA (ncRNA) using an RCSG (RNA Common-Structural Grammar) and show the effectiveness of the algorithm. The algorithm consists of two steps: structure learning step and sequence learning step. Both steps are based on genetic programming. Generally, genetic programming has been applied to learning programs automatically, reconstructing networks, and predicting protein secondary structures. In this study, we use genetic programming to optimize structural grammars. The structural grammars can be formulated as rules of tree structure including function variables. They can be learned by genetic programming. We have defined the rules on how structure definition grammars can be encoded into function trees. The performance of the algorithm is demonstrated by the results obtained from the experiments with RCSG of tRNA and 5S small RNA.

1 Introduction

The universe of functionally important non-transcribed RNAs is rapidly expanding but their systematic identifications in genomes remain challenging. A number of non-coding RNA (ncRNA) genome screening approaches have been reported in the literature, including composition, sequence and structure similarity-based and comparative genomics [1], [2], [3]. For example, some focus on secondary structure prediction for a single sequence and some aim at finding a global alignment of multiple sequences. Yet other groups predict the structure based on free energy minimization or make comparative sequence analyses to determine the structure. However, these methods are limited to sets of short sequence or require the structural information of some known sequences [4]. Also these approaches usually use only positive data to learn a model and there are no doubts that these methods have a weak constitution for noise. To overcome these limitations, we have tried to develop a general method. One of these general methods is learning and optimization of RNA common structure [5]. Most ncRNAs have low sequence similarity but high structure similarity [6] making it suitable for using common-structure for identification of putative ncRNAs [4], [7].

Searching for these common-structures is a crucial step to identify similar new functional ncRNAs in genome. Recently, some researches have various methods for finding common-structure. For instance, hidden Markov models and genetic algorithm have been applied to search common structure [7], [8]. However, these methods have two great limitations: first, they are not at the level to use as applications, and second, they are not based on learning from structure, thus, unable to find distant homologies.

On the other hand, some research groups have developed the structural grammar, which is context free type [9], [10], [11]. Using the structural grammar, we can easily and formally express RNA secondary structures. Macke et al. introduced the structure definition language, which is used in RNAMotif [12]. The language can not only easily represent various RNA secondary structure elements such as loop, bulge, stem and mispair, but also include some sequence features such as conserved motif and mismatch number. This structure definition language allows abstraction of the structural pattern into a 'descriptor' with a pattern language so that it can give detailed information regarding base pairing, length and motif. In addition, these structural grammar and structure definition language can be applied to learning the common-structure. The results of the learning represent RNA common-structural grammars (RCSGs). There are various approaches to learning the RCSGs but we apply genetic programming, a kind of evolutionary algorithms, with focus on two steps, structural learning step and motif learning step.

Genetic programming is an automated method for creating a working genetic program, which is called individual and generally represented by tree structure, from a high-level problem statement of a problem [13]. A genetic tree consists of elements from a function set and a terminal set. Function symbols appear as internal nodes. Terminal symbols are used to denote actions taken by the program. Genetic programming does this by genetically breeding a population of genetic programs using the principles of Darwinian natural selection and biologically inspired operations. Genetic programming uses crossover and mutation as the transformation operators, which can endow variation to genotype, to change candidate solutions into new candidate solutions [13].

We optimize the RCSG of fly tRNA and eukaryotic 5S small rRNA using our application of genetic programming and present the result of evaluation of our method for identifying mouse tRNA and 5S small rRNA.

2 Materials and Methods

2.1 Genetic Programming to Optimize RNA Common-Structural Grammar

To identify the putative ncRNAs in genome database, we have tried to find common-structure conserved among ncRNAs. We have implemented a program for learning the common-structure, which is devised by genetic programming. We call the program esRCSG, evolutionary search for RNA Common-Structural Grammar. The algorithm of esRCSG is summarized as a pseudo-code in Figure 1. The algorithm

```
begin                          /* Structural Learning */
        $t = 0$                /* generation */
        initialize $P(t)$      /* population */
        convert $P(t)$         /* tree to grammar*/
        evaluate $P(t)$
        while (not termination-condition) do
        begin
           $S = S +$ above($P(t)$)                   /* Top group for Seq learning*/
           $t = t+1$
           select $P(t)$ from $P(t-1)$               /* selection */
           crossover-mutate $P(t)$ except Best       /* genetic operators */
           convert $P(t)$
           evaluate $P(t)$                           /* fitness function */
           if (local search)
               while (not termination-condition) do
                   $j = j + 1$
                   $P_j(t)$ = mutate $P(t)$
                          if (evaluate $P(t)$ < evaluate $P_j(t)$ )
                                 $P(t) = P_j(t)$
        end
end
$w$ = wordwise(training data)
begin                                        /* Learning of Sequence */
        $t = 0$                              /* generation */
        initialize $S$(t) from $S$ with $w$  /* population */
        convert $S(t)$
        evaluate $S$(t)
        while (not termination-condition) do
        begin
           $t = t+1$
           select $S(t)$ from $S(t-1)$                  /* selection */
           mutate $S(t)$ for only seq. except Best      /* genetic operators */
           convert $S(t)$
           evaluate $S(t)$
        end
end
```

Fig. 1. The pseudo-code for RCSG optimization algorithm. The algorithm consists of two steps: (1) learning RCSG from a set of training data, known miRNA precursors without sequence-related variables (such as “seq” and “mismatch”); (2) Optimizing RCSG which is learned in structural learning by incorporating the sequence-related variables. The "word-wise“ method randomly splits sequences of training data set into 7-mer words. Initialization of this step is accomplished by incorporating the words into RCSGs that are learned in structural learning.

consists of two-step; a structural learning which optimizes only tree structure of grammar without sequence and a learning of sequence that specifies RCSGs of structural learning by incorporating a word, fragment of sequence, into the RCSGs. Both steps (1) (2) are implementations (or instances) of genetic programming and both share many of the procedures. The common procedures of (1) and (2) can be described as follows: (a) initialize the population with randomly generated trees; (b) convert all function trees into structural grammars; (c) calculate the fitness, specificity, sensitivity, and complexity for all grammars with the positive and negative training data set; (d) evaluate all structural grammars in terms of the sensitivity, specificity and complexity; (e) using ranking selection, select function trees that will generate offspring (next generation); (f) apply variations, such as mutation and crossover, with the selected function trees; (g) Iterate steps (b) through (f) for the user-defined number of generations. There are two differences between two steps; the structural learning includes a local search procedure; the learning of sequence uses the wordwise method and utilizes the word to initializes the population. In addition, structural learning step uses mutation and crossover as variation operators to change a tree structure but learning of sequence uses only mutation to change the sequence and the number of mismatch without changing tree structure.

Individual representation. In order to convert structural grammars into function trees, we have defined the function f1, f2 and root as shown in Figure 2a. These functions can be formulated by some expression rules (Figure 2a). Therefore, using the expression rules, we can represent the structural grammars (Figure 2c) as function trees (Figure 2b). Thus, we can use the function trees as individuals of genetic programming because one of the characteristics of genetic programming is that individuals are represented by tree structures.

To avoid creation of invalid structural grammars, function trees have some constraints on the order of the function and the terminal node. First, f2 function should not appear consecutively in the same depth of the tree, contiguous f2 functions can be considered as only one. Second, f2 function can only appear as terminal node to terminate recursive generation of function tree. Finally, variables 'minlen' and 'maxlen' should always come in pair and should not coexist with variable 'len.'

Population initialization. An initial population is randomly created with some constraints about function tree as described above. The initial population contains various function trees because there is no limitation in the number of nodes and the width of tree. That makes it possible to cover a wide range for searching start point. The broad coverage at start point is one of the major reasons the esRCSG is efficient for searching optimal solution.

Fitness function. The fitness function (Equation 1) is defined by using specificity, sensitivity and the complexity that are defined at Equation (4).

$$Fitness = spC * Specificity + stC * Sensitivity - Complexity \quad (1)$$

$$spC + stC = 1 \quad (2)$$

Two parameters, *spC* and *stC* were added as a way to regulate the effects of specificity and sensitivity. To normalize the fitness, the sum of *spC* and *stC* is always 1 (Equation 2). The parameters decide the trade-off between the specificity and the sensitivity on the fitness function.

The complexity, which is a negative factor to the fitness function, controls the growing of the tree structures. Without the complexity term, the size of the tree does not converge to a minimum description length where the tree has best efficiency and can grow infinitely in evolving the trees. To overcome that size problem, we make

$Comp_i^j$ include the node number and the depth of *j*th tree on *i*th generation (Equation 3). Equation (4) describes the definition of *Complexity* of *j*th tree on *i*th generation.

(a)

Fun	Grammar	Variable
f1	h5 (f1 or f2) h3	minlen/maxlen, len, mispair, seq, mismatch
f2	ss	minlen/maxlen, len, seq, mismatch
root	descr	

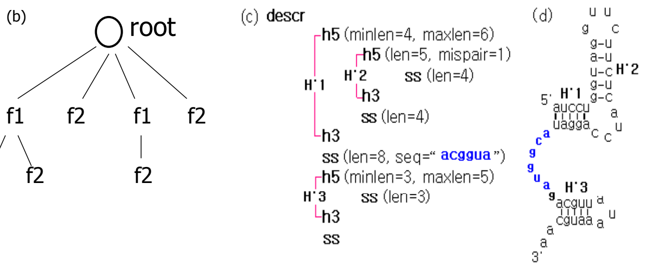


Fig. 2. (a): Function f1 generates recursively structural grammar, including one helix structure and either f1 or f2 as next deviation. Function f2 only represents ss (single strand), which means loop, bulge and single strand. Both f1 and f2 contain some variables, which measure structural information such as the length of helix (len), the number of pair (mispair) and mismatch, and sequence (seq). (b): A function tree to which can be converted into structural grammar (c): The child nodes of root in (b) conform to the first indentation of (c) and the nodes of second depth conform to the second indentation of (c). One helix that consists of the pair of h5-h3, h means helix, 5 and 3 mean 5' end and 3' end) (d): Secondary RNA structure is represented by structural grammar (c). H1, H2 and H3 in (c) and (d) are helix structures.

$$Comp_i^j = TreeDepth_i^j \times 10 + NodeNum_i^j \tag{3}$$

$$Complexity_i^j = \frac{1}{(NS+PS)^2} \times \frac{Comp_i^j}{Comp_{i-1}^{best}} \tag{4}$$

where *Complexity* is normalized by square of the number of training data set (*NS* + *PS*). *NS* is the number of negative training data set (see Table 1) and *PS* is the number of positive training data set (see Table 1). $Complexity_i^j$ depends on $Comp_{i-1}^{best}$ which is *Comp* of the best individual (tree) on (*i*-1)th generation. That dependency makes the trees have the minimum *Comp* as the progress of generation. Finally, the size of the best tree on last generation is converged into minimum length. This is the effect of the Occam's razor [14] of the complexity factor in our fitness function.

Variation. The variation operators are applied so that each descendent will have a different tree structure relative to the parents. The first operator to perturb the tree is the mutation. The mutation changes the value of the function variable by a random variable drawn from Poisson distribution. The crossover exchanges each sub-tree in two parent trees via single-point recombination to generate two new offspring trees. In the crossover, two parent function trees are selected at random from the population and then each single recombination point is selected at random from the each parent.

2.2 Dataset

We applied our algorithm to tRNA sequences of *Drosophila melanogaster* (available at http:// www.ncbi.nlm.nih.gov/entrez) and the eukaryotic 5S small rRNA (available at http://rose.man.poznan.pl/5SData/5SRNA.html) as training and test sets (Table 1).

Table 1. Training and Test Dataset.

	Positive Dataset	Negative Dataset
Training set	50 tRNAs	200 sequences; hairpins, pseudo-knots, IRE, consecutive hairpins, miRNA precursors, bulges, internal loops, rRNA, and fragment of mRNA
	50 rRNAs	200 sequences; hairpins, pseudo-knots, IRE, consecutive hairpins, miRNA precursors, bulges, internal loops, tRNA, and fragment of mRNA
Test set	100 tRNAs	290 sequences, negative data like training set
	100 rRNAs	290 sequences, negative data like training set

Negative data set consists of some classes; linear sequences that are extracted from mRNA and simple secondary structure elements such as IRE (Iron-responsive element), pseudo-knots, hairpins, bulges and internal loops. Also negative data set includes ncRNAs except itself. The negative data set is a negative rudder to decide the direction of learning. Hence, the negative data have to be collected carefully and the distribution by the classes of those has to be considered.

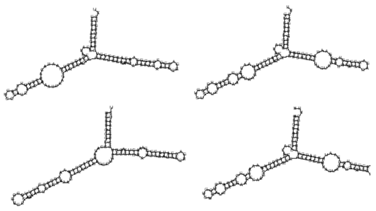


Fig. 3. The secondary structures of eukaryotic 5S small rRNA, these structures were drawn by RNAfold of Vienna package.

3 Results

3.1 RCSG of 5S Small rRNA by One-Step Learning

We show the result of 5S small rRNA, one of ncRNAs, preceding the result of tRNA. First, we looked into the secondary structure of four 5S small rRNAs within the positive training set, which are eukaryotic 5S small rRNAs. Figure 3 shows the secondary structure, which are predicted by RNAfold, of the rRNAs. As the results show, the common structure of 5S small rRNAs in training data should have two stem structures. To optimize RCSG of 5S small rRNA, esRCSG uses the positive training set (= 50) and negative training data set (= 200) as Table 1. The optimized RCSG is represented in the Figure 4(a) and is learned using only structural learning step. It is opti mized under the parameters as Figure 4(b) and it is the best solution learned from four trainings (Sensitivity = 0.94;

(a)

```
descr
   ss(len=30)
   h5(len=7, mispair=5)
         h5(len=1, mispair=1)
               h5(mispair=2)
                     ss (len=30)
               h3
               ss(minlen=5, maxlen=21 )
         h3
         h5(mispair=1)
               ss(minlen=5, maxlen=22 )
         h3
```

(b)

Data	50 5s small rRNAs
Population #	100 individuals
Generation #	30 iterations
spC	0.95
stC	0.05
Sensitivity	0.94
Specificity	0.98
Fitness	0.95

Fig. 4. (a): The optimal RCSG of 5S small rRNA for structural learning; (b): The setting of the parameters and the best values for training.

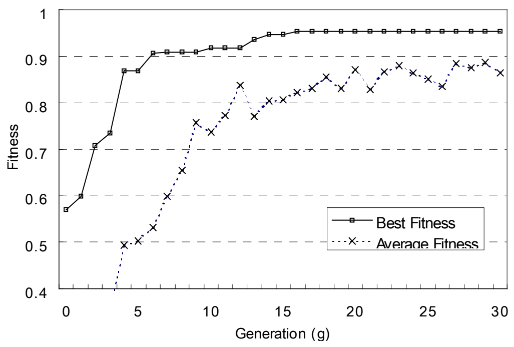


Fig. 5. The plot of the best fitness and the average in 5S small rRNA experiments.

Specificity = 0.98; Fitness = 0.95). However, the values of the variables of the best RCSG have broader range than we expected. It means that the RCSG may be not optimal actually. Thus, to probe the local optima, we have applied the local search, by decreasing the values of the variables gradually.

To analyze the change of the best fitness according to the iterations, we measure the best fitness for each generation (Figure 5). Based on the results, the best fitness reaches a saturation point (0.95) at about thirteenth generation. It is because the searching problem of RCSG for 5S small rRNA is not complex. In most experiments, the esRCSG is used to detect best RCSG before the fifteenth generation. Also, because we apply the elitism during the selection, we can guarantee the best fitness in the last generation. The average fitness also is increased during iteration of esRCSG gradually.

On the other hand, we have searched for the RCSG under several different parameter conditions by varying *spC* and *stC*. We found that the trade-off between specificity and sensitivity was considerably important in searching for the optimum solution, but the population size was not directly related with the problem. In the results, we detected the best solution at *spC* (= 0.95) and *stC* (= 0.05). To assess the optimized RCSG, we evaluated it with the test set of 5S small rRNAs. The sensitivity was 0.83 and the specificity was 0.86 in the results.

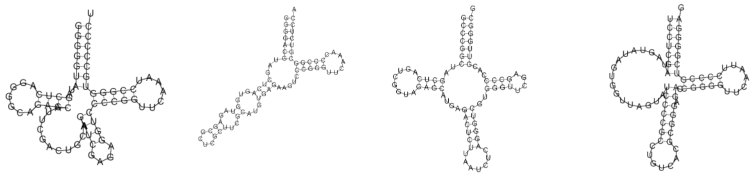


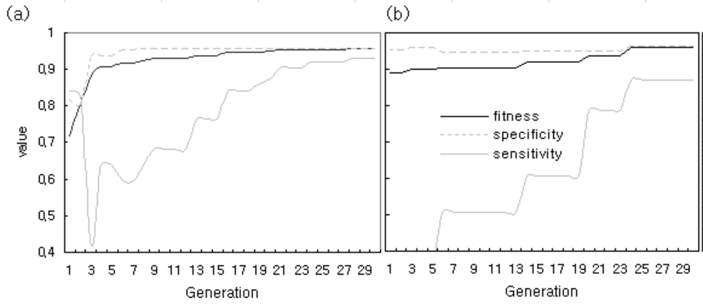
Fig. 6. The secondary structures of *drosophila melanogaster* tRNAs, these structures were drawn by RNAfold of Vienna package. We applied free energy minimum approach with rescale energy parameter to temperature 37°C.

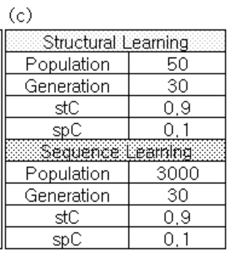
3.2 RCSG of tRNAs by Two-Step Learning

First, we looked into the secondary structure of four tRNAs within the positive training set, which are tRNAs of *drosophila melanogaster*. Figure 6 shows the secondary structure, which are predicted by RNAfold, of the tRNAs. As the results show, the common structure of tRNAs in training data should have two stems instead of three stems. The RCSG of tRNAs, which is learned by two-step genetic programming, is represented as follow. The result shows that the RCSG has conserved two stems, bulge structure and conserved motif as common structure. The conserved motif includes 7 nucleotides, “gaucacu” with allowing 3 mismatches at the loop of second stem loop structure.

```
<Best RCSG for tRNA>
 descr
     h5 ( mispair=2 )
           h5 ( mispair=3 )
                 ss ( len=10 )
           h3
           h5 ( mispair=1 )
                 ss ( len=24, seq="gaucacu", mismatch=3 )
           h3
           ss (minlen = 2, maxlen = 15)
     h3
```

```
<Training>                                  <Test>
Step1_#Structural Learning results          Specificity = 0.946
      Fitness = 0.957                       Sensitivity = 0.840
      Specificity = 0.957
      Sensitivity = 0.931
Step2_#Sequence Learning results
      Fitness = 0.959
      Specificity = 0.964
      Sensitivity = 0.870
```




(c)

Structural Learning	
Population	50
Generation	30
stC	0.9
spC	0.1
Sequence Learning	
Population	3000
Generation	30
stC	0.9
spC	0.1

Fig. 7. The results of two-step genetic programming. (a): The plot of the obtained values during structural learning step, (b): The plot of the obtained values during sequence learning step, (c): The optimized parameters for two steps genetic programming.

Through two-step genetic programming, the sensitivity of the optimized RCSG is diminished but the fitness and specificity of the optimized RCSG increased. The results show sequence learning, the second step, is efficient to gain the specificity. High specificity is important in our study because higher specificity produces less false specificity, sensitivity with the test data as mentioned in the table 1. The results show positive and experimental errors. Next, to validate the optimized RCSG, we measured that the specificity and the sensitivity which are presented in the training rarely change in the test. It means that our algorithm is very efficient for identification of putative ncRNA. Figure 7 shows the change of the best fitness, specificity and sensitivity according to generation in both structural learning step (Figure 7a) and sequence learning step (Figure 7b). Also, we were able to conclude that the specificity converged more quickly than the sensitivity due to high *spC* (=0.9) parameter.

4 Conclusions

In this study, we suggested an efficient approach for searching of a RCSG with ncRNA sequences using genetic programming, which is of evolutionary computation. With fly tRNA sequences and eukaryotic 5S small rRNAs, our learning algorithm, application of genetic programming, learned distinctive RCSGs. The optimized

RCSGs well reflected common structures of training set and could be applied as common structure model for searching of putative ncRNAs.

We can derive two contributions from our new approach. The first contribution is that we have proven the possibility of learning common-structural grammar from structurally unknown sequences through genetic programming. We believe that our approach can be adapted for various applications such as RNA similarity search and putative RNA identification. The second contribution is that we have shown that it is possible to design and generate the RNA structural grammar automatically. Designing of the RNA structural grammar according to the RNA sequences is a difficult problem. Therefore, the system generating structural grammar may be considerably useful.

Acknowledgments. This research was supported by the National Research Laboratory (NRL) Program and the Systems Biology (SysBio) Program of the Ministry of Science and Technology and the BK21-IT Program of the Ministry of Education.

References

1. Rivas E. and Eddy S.R. A dynamic programming algorithm for RNA structure prediction including pseudoknots. *J. Mol. Biol.*, (1999) 285:2053-2068.
2. Zuker M. On finding all suboptimal foldings of an RNA molecule. *Science,* (1989) 244:48-52.
3. Eddy S.R. and Durbin R. RNA sequence analysis using covariance models. *Nucleic Acids Research*, (1994) 22:2079-2088.
4. Gorodkin J., Stricklin S.L. and Stormo G.D. Discovering common stem-loop motifs in unaligned RNA sequences. *Nucleic Acids Reearch*, (2001) 29:2135-2144.
5. Perriquet O., Touzet H. and Dauchet M. Finding the common structure shared by two homolous RNAs. *Bioinformatics*, (2003) 19:108-116.
6. Gary B. Fogel, V. William Porto, Dana G. Weekers, David B. Fogel, Richard H. Griffey, John A. McNeil, Elena Lesnik, David J. Ecker and Rangarajan Sampath. Discovery of RNA structural elements using evolutionary computation. *Nucleic Acids Research*, (2002) 30:5310-5317.
7. Jih-H. Chen, Shu-Yun Le and Jacob V. Maizel. Prediction of common secondary structures of RNAs : a genetic algorithm approach. *Nucleic Acids Research*, (2000) 28:991-999.
8. Sakakibara Y. Pair Hidden Markov models on tree structures *Bioinformatics*, (2003) 19:i232-240.
9. Cai L., Malmberg R.L., Wu Y. Stochastic modeling or RNA pseudoknotted structures: a grammatical approach. *Bioinformatics,* (2003) 19:i66-i73.
10. Sakakibara Y., Brwon M., Hughey, R., Mian I.S., Sjolander K., Underwood R.C. and Haussler D. Stochastic context-free grammars for tRNA modeling. *Nucleic Acids Research*, (1994) 22:5112-5120.
11. Knudsen B. and Hein J. RNA secondary structure prediction using stochastic context-free grammars and evolutionary history. *Bioinformatics*, (1999) 15:446-454.

12. Thomas J. Macke, David J. Ecker, Robin R. Gutell, Daniel Gautheret, David A. Case and Rangarajan Sampath. RNAMotif, an RNA secondary structure definition and search algorithms. *Nucleic Acids Research*, (2001) 29:4724-4735.
13. Koza J.R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, (1992).
14. Zhang B.-T., Ohm P., and Mühlenbein H. Evolutionary neural trees for modeling and predicting complex systems. *Engineering Applications of Artificial Intelligence*, (1997) 10:473-483.