Learning Heuristics for Mining RNA Sequence-Structure Motifs

Achiya Elyasaf, Pavel Vaks, Nimrod Milo, Moshe Sipper, and Michal Ziv-Ukelson

Abstract The computational identification of conserved motifs in RNA molecules is a major—yet largely unsolved—problem. Structural conservation serves as strong evidence for important RNA functionality. Thus, comparative structure analysis is the gold standard for the discovery and interpretation of functional RNAs.

In this paper we focus on one of the functional RNA motif types, sequence-structure motifs in RNA molecules, which marks the molecule as targets to be recognized by other molecules.

We present a new approach for the detection of RNA structure (including pseudoknots), which is conserved among a set of unaligned RNA sequences. Our method extends previous approaches for this problem, which were based on first identifying conserved stems and then assembling them into complex structural motifs. The novelty of our approach is in simultaneously preforming both the identification and the assembly of these stems. We believe this novel unified approach offers a more informative model for deciphering the evolution of functional RNAs, where the sets of stems comprising a conserved motif co-evolve as a correlated functional unit.

Since the task of mining RNA sequence-structure motifs can be addressed by solving the maximum weighted clique problem in an *n*-partite graph, we translate the maximum weighted clique problem into a state graph. Then, we gather and define domain knowledge and low-level heuristics for this domain. Finally, we learn hyper-heuristics for this domain, which can be used with heuristic search algorithms (e.g., A*, IDA*) for the mining task.

The hyper-heuristics are evolved using HH-Evolver, a tool for domain-specific, hyper-heuristic evolution. Our approach is designed to overcome the computational limitations of current algorithms, and to remove the necessity of previous assumptions that were used for sparsifying the graph.

This is still work in progress and as yet we have no results to report. However, given the interest in the methodology and its previous success in other domains we are hopeful that these shall be forthcoming soon.

A. Elyasaf (✉) • P. Vaks • N. Milo • M. Sipper • M. Ziv-Ukelson
Department of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel
e-mail: achiya.e@gmail.com; pavel.vaks@gmail.com; milo.nimrod@gmail.com; sipper@cs.bgu.ac.il; michaluz@cs.bgu.ac.il

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XIII*,
Genetic and Evolutionary Computation, DOI 10.1007/978-3-319-34223-8_2

Keywords Genetic algorithms • Genetic programming • Hyper heuristic

1 Introduction

1.1 RNA Structural Motif Discovery

RNA is a biological macromolecule which, like DNA, is constructed of four letter alphabet (A, C, G and U). RNA has many roles in biological mechanisms, some of which we describe below.

Over the last few years non-coding RNAs (ncRNAs) have been recognized as a highly abundant class of RNAs that do not code for proteins but nevertheless are functional in many biological processes, including localization, replication, translation, degradation, regulation, and stabilization of biological macromolecules (Mandal and Breaker 2004).

1.2 Biological Preliminaries and Definitions

An RNA molecule is defined by a *sequence* of letters (called bases) and a set of pairings between its bases. The base *C* typically pairs with *G*, *A* typically pairs with *U*, and another weaker pairing can occur between *G* and *U*. This base-paired structure is called the *secondary structure* of the RNA. Paired bases almost always occur in a nested fashion. Informally, this means that if we draw arcs over an RNA sequence connecting base pairs, none of the arcs cross each other. When non-nested base pairs occur, they are called *pseudoknots* (see Fig. 1). Most of current RNA sequence-structure analysis algorithms ignore pseudoknots. This is done mostly in order to simplify the problem, due to the fact that prediction of structure while allowing pseudoknots is NP hard (Akutsu 2000). In nature, there are important examples of RNA sequence-structure motifs that include pseudoknots (Staple and Butcher 2005; Brierley et al. 2008).

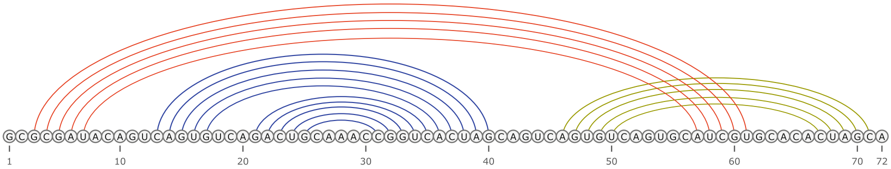
GCGCGAUACAGUCAGUGUCAGACUGCAAACCGGUCACUAGCAGUCAGUGUCAGUGCAUCGUGCACACUAGCA
1
10
20
30
40
50
60
70
72

Fig. 1 An RNA sequence and its structure (defined by the *arcs*). In the figure there are three stems (*marked red, green, and blue*). The *green and red stems* cross each other, indicating that the structure of the exemplified RNA contains a pseudoknot (Color figure online)

In RNA studies, structural conservation serves as a strong evidence for essential RNA functionality. Thus, comparative structure analysis is the gold standard for the discovery and interpretation of functional RNAs. In particular, an important and well-studied problem is that of RNA motif discovery. In molecular biology, a motif is a sequence pattern that is widespread and has, or is conjectured to have, a biological significance. In the case of functional RNAs, the sought motifs are patterns combining sequence and structural features, thus denoted RNA sequence-structure motifs.

In RNA motif discovery, a natural building block is a stem (a local non-crossing set of base pairs). Therefore, some approaches for the mining of RNA motifs are based on first identifying sets of stems that are widespread, and then combining the stems to form more complex structural motif patterns. In this paper we present a novel approach that addresses both problems *simultaneously* using learning of heuristic functions and search techniques. Given a set of n RNA molecules we seek a set of similar stems, each shared by a minimum of $k \leq n$ molecules. Using A^* algorithm along with a learned heuristic function, we mine the n molecules both in sequence and complex structural characteristics. The measure we use for scoring the similarity between two stems models the typical changes caused by evolution.

1.3 Heuristic Search

Heuristic search algorithms are strongly based on the notion of approximating the cost of the cheapest path from a given configuration (or *state*) to the problem's solution (or *goal*). Such approximations are found by means of a computationally efficient function, known as a *heuristic function*. By applying such a function to states reachable from the current one considered, it becomes possible to select more-promising alternatives earlier in the search process, possibly reducing the amount of search effort (typically measured in number of states expanded) required to solve a given problem. The putative reduction is strongly tied to the quality of the heuristic function used: employing a perfect function means simply "strolling" onto the solution (i.e., no search de facto), while using a bad function could render the search less efficient than totally uninformed search, such as breadth-first search (BFS) or depth-first search (DFS).

A heuristic function is said to be *admissible* if it never overestimates the cost to the goal. Thus, the higher the heuristic value, the closer it is to the true cost of the cheapest path to goal. Using an admissible heuristic with an optimal search algorithm (e.g. A* or iterative deepening A*, IDA*, Hart et al. 1968; Korf 1985) guarantees that any solution found will be optimal, i.e., with minimal solution length.

Combining several heuristics to get a more accurate one is considered one of the most difficult problems in contemporary heuristics research (Samadi et al. 2008; Burke et al. 2010). Of course, if all of the heuristic functions are admissible and an optimal solution is what we are looking for, then we could use the max heuristic (which takes the heuristic function with the maximal value). However, when we do not need to guarantee optimality or when we do not use only admissible heuristics, a good solution can be found in a more efficient way.

1.4 Hyper Heuristics

Within combinatorial optimization, the term hyper-heuristics was first used in 2000 (Cowling et al. 2000) to describe heuristics to choose heuristics. This definition of hyper-heuristics was expanded later (Burke et al. 2010) to refer to an automated methodology for selecting or generating heuristics to solve hard computational search problems. In the process of hyper-heuristics learning, heuristics and domain knowledge are used as building blocks. These heuristics can be of high level, usually complex and memory-consuming (e.g., landmarks and pattern databases), or low-level domain knowledge and heuristics that are usually intuitive and straightforward to implement and compute.

Hyper-heuristics have been applied in many research fields, among them:

- Classical planning (Yoon et al. 2008; Fawcett et al. 2011; Hoffmann and Nebel 2001).
- Classical NP-Complete domains, e.g., personnel scheduling (Burke et al. 2003), traveling salesman (Oltean 2005), and vehicle routing (Garrido and Rojas 2010).
- Classical AI domains and puzzles, e.g., the Rush Hour puzzle (Hauptman et al. 2009), the game of FreeCell (Elyasaf et al. 2012), and the Tile Puzzle (Arfaee et al. 2010; Samadi et al. 2008).

The growing research interest in techniques for automating the design of heuristic search methods motivates the search for automatic systems for generating hyper-heuristics.

1.5 Our Approach: Learning Hyper Heuristics for the Task of Mining RNA Sequence-Structure Motifs

In this paper, we present a novel way for mining RNA sequence-structure motifs. We translate the problem into a search graph and devise 97 heuristics for this domain. Next, we use these heuristics as building blocks for learning hyper heuristics using HH-Evolver—an evolutionary algorithm system designed for this type of learning.

This is still work in progress and as yet we have no results to report. However, given the interest in the methodology and its previous success in other domains we are hopeful that these shall be forthcoming soon.

The contributions of this work are as follows:

1. We present a novel approach the task of mining RNA sequence-structure motifs with possible pseudoknots.
2. While previous approaches artificially divide the mining task into conserved stem identification followed by the assembly of such stems into complex structural motifs, we present the *first* approach for preforming the two tasks simultaneously. We believe this unified approach offers a more informative model for deciphering the evolution of functional RNAs, where the sets of stems comprising a conserved motif co-evolve as a correlated functional unit.
3. By using a more efficient search we are able to use a denser graph with more nodes (stems) and edges (relations between stems). Thus, we are able to remove the necessity of previous algorithms to sparsify the graph, at the expense of sensitivity, by imposing assumptions and rigid limitations.
4. The use of hyper heuristics enables the mining algorithm to find the exact conditions (i.e., biological indicators) regarding *when* to apply each heuristic, or combinations.
5. We push the limit of what has been done with evolution further mining RNA sequence-structure motifs one of the most difficult domains to which evolutionary algorithms have been applied to date.
6. Along the way we devise several novel heuristics for this domain. The methodology of creating these heuristics could be applied to other biological domains and other.
7. By devising novel heuristics and evolving them into hyper-heuristics, we continue our previous presentation of a new framework for solving many (non-admissible) heuristic problems, which proved to be efficient and successful.
8. We transform a non-search domain into a state graph, and thus strengthen the bridge between the learning and search community to the biological world.

The rest of the paper is organized as follows: In the next section we examine previous and related work. In Sect. 3 we describe our method. Finally, we end with concluding remarks and future work in Sect. 4.

2 Previous Work

2.1 Mining Common Structure Among a Set of Unaligned RNA Sequences

Several approaches exist for identifying common structure among a given set of RNA molecules. The first approach (Pederson et al. 2006; Hofacker et al. 2002; Washietl and Hofacker 2004) is to align sequences using standard multiple sequence alignment tools (e.g., ClustalW, Thompson et al. 1994), then use structure-conserving mutations for the inference of a consensus structure. However, in order for this approach to work the multiple sequence-alignment step needs to rely on a very well-conserved sequence, which is rarely the case with swiftly evolving ncRNAs.

The second approach applies Sankoff's "Simultaneous Alignment with Folding (SAF)"—an algorithm designed to simultaneously align a set of co-functional RNA molecules and predict their common secondary structure. However, the algorithm requires extensive computational resources both in time ($O(n^6)$) and memory ($O(n^4)$) (Backofen et al. 2011). Thus, current implementations (Mathews and Turner 2002; Havgaard et al. 2005; Will et al. 2007; Siebert and Backofen 2005; Hofacker et al. 2004; Torarinsson et al. 2007) are either restricted in the input size or apply a restricted application of the SAF approach.

The third approach, denoted as the "pre-folding" approach, is applied when no helpful level of sequence conservation is observed. It excludes the sequence alignment step, predicts secondary structures for each sequence (or a sub-group of sequences) separately, and directly aligns the structures. Because of the nested branching nature of RNA structures, these are represented as trees. Tree comparison and tree alignment models in the context of detecting conserved RNA structure have been proposed and implemented in Hofacker et al. (1994), Sczyrba et al. (2003), Hochsmann et al. (2003), Wang and Zhang (2001), and Milo et al. (2013). Predicting a global secondary structure from a single molecule is still error, where the best approaches may yield up to 75 % accuracy (Do et al. 2006).

Due to computational limitations the above methods are generally restricted to finding conserved sequence-structure motifs without considering pseudoknots. A leading approach that removes this restriction is the one proposed by Ji et al. (2004). Here, rather then trying to predict a *global* structure, i.e., one that is common to the (full-length) input sequences, *local* common structure is sought, i.e., a set of substrings that share a similar predicted structure, where each substring belongs to a different input sequence.

Ji et al. apply a multi-stage approach to identify common structure among a given set of n sequences. During a preprocessing stage, an n-partite undirected weighted stem-graph is constructed, by first extracting local stems from each of the sequences. The extracted stems serve as the nodes of the graph, and are partitioned by their sequence of origin. Weighted edges are constructed between pairs of nodes representing the similarity between stems that were extracted from different sequences. Stem similarity is computed by using a single pre-defined scoring function. Ji et al. sparsify this stem-graph by using fixed thresholds on the scoring function and the stability of the stems. In addition, they require the existence of conserved sequential regions of a fixed size, termed *anchors*.

In the next stage, conserved stems are identified. This is done by mining all cliques in the stem-graph spanning a minimum of $k \leq n$ sequences.

This is followed by a final stage, where stem-cliques are combined to form complex motifs. This is achieved by assembling as many compatible cliques as possible according to topological order and evaluating the plausibility of a proposed structure by the significance of its members.

In this paper we extend the work of Ji et al. By applying evolution-based learning, we remove the necessity of the aforementioned single scoring function and the stringent cutoff thresholds. Our search is more informed, and computes intra-clique optimizations within the wide context of topological inter-clique assembly considerations. This allows us to speed up the search without the loss of sensitivity. In fact, sensitivity can now be increased, as the more efficient search allows us to get rid of the preprocessing sparsification used by Ji et al. We use evolution to combine multiple scoring functions and dynamic thresholds.

2.2 Learning Hyper Heuristics

2.2.1 Learning Hyper Heuristics for Planning Systems

Planning systems are strongly based on the notion of heuristics (e.g., Bonet and Geffner 2005; Hoffmann and Nebel 2001). However, relatively little work has been done on *evolving* heuristics for planning.

Aler et al. (2002) (see also Aler et al. 1998, 2001) proposed a multi-strategy approach for learning heuristics, embodied as ordered sets of control rules (called *policies*), for search problems in AI planning. Policies were evolved using a GP (Genetic Programming) based system called EvoCK (Aler et al. 2001), whose initial population was generated by a specialized learning algorithm, called Hamlet (Borrajo and Veloso 1997). Their hybrid system, Hamlet-EvoCK, outperformed each of its sub-systems on two benchmark problems often used in planning: Blocks World and Logistics (solving 85 and 87 % of the problems in these domains respectively). Note that both these domains are considered relatively easy (e.g., compared to Rush Hour and FreeCell, mentioned above), as evidenced by the fact that the last time they were included in an IPC (International Planning Competition) was in 2002.

Levine and Humphreys (2003), and later Levine et al. (2009), also evolved policies and used them as heuristic measures to guide search for the Blocks World and Logistic domains. Their system, L2Plan, included rule-level genetic programming (for dealing with entire rules), as well as simple local search to augment GP crossover and mutation. They demonstrated some measure of success in these two domains, although hand-coded policies sometimes outperformed the evolved ones.

2.2.2 Learning Hyper Heuristics for Specific Domains

Samadi et al. (2008) used artificial neural networks (ANNs) (Mitchell 1999) for learning combinations of heuristics for the sliding-tile puzzle and the 4-peg Towers of Hanoi. They used pattern databases (PDBs) (Korf 1997) and weighted PDBs as input signals for the ANN.

Arfaee et al. (2010) also used ANNs for learning hyper heuristics for several domains, however, in addition to the use of small PDBs as input signals, Arfaee et al. used low-level heuristics and domain knowledge as well. For the sliding-tile puzzle, for example, they used the following additional signals: the number of out-of-place tiles, the position of the blank, the number of tiles not in the correct row, and the number of tiles not in the correct column.

Hauptman et al. (2009) and later Elyasaf et al. (2012) evolved heuristics for the Rush Hour puzzle and the game of FreeCell (respectively). They compared the performance of different types of hyper heuristics on these domains. In both cases, evolved hyper heuristics (along with heuristic search) greatly reduced the number of nodes required to solve instances of the domains, compared to standard methods and previous solvers. In this paper we use their method for learning hyper heuristics for the problem of mining RNA sequence-structure motifs.

3 Method

As explained in Sect. 2.1, we extend the work of Ji et al. (2004). The overall scheme of our method is summarized by the following major steps:

- Cast the problem of mining RNA sequence-structure motifs as one of maximum weighted clique in an *n*-partite graph.
- Translate the maximum weighted clique problem into a search graph, where each state consists of the set of cliques identified so far.
- Gather and define domain knowledge and low-level heuristics for this domain.
- Learn hyper-heuristics for this domain, which can be used with heuristic search algorithms (e.g., A*, IDA*) for the mining task.

In their paper, Ji et al. (2004) list several heuristics and techniques that already exist for the task of mining RNA sequence-structure motifs and for the task of finding maximum weighted cliques, however they note that they are *"not aware of any effective optimization or heuristic algorithm feasible for our problem"*. For this reason they used a fixed scoring function and harsh pruning thresholds (see Sect. 2.1).

Since there are many heuristics, both for the task of mining RNA sequence-structure motifs and for the task of finding maximum weighted cliques, we believe that learning algorithms can find a combination of these heuristics that outperforms the handcrafted scoring formula used by Ji et al.

The key differences between our approach and Ji et al. are: (1) We use learning algorithms with *many* heuristics along with domain knowledge to avoid the use of a single formula; (2) we let evolution guide the search and remove the need to rely on fixed thresholds and anchoring; (3) we extend the stem extraction module by using the approach of Milo et al. (2014), which allows us find a wider range of stem types; (4) we perform both mining tasks (identification of conserved stems and the assembly of conserved stems into complex motif structures) simultaneously.

3.1 Casting the Problem of Mining RNA Sequence-Structure Motifs as One of Maximum Weighted Clique in an n-Partite Graph

Given a set of n molecules, we extend the stem extraction module of Ji et al. by using the extraction method described in Milo et al. (2014), allowing us to extract more stems from the molecules. Next, we construct an n-partite undirected weighted graph to represent the stems and their relations. Each vertex represents a stem, and the graph is partitioned into n parts. Each part comprises the stems from one molecule. Only stems (or vertices) from different parts can be connected. We call the edges between the stems *stem edges*, to differentiate between these edges and the edges described in Sect. 3.2. While Ji et al. sparsify this graph using rigid thresholds, we do not sparsify the graph at this stage and connect *all* stems of different partitions.

For each stem edge we define several features (dubbed stem edge features, or SEF), described in Sect. 3.3.1. For each feature it is possible to define a hard threshold set by the user, or let the learning algorithm set the threshold value.

3.2 Converting the Maximum Weighted Clique Problem into a State Graph

Once the n-partite graph is built, Ji et al. (2004) extract all possible stem-cliques, a task for which the worst-case time complexity is $O(m^n)$, where m is the maximum number of stems examined in one sequence and n is the number of total RNA sequences. The average run-time is much less than the worst case, due to the thresholds and pruning. However, the time could still be impractical in many cases, and furthermore the sparsification comes at the expense of data-mining sensitivity. Moreover, if the goal is to mine the input and find significantly similar structures among the molecules, one does not need to exhaustively consider all possible cliques. If we are able to find the most important cliques first, we can stop the search at an earlier phase. This is the reason that we now turn to heuristic search algorithms such as A*.

In order to search for cliques in an A* manner, we first define a new search graph, where each state contains the set of cliques (and partial cliques) found so far. The edges represent either the start of a new clique or adding a stem to an existing clique. As opposed to standard search domains, here we do not know which state is the goal state (i.e., the state with the most important cliques). We will discuss this point further in Sect. 3.4.3.

We now turn to describe the domain knowledge and low-level heuristics gathered for the search algorithm.

3.3 Gather and Define Domain Knowledge and Low-Level Heuristics for this Domain

There are three different types of heuristics we use:

1. *SEF heuristics*. heuristics that are derived from the stem edge features. Recall that each search state represents the cliques founds so far and that each clique contains several stem edges. With this notion, we define seven heuristics for each stem edge feature, f:

   (a) Return the minimal/maximal/average/median f value of all stem edges of the state.
   (b) Compute average f value per clique; return average of all average values.
   (c) Compute median f value per clique; return average of all median values.
   (d) Compute average f value per clique; return median of all average values.

   The full SEF list is described below.
2. *Topological relation heuristics*. heuristics that measure the topology preservation between stems from different cliques using the *topological relations*. We have two heuristics of this type: The first is described in Milo et al. (2014), and the second is a heuristic version of the structure assembly algorithm described in Ji et al. (2004).
3. *Clique-specific heuristics*. standard clique heuristics such as maximal, minimal, average, median node degree.

All of our heuristics preserve the basic rule of heuristic functions: the lower the value, the closer we are to the goal. Towards this end we change the heuristic value h to $1/h$, where applicable.

There is a fourth type of heuristics—classic search heuristics (e.g., pattern database)—however, previous work has shown that using domain knowledge-based heuristics provides good solutions.

The distribution of the heuristic types is described in Table 1.

Table 1 Distribution of heuristic types

Heuristic type	Number of heuristics
SEF	91 (7 heuristics $*$ 13 features $=$ 91)
Topological relation heuristics	2
1 Clique-specific heuristics	4
Total	97

3.3.1 Stem Edge Features

As described in Sect. 2.1, Ji et al. (2004) defined a fixed equation for describing similarity between two stems, which was a combination of five features. We use these features, as well as the equation, as part of our stem edge features (SEF), along with additional features described below.

Some of our features are added twice: once as is (i.e., $f_l(i_x, j_y)$), and once divided by the energy, using the formula $\bar{f}_l(i_x, j_y) = \frac{2 \cdot f_l(i_x, j_y)}{2 + r_x(i) + r_y(j)}$ (as described by Ji et al.).

The complete list of our features is described in Table 2.

Table 2 The list of features

Feature	Origin	Description
f_1 : Helix length	[1]	Number of base-pairs in the stem
f_2:$f_2 = \bar{f}_1$	[1]	[*]
f_3 : Helix sequence	[1]	The sequence of bases in the stem
f_4:$f_4 = \bar{f}_3$	[1]	[*]
f_5 : Loop sequence	[1]	The sequence of letters between the innermost base-pair in the stem
f_6:$f_6 = \bar{f}_5$	[1]	[*]
f_7 : Stem stability	[1]	The free energy value of the stem
f_8:$f_8 = \bar{f}_7$	[1]	[*]
f_9 : Relative positions	[1]	The position of the left base in the outermost base-pair in the stem (relative to the sequence)
f_{10}:$f_{10} = \bar{f}_9$	[1]	[*]
f_{11} : Ji. et al. Similarity	[1]	$\frac{2 \cdot \sum_{l=1,3,5,7} \{w_l \cdot f_l(i_x, j_y)\}}{2 + r_x(i) + r_y(j)}$
f_{12} : Context	[3]	The shift between both helix counterparts of a stem in the anchor or non-anchor region
f_{13} : StemSearch similarity	[2]	Similarity score used by StemSearch, determined by structural and sequential similarity

[1]—Features taken from Ji et al. (2004)
[2]—Features taken from Milo et al. (2014)
[3]—Features designed by us
[*]—$\bar{f}_l(i_x, j_y) = \frac{2 \cdot f_l(i_x, j_y)}{2 + r_x(i) + r_y(j)}$

3.4 Learning Hyper Heuristics Using HH-Evolver

Combining several heuristics to get a more accurate one is considered one of the most difficult problems in contemporary heuristics research (Samadi et al. 2008; Burke et al. 2010).

This task typically involves solving three major sub-problems:

1. How to combine heuristics by *arithmetic* means, e.g., by summing their values or taking the maximal value.
2. Finding exact conditions (i.e., *logic* functions) regarding *when* to apply each heuristic, or combinations thereof—some heuristics may be more suitable than others when dealing with specific state configurations.
3. Finding the proper set of domain configurations in order to facilitate the learning process while avoiding pitfalls such as overfitting.

The problem of combining heuristics is difficult mainly because it entails traversing an extremely large search space of possible numeric combinations, logic conditions, and state configurations. To tackle this problem we turn to *evolution*.

As previously mentioned, we use the learning method of Hauptman et al. (2009) and Elyasaf et al. (2012) for the mining problem. For this task we use their tool, *HH-Evolver* (Elyasaf and Sipper 2013), a hyper-heuristic generator for search domains. The HH-Evolver system receives as input: a domain, several heuristics for the domain, and a dataset of domain instances to be used partly as training set and partly as test set. HH-Evolver generates a population of random hyper-heuristics and trains them over generations against the training set. When used with a heuristic search algorithm, the individuals are required to produce near-optimal solutions to the instances encountered. The search-size (i.e., the number of states encountered during the search) should be small as well.

In order to properly solve the aforementioned sub-problems, we use three different HH-Evolver genomic representations—Standard GA, GP, and policies—all of which are thoroughly described below. These representations use the heuristics given as input to HH-Evolver.

3.4.1 The Hyper Heuristic-Based Genome

We use three different genomic representations. All of these representation were used by Hauptman et al. (2009) and Elyasaf et al. (2012).

All of our representations comprise real-value thresholds for all minimum heuristics (e.g., minimal node degree heuristic). During the search we prune nodes when one of the heuristic values exceeds its threshold.

Standard GA (Genetic Algorithm)

This type of hyper-heuristic only addresses the first problem of how to combine heuristics by arithmetic means. Each individual comprises 97 real values in the range [0, 1], representing a linear combination of all 97 heuristics described above (Table 1). Specifically, the heuristic value, H, designated by an evolving individual is defined as $H = \sum_{i=1}^{97} w_i h_i$, where w_i is the ith weight specified by the genome, and h_i is the ith heuristic shown in Table 1. As results in other domains showed (Hauptman et al. 2009; Elyasaf et al. 2012), the GA proved quite successful and was therefore retained as a yardstick to measure against when we embarked upon our GP path.

GP (Genetic Programming)

As we want to embody both combinations of estimates and conditions for applying each combination, we evolve GP-trees as described in Koza (1994). The function set included the functions $\{IF, AND, OR, \leq, \geq, *, +\}$, and the terminal set included all heuristics in Table 1, as well as random numbers within the range [0, 1].

Policies

The last genome used also combines estimates and application conditions, using ordered sets of control rules, or *policies*. As stated above, policies have been evolved successfully with GP to solve search problems—albeit simpler ones (e.g., Hauptman et al. 2009; Elyasaf et al. 2012).

The policies have the following structure:

$RULE_1$: IF $Condition_1$ THEN $Value_1$
.
.
.
$RULE_N$: IF $Condition_N$ THEN $Value_N$
$DEFAULT$: $Value_{N+1}$

where $Condition_i$ and $Value_i$ represent conditions and estimates, respectively.

Policies are used by the search algorithm in the following manner: The rules are ordered such that we apply the first rule that “fires” (meaning its condition is true for the current state being evaluated), returning its *Value* part. If no rule fires, the value is taken from the last (default) rule: $Value_{N+1}$. Thus individuals, while in the form of policies, are still heuristics—the value returned by the activated rule is an arithmetic combination of heuristic values, and is thus a heuristic value itself. This accords with our requirements: rule ordering and conditions control when we apply a heuristic combination, and values provide the combinations themselves.

Thus, with N being the number of rules used, each individual in the evolving population contains N *Condition* GP trees and $N + 1$ *Value* sets of weights used for computing linear combinations of heuristic values. Experimenting with several sizes of policies in different domains showed that $N = 5$ provides enough rules per individual, while avoiding cumbersome individuals with too many rules. The depth limit used for the *Condition* trees is set to 5 for the same reason.

For *Condition* GP trees, the function set included the functions $\{AND, OR, \leq, \geq\}$, and the terminal set included all heuristics in Table 1. The sets of weights appearing in *Value*s all lie within the range [0, 1], and correspond to the heuristics listed in Table 1.

The genetic operators for the individuals are described thoroughly in Elyasaf et al. (2012).

3.4.2 Training and Test Sets

We use the data from the *RFAM* (Griffiths-Jones et al. 2005) database to create training and testing data sets. RFAM is a curated database of RNA molecules grouped together by common functionality (termed *RNA families*). This allows us to generate a large and diverse set of syntactic inputs for our hyper-heuristic population. This set is divided to training set and test sets, using a standard cross-validation method.

The set is generated as follow: For each family we randomly divide the members into different (overlapping) groups of different sizes. We then add to each group a random amount of noise in the range of $[0, \frac{n}{5}]$, where the noise is defined as members of a different functional family. Each generated group is used as an instance in the set.

The above technique allows us to collect different statistical metrics needed for computing the correctness of solutions (described below).

3.4.3 Fitness

The individual's fitness score preserves the rule that the higher the value, the better the individual (as opposed to heuristic functions). The score is obtained by running A* on m instances taken from the training set, with the individual used as the heuristic function. The g value is defined as the length of the path from the initial state to the current one. During the search, each individual keeps the best state found so far (i.e., the one with the lowest heuristic value). Once the search space is exhausted or the time limit (described below) is reached, the individual returns that state, designating it as the goal state.

The fitness score is then calculated, incorporating three elements:

1. `Answer's correctness (80 %)`: The most important element is the correctness, hence its weight is 80 % of the score. The correctness calculation is described below.
2. `Time to solution (10 %)`: The time to solution is defined as the average time it took the individual to find the solution for all instances. We wish to reduce this time, and thus the element's weight is 10 %. The score of the element is $1/\mathit{time\_to\_solution}$.
3. `Number of cliques (10 %)`: Since the cliques represent the similarities between the molecules, the more correct cliques the individual finds, the better the solution is. The element's weight is 10 %.

Each element is normalized to a value in the range $[0, 1]$. Towards this end we divide the element value by its maximal value, where applicable.

Correctness is calculated by the *Positive Predictive Values* (PPV) formula: $\mathit{correctness} = \frac{TP}{TP+FP}$, where TP is the number of times the individual marked a stem correctly in a clique (for all training instances), and FP is the number of times the individual marked a stem incorrectly in a clique (also for all training instances). In case $TP = FP = 0$, the correctness value is 0. For example, if we have three molecules from the same family and one "noisy" molecule from a different family, we increase TP by one for each stem in a clique that belongs to a molecule of the family. Consequently, we increase FP by one for each stem in a clique that belongs to the noisy molecule.

The number of instances the individuals are trained on, m, is limited by the computational resources. Yet, the user should set it to be as high as possible in order to reach convergence. Avoiding over-fitting is achieved by using a standard cross-validation method.

The minimum clique size, k, is set by the user according to the molecular families and should be in the range $[\frac{n}{2}, n]$.

3.4.4 Search Time

The time limit for a hyper-heuristic to return an answer is set initially by the user. However, during evolution, we collect the median time for finding the best state. In such a way we can reduce the time limit to be the *average time* + *1 min*.

4 Concluding Remarks

We presented a new approach for the detection of RNA structure (including pseudoknots), which is conserved among a set of unaligned RNA sequences. Our method extends previous approaches that were based on first identifying conserved stems and then assembling them into complex structural motifs. The novelty of our approach is in simultaneously preforming both the identification and the assembly of these stems. We believe this novel unified approach offers a more informative model for deciphering the evolution of functional RNAs, where the sets of stems comprising a conserved motif co-evolve as a correlated functional unit.

We began by casting the problem of mining RNA sequence-structure motifs as one of maximum weighted clique in an *n*-partite graph of stems. Next we addressed the maximum weighted clique problem as a heuristic search problem, by translating it into a search graph, where each state consists of the set of cliques identified so far. We then defined domain knowledge and low-level heuristics for this domain. Finally, we learned hyper-heuristics for this domain, which could be used with heuristic search algorithms (e.g., A*, IDA*) for the mining task.

This paper extends a leading approach by Ji et al. (2004) with several important modifications. By applying evolution-based learning we remove the necessity of their single scoring function and the stringent cutoff thresholds. Our search is more informed and computes intra-clique optimizations within the wide context of topological inter-clique assembly considerations. This allows us to speed up the search without the loss of sensitivity. In fact, sensitivity can now be increased, as the more efficient search obviates the preprocessing sparsification used by Ji et al. We used evolution to combine multiple scoring functions and dynamic thresholds.

This is still work in progress and as yet we have no results to report. However, given the interest in the methodology and its previous success in other domains we are hopeful that these shall be forthcoming soon (servers are churning as you read these lines).

Our work may lead to several possible research directions:

- We believe our unified approach offers a more informative model for deciphering the evolution of functional RNAs, where the sets of stems comprising a conserved motif co-evolve as a correlated functional unit. We hope that our results will shed light on functional RNAs and that new biological discoveries will follow.
- The methodology presented here could be applied to other biological and non-biological domains.
- There are many heuristics and similarity measurements that are not described in this can be incorporated into our system.

Acknowledgements This research was supported by the Israel Science Foundation (grant no. 123/11 and grant no. 179/14).

References

Akutsu T (2000) Dp algorithms for rna secondary structure prediction with pseudoknots. Discrete Appl Math 104(1–3):45–62

Aler R, Borrajo D, Isasi P (1998) Genetic programming of control knowledge for planning. In: Proceedings of AIPS-98

Aler R, Borrajo D, Isasi P (2001) Learning to solve planning problems efficiently by means of genetic programming. Evol Comput 9(4):387–420

Aler R, Borrajo D, Isasi P (2002) Using genetic programming to learn and improve knowledge. Artif Intell 141(1–2):29–56

Arfaee SJ, Zilles S, Holte RC (2010) Bootstrap learning of heuristic functions. In: Proceedings of the 3rd international symposium on combinatorial search (SoCS2010), pp 52–59

Backofen R, Tsur D, Zakov S, Ziv-Ukelson M (2011) Sparse folding: time and space efficient algorithms. J Discrete Algorithms 9(1):12–31

Bonet B, Geffner H (2005) mGPT: A probabilistic planner based on heuristic search. J Artif Intell Res 24:933–944

Borrajo D, Veloso MM (1997) Lazy incremental learning of control knowledge for efficiently obtaining quality plans. Artif Intell Rev 11(1–5):371–405

Brierley I, Gilbert RC, Pennell S (2008) Pseudoknots and the regulation of protein synthesis. Biochem Soc Trans 36(4):684–689

Burke EK, Kendall G, Soubeiga E (2003) A tabu-search hyperheuristic for timetabling and rostering. J Heuristics 9(6):451–470. http://dx.doi.org/10.1023/B:HEUR.0000012446.94732.b6

Burke EK, Hyde M, Kendall G, Ochoa G, Ozcan E, Woodward JR (2010) A classification of hyper-heuristic approaches. In: Gendreau M, Potvin J (eds) Handbook of meta-heuristics, 2nd edn. Springer, Berlin, pp 449–468

Cowling PI, Kendall G, Soubeiga E (2000) A hyperheuristic approach to scheduling a sales summit. In: Burke EK, Erben W (eds) PATAT. Lecture notes in computer science, vol 2079. Springer, Berlin, pp 176–190. doi:10.1007/3-540-44629-X_11

Do CB, Woods DA, Batzoglou S (2006) Contrafold: RNA secondary structure prediction without physics-based models. Bioinformatics 22(14):e90–e98

Elyasaf A, Sipper M (2013) Hh-evolver: a system for domain-specific, hyper-heuristic evolution. In: Proceedings of the 15th annual conference companion on genetic and evolutionary computation GECCO '13 companion. ACM, New York, pp 1285–1292. doi:10.1145/2464576.2482707. http://doi.acm.org/10.1145/2464576.2482707

Elyasaf A, Hauptman A, Sipper M (2012) Evolutionary design of FreeCell solvers. IEEE Trans Comput Intell AI Games 4(4):270–281. doi:10.1109/TCIAIG.2012.2210423. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6249736

Fawcett C, Karpas E, Helmert M, Roger G, Hoos H (2011) Fd-autotune: domain-specific configuration using fast-downward. In: Proceedings of ICAPS-PAL 2011

Garrido P, Rojas MCR (2010) DVRP: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic. J Heuristics 16(6):795–834. http://dx.doi.org/10.1007/s10732-010-9126-2

Griffiths-Jones S, Moxon S, Marshall M, Khanna A, Eddy SR, Bateman A (2005) RFAM: annotating non-coding RNAS in complete genomes. Nucleic Acids Res 33(suppl 1):D121–D124

Hart PE, Nilsson NJ, Raphael B (1968) A formal basis for heuristic determination of minimum path cost. IEEE Trans Syst Sci Cybern 4(2):100–107

Hauptman A, Elyasaf A, Sipper M, Karmon A (2009) GP-Rush: using genetic programming to evolve solvers for the Rush Hour puzzle. In: GECCO'09: Proceedings of 11th annual conference on genetic and evolutionary computation conference. ACM, New York, pp 955–962. doi:10.1145/1569901.1570032. http://dl.acm.org/citation.cfm?id=1570032

Havgaard J, Lyngso R, Stormo G, Gorodkin J (2005) Pairwise local structural alignment of RNA sequences with sequence similarity less than 40%. Bioinformatics 21(9):1815–1824

Hochsmann M, Toller T, Giegerich R, Kurtz S (2003) Local similarity in RNA secondary structures. In: Proceedings of the IEEE computer society conference on bioinformatics, Citeseer, p 159

Hofacker I, Fontana W, Stadler P, Bonhoeffer L, Tacker M, Schuster P (1994) Fast folding and comparison of RNA secondary structures. Monatshefte fur Chemie/Chemical Monthly 125(2):167–188

Hofacker I, Fekete M, Stadler P (2002) Secondary structure prediction for aligned RNA sequences. J Mol Biol 319:1059–1066

Hofacker I, Bernhart S, Stadler P (2004) Alignment of RNA base pairing probability matrices. Bioinformatics 20(14):2222–2227

Hoffmann J, Nebel B (2001) The FF planning system: fast plan generation through heuristic search. J Artif Int Res 14(1):253–302. http://dl.acm.org/citation.cfm?id=1622394.1622404

Ji Y, Xu X, Stormo GD (2004) A graph theoretical approach for predicting common rna secondary structure motifs including pseudoknots in unaligned sequences. Bioinformatics 20(10):1591–1602

Korf RE (1985) Depth-first iterative-deepening: an optimal admissible tree search. Artif Intell 27(1):97–109

Korf RE (1997) Finding optimal solutions to Rubik's cube using pattern databases. In: Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on innovative applications of artificial intelligence, AAAI'97/IAAI'97, AAAI Press, pp 700–705

Koza JR (1994) Genetic programming II: automatic discovery of reusable programs. MIT Press, Cambridge, MA

Levine J, Humphreys D (2003) Learning action strategies for planning domains using genetic programming. In: Raidl GR, Meyer JA, Middendorf M, Cagnoni S, Cardalda JJR, Corne D, Gottlieb J, Guillot A, Hart E, Johnson CG, Marchiori E (eds) EvoWorkshops. Lecture notes in computer science, vol 2611. Springer, New York, pp 684–695

Levine J, Westerberg H, Galea M, Humphreys D (2009) Evolutionary-based learning of generalised policies for AI planning domains. In: Rothlauf F (ed) Proceedings of the 11th annual conference on genetic and evolutionary computation (GECCO 2009). ACM, New York, pp 1195–1202

Mandal M, Breaker RR (2004) Gene regulation by riboswitches. Cell 6:451–463

Mathews DH, Turner DH (2002) Dynalign: an algorithm for finding the secondary structure common to two RNA sequences. J Mol Biol 317(2):191–203

Milo N, Zakov S, Katzenelson E, Bachmat E, Dinitz Y, Ziv-Ukelson M (2013) Unrooted unordered homeomorphic subtree alignment of rna trees. Algorithms Mol Biol 8(1):13

Milo N, Yogev S, Ziv-Ukelson M (2014) Stemsearch: Rna search tool based on stem identification and indexing. Methods

Mitchell TM (1999) Machine learning and data mining. Commun ACM 42(11):30–36

Oltean M (2005) Evolving evolutionary algorithms using linear genetic programming. Evol Comput 13(3):387–410. http://dx.doi.org/10.1162/1063656054794815

Pederson J, Bejerano G, Siepel A, Rosenbloom K, Lindblad-Toh K, Lander E, Kent J, Miller W, Haussler D (2006) Identification and classification of conserved RNA secondary structures in the human genome. PLOS Comput Biol 2:e33

Samadi M, Felner A, Schaeffer J (2008) Learning from multiple heuristics. In: Fox D, Gomes CP (eds) Proceedings of the twenty-third AAAI conference on artificial intelligence (AAAI 2008), AAAI Press, pp 357–362

Sczyrba A, Kruger J, Mersch H, Kurtz S, Giegerich R (2003) RNA-related tools on the bielefeld bioinformatics server. Nucleic Acids Res 31(13):3767

Siebert S, Backofen R (2005) MARNA: multiple alignment and consensus structure prediction of RNAs based on sequence structure comparisons. Bioinformatics 21(16):3352–3359

Staple DW, Butcher SE (2005) Pseudoknots: RNA structures with diverse functions. PLoS Biol 3(6):e213

Thompson J, Higgins D, Gibson T (1994) CLUSTALW: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic Acids Res 22(22):4673

Torarinsson E, Havgaard JH, Gorodkin J (2007) Multiple structural alignment and clustering of RNA sequences. Bioinformatics 23(8):926–932

Wang Z, Zhang K (2001) Alignment between two RNA structures. Lecture notes in computer science. Springer, Berlin, pp 690–702

Washietl S, Hofacker I (2004) Consensus folding of aligned sequences as a new measure for the detection of functional RNAs by comparative genomics. J Mol Biol 342:19–30

Will S, Reiche K, Hofacker IL, Stadler PF, Backofen R (2007) Inferring non-coding RNA families and classes by means of genome-scale structure-based clustering. PLOS Comput Biol 3(4):e65

Yoon SW, Fern A, Givan R (2008) Learning control knowledge for forward search planning. J Mach Learn Res 9:683–718. http://doi.acm.org/10.1145/1390681.1390705

Springer

http://www.springer.com/978-3-319-34221-4

Genetic Programming Theory and Practice XIII
Riolo, R.; Worzel, B.; Kotanchek, M.; Kordon, A. (Eds.)
2016, XX, 262 p. 69 illus., 31 illus. in color., Hardcover
ISBN: 978-3-319-34221-4