Locomotif: from graphical motif description to RNA motif search
Janina Reeder, Jens Reeder and Robert Giegerich*
Faculty of Technology, Bielefeld University, D-33615 Bielefeld, Germany

ABSTRACT
Motivation and Results: Motivated by the recent rise of interest in small regulatory RNAs, we present Locomotif—a new approach for locating RNA motifs that goes beyond the previous ones in three ways: (1) motif search is based on efficient dynamic programming algorithms, incorporating the established thermodynamic model of RNA secondary structure formation. (2) motifs are described graphically, using a Java-based editor, and search algorithms are derived from the graphics in a fully automatic way. The editor allows us to draw secondary structures, annotated with size and sequence information. They closely resemble the established, but informal way in which RNA motifs are communicated in the literature. Thus, the learning effort for Locomotif users is minimal. (3) Locomotif employs a client-server approach. Motifs are designed by the user locally. Search programs are generated and compiled on a bioinformatics server. They are made available both for execution on the server, and for download as C source code plus an appropriate makefile.
Availability: Locomotif is available at http://bibiserv.techfak.uni-bielefeld.de/locomotif
Contact: robert@techfak.uni-bielefeld.de

1 INTRODUCTION
With the recently increased interest in the regulatory role of RNA, various experimental methods have been devised for isolating candidates of regulatory RNAs (Hüttenhofer and Vogel, 2006; Vogel and Sharma, 2005). Genome tiling arrays, also covering ‘intergenic’ regions (i.e. between protein coding genes) have been devised and allow the monitoring of RNA gene expression under varying conditions (Rübberg et al., 2003). Gene prediction from a single genomic sequence, akin to gene finders like GLIMMER or GenScan (Burge and Karlin, 1998; McHardy et al., 2004) is not possible to date for RNA genes. RNAz (Washietl et al., 2005), currently the most popular approach to RNA gene prediction, requires the use of multiple genome alignments and hence, a moderate to high degree of conservation. It is based on the insight that with RNA genes, function is encoded within structure, and seeks to find loci of significant conservation of secondary structure. A run of RNAz on bacterial genomes under moderately strict parameters generates several hundred to a few thousand RNA gene candidates.

*To whom correspondence should be addressed.

Given that a suspected RNA gene has been obtained by whichever method, the problem of its functional classification arises. The Rfam database (Griffiths-Jones et al., 2005) is a widely used resource, containing covariance models of over 500 RNA families. Covariance models can be built when a moderate number of related sequences with a common structure and a moderate amount of sequence conservation are available. If a new RNA gene candidate scores well against an Rfam family model, this serves as a first indicator about its functional role.

Here, we deal with the case where a run against Rfam is negative—either because a suitable family model does not yet exist, or because the candidate at hand shows too little sequence similarity to be recognized by one of the models. We also address the case where we are not dealing with a complete RNA gene candidate, but rather with a suspected regulatory motif (say) inside a 5′ untranslated region.

In these situations, a prediction of its secondary structure can be readily obtained (Hofacker et al., 1994; Zuker, 2003), if desired also a small but representative set of structural alternatives (Giegerich et al., 2004b). This again may guide experimental studies to validate functionally relevant parts in these structures by introducing sequence mutations. Of course, it is laborious and costly (and for some organisms infeasible) to check for the manifestation of a phenotype. Therefore, one wants to employ computational means to accumulate further evidence for or against a functional role of the candidate. One wants to check for occurrences of a similar structure in data sets from other organisms—not only among candidates from other experimental harvests or computational predictions, but also directly by scanning a genome sequence. This leads to the problem of matching a structural motif to sequence data, which is addressed here.

Several approaches to RNA motif matching have been developed (Billoud et al., 1996; Dsoouza et al., 1997; Laferrière et al., 1994; Macke et al., 2001; Pesole et al., 2000), the most advanced (and available) tool probably being RNaMotif. It provides a textual motif description language, which is fairly easy to learn, and allows to specify helices and sequence motifs in their 5′ to 3′ succession. Still, this is quite distinct from the graphical form in which biologists usually describe structural RNA motifs in their publications. RNaMotif automatically generates a matcher program from the description.

The common drawback of these tools is that they are combinatorial matchers. The fact that a sequence can exhibit a particular pattern of base pairing does not imply that this pattern is favored by the thermodynamics of RNA folding.

© 2007 The Author(s)
This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/2.0/uk/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.
Therefore, a large number of (often overlapping and redundant) matches is generated, most of which are refuted ex post by checking folding energy. An alternative is the development of thermodynamic matchers (TDMs), first advocated in (Reeder and Giegerich, 2004). TDMs are specialized folding algorithms, using the established thermodynamic model to evaluate the capability of a sequence to fold into a predefined motif. TDMs do not use a motif description language, but the motif is hard coded in the program. This guarantees good efficiency, asymptotically the same as general folding programs. A further encouraging result in this direction is the recent demonstration (Höchsmann et al., 2006) that low-energy foldings into specialized structure have higher significance than low-energy folds into arbitrary structures [see Clote et al. (2005) and the discussion summarized therein].

The main obstacle to such an approach is the necessity to develop a new folding algorithm for each special structure or motif, which is a laborious task even for a well-trained bioinformatician. This problem is alleviated by the Locomotif system (Reeder, 2007) presented here.

The name Locomotif is (loosely) derived from its task—locating structural motifs in RNA sequences. Locomotif goes beyond the aforementioned approaches in three ways: (1) Locomotif employs a graphical motif description language. This graphical language is very similar to the established, but informal way in which RNA motifs are communicated in the literature. (2) Matching algorithms are derived from the graphics in a fully automatic way, employing dynamic programming techniques incorporating the established thermodynamic model of RNA secondary structure formation. (3) Since translating structural graphics into efficient matchers requires non-trivial processing, Locomotif employs a client-server approach. Motifs are designed by the user locally, search programs are generated and compiled on a bioinformatics server, and are available both for execution on the server, or for download.

In the subsequent sections, we describe the process of interactive motif design (Section 2) with Locomotif, the details of the graphical motif description language (Section 3), a case study in motif design (Section 4), the transformation of structure drawings into efficient algorithms (Section 5) and the overall client-server architecture of the Locomotif system (Section 6). We conclude with a discussion of the possible extensions to the Locomotif approach.

## 2 INTERACTIVE MOTIF CONSTRUCTION

The front end of the Locomotif system is a visual editor that allows us to define motifs without requiring any detailed study of a textual descriptor language. Instead, visual building blocks are combined manually on an interactive canvas. This section concentrates on the overall process of motif construction, while a more detailed description of the building blocks and their refinements is deferred to the next section.

2.1 Basic motif building blocks

The basic graphical building blocks for motif definition correspond to common elements of RNA secondary structure. An RNA motif can be composed of single strands, stems (also called helices), bulges, internal loops, hairpin loops and multiloops, and for each of these elements, a graphical building block is available. The editor controls the use of these building blocks, in order to ensure correctness of the motif structure at all times. For example, we restrict the role of single strands to connecting or prolonging motif parts. You cannot compose a multiloop from stems and single strands, you must use the multiloop building block. Attachment of two hairpin loops onto each other, thereby creating a circular structure, is also prevented. In such cases, the building blocks will simply refuse to connect.

Such patronizing behavior of the editor has a 3-fold reason: it standardizes to a certain extent the way motifs are constructed, ensures that they are biologically plausible and that they can eventually be translated into correct TDMS.

2.2 Moving blocks around

Upon a click on one of the buttons shown in Figure 1, the chosen building block forms a mobile RNA motif puzzle piece that can be moved around within the editing surface. In a drag-and-drop mode, we can freely choose where to place the building block and release it from the mouse cursor with another click. Then, any following building block can either be placed on a distinct location of the drawing canvas, forming a new motif part. Alternatively, it can be attached to any open side of the previous building block(s), to which it will be attracted in a magnetic fashion. We only need to move our new puzzle pieces into proximity of their intended location, and they will snap in their place within the overall motif. When attached,
the sequence strands are elongated, and if necessary, the 5' and 3' end of the overall motif are transported to a new position.

2.3 Screen layout

During the motif construction, rotations can be performed manually in order to best adapt the motif definition to an already existing view of the motif. Here, either the entire motif structure can be rotated or just the building block currently attached to the mouse cursor. Also, if a building block is intended to be attached to an existing structure and needs to be rotated for a proper fit, this will be calculated automatically: the visual building block snaps in place.

Similarly, the entire motif structure can be moved around to allow adjustments if the original placing within the design interface proves wrong. That way, we can add building blocks to all sides without running into problems regarding the size of the view on the motif structure. Also, for very large motifs, we can zoom in or out to obtain a more general or finer view on the motif structure.

2.4 Motif annotation and editing

In addition to defining the basic motif structure, detailed information on the motif can be specified via editing interfaces. For each building block, we can open a textual annotation interface by clicking on it, and depending on its type, several restrictions can be imposed (see Fig. 2 for an example and Section 3 for details). Whenever we are not satisfied with the motif structure as designed, we can edit it by deleting or detaching building blocks from the structure. In the first case, the building block is completely removed from the screen, in the second case it is reattached to the mouse cursor with all the properties already stored for the building block. We can only remove building blocks from the ends of the structure to avoid gaps in the motif. Note that in this case, both a building block with an open side as well as a hairpin loop connected to the structure are removable ends. Furthermore, before removing any other building blocks, we must delete all single strands from the motif. While these restrictions can be somewhat bothersome, they are necessary to avoid displacement of 5' and 3' end on separate motif parts or the addition of single strands to several sides of a motif part.

2.5 Motif construction strategies

The strategy how to construct a complex motif is left to the user. We prefer constructing a motif in a top–down fashion, dividing it into the different motif parts connected by single strands on the top level (if present). Each motif part is then built individually by placing the appropriate building blocks next to each other. Another good way to go about is to build the structure iteratively from 5' to 3' end with gaps for any potential single strand connections. A third strategy is to identify the most prominent motif areas, such as a large multiloop, e.g. and commence the motif definition in those areas. When several related motifs are to be designed, parts described first may be stored and re-used. At the very end, single strands can be used to connect the different motif parts or to extend the 5' or 3' end.

2.6 Motif inspection

Via tooltips or a general information window (see Fig. 3), the user can access all the restrictions imposed on the individual building blocks of the motif. Furthermore, an online shape string which is updated according to every change made by the user, is displayed at the bottom of the graphical user interface. [The shape is an abstract view of the described structure (Giegerich et al., 2004b).]

2.7 Search parameters

Eventually, when the motif has been compiled into a search program, searches can be performed both locally and globally.
In the first case, the user specifies a sequence in which the best occurrence of the motif structure is found. In the second case, the entire sequence is folded according to the motif definition in the best possible way. Note that in both cases, the result might be empty, if the sequence and the defined motif are incompatible. (This often occurs when motifs are too restrictive.)

It is the responsibility of the motif designer to specify the intended search mode. These parameters can be set within a motif head building block that is represented by a red circle around the 5' end of the structure. Here, we can name the motif, choose global or local search mode, and restrict the overall motif size.

2.8 Output formats

We can store the motif definition to be reopened within the Locomotif system. Also, we can export graphics of the visual structure in formats such as png or jpg. Publication-quality output is provided via svg files. We intend to enhance these output modes with textual annotations next to the building blocks in the near future. This might not only make motif descriptions in the literature more uniform, but would effectively enable the reader to built on the experience captured in a successful published motif, to copy it, modify it, generate and use a similar matcher on further sequences of interest. Additionally, intermediate files generated from the graphics, such as the XML and ADP files described in Section 5, can be saved for later use.

2.9 Testing and refinement

Once a motif has been compiled into a matcher as described in Section 6, the program can be run within the editor using the ‘Run Matcher’ interface (Fig. 4). Here, the user specifies the sequences (either directly or via file upload) and the matcher by its ID. The results are presented in an extra window which opens automatically.

Typically, the design process is iterative. After some testing on small data, a user returns to the editor to refine or generalize the motif. Eventually, of course, the generated matcher will run as a stand-alone program.

3 THE LocomotIF DESCRIPTION LANGUAGE

The success of a matcher depends on how well known structural details can be expressed within the (graphical) motif description language. In addition to the basic building blocks described so far, Locomotif offers specialization via textual annotation, and generalization via compound building blocks.

3.1 Annotations

Annotations specialize building blocks with respect to size and sequence content.

The size of loop segments can be specified either as an exact value or as a range; for stems, the number of base pairs can be restricted accordingly. Also, global size restrictions can be given, effective upon the entire substructure rooted at a particular building block or upon the entire motif via the motif head.

For loop regions in all building blocks, a sequence motif can be given, and for all base pairs, both bases can be annotated. Any character of the IUPAC code can be used to include optionality. Yet, for each loop, we can store only one sequence motif. Two distinct motifs within one loop can be included by using ‘N’ characters to include a spacer between them. Of course, this feature is restricted to those cases where an exact number of bases in between two sequence motifs is known. In all cases, conflicting restrictions, such as an exact loop size of 2 combined with a sequence motif of length 4 are prohibited.

For stems, either a sequence motif on one strand or a sequence of specific base pairs can be defined. Since each base is included explicitly in the generated code, in both cases the beginning of the stem must coincide with the beginning of the stem building block. If this is not desired, we can work around this restriction by placing several stem building blocks next to each other. Then, the framing ones can be restricted to the appropriate size and the middle one contains the desired sequence motif. Furthermore, we can choose to define a looser stem type that may be interrupted by small loops of maximum 2 bases. This feature adds considerable flexibility and was suggested by Hofacker. Here, no sequence motifs can be given since we cannot be sure of the location of potential small loop interruptions. Hence, we have to rely on a similar work around as described above.

In case of the multiloop, an interactive visual interface is used to determine the number of helices combined by the multiloop. In a similar interface, each loop region or exit base pair can be selected for specification of sequence motifs or size restrictions.

Table 1 gives an overview on the exact types of restrictions which can be made for the different building blocks.

In case of the multiloop, an interactive visual interface is used to determine the number of helices combined by the multiloop. In a similar interface, each loop region or exit base pair can be selected for specification of sequence motifs or size restrictions.

Table 1 gives an overview on the exact types of restrictions which can be made for the different building blocks.

Each building block is aware of the fact that it has neighbors, but not of the kind of neighbors. Therefore, all annotations

![Fig. 4. A graphical user interface is available within Locomotif to run a generated matcher on the server.](image)
can only refer to an individual building block or a substructure, unaware of its content. We can therefore not specify constraints between two different building blocks. For example, we cannot restrict the size of a hairpin loop depending on the actual size of a neighboring stem. This is because the formal grammars underlying the generated matcher are context free.

3.2 ‘Don’t care’ structures

Compound building blocks generalize over structures built from basic blocks—they allow to define motifs with structural ‘don’t care’-parts. A ClosedStruct is an internal building block which represents any combination of successive stems, bulges and internal loops. A ClosedEnd represents a compound end of a motif part that can contain any substructure except for a plain single strand.

A ClosedStruct cannot be restricted (because the number and order of building blocks it contains is not defined), except for a global size restriction. With a ClosedEnd, global and local size restriction fall in one.

4 A CASE STUDY: OXYS

The correctness of the generated matcher need not to be evaluated. A generated matcher always finds the best hit according to its specification, so it is up to the user to define a motif suitable for the purpose. Here, we describe an example use case, demonstrating how the Locomotif system may be used in practice.

In Escherichia coli, a small untranslated RNA, OxyS, is expressed upon oxidative stress and affects the expression of multiple genes, one being fhlA (Argaman and Altuvia, 2000). The repression is mediated by base pairing between two hairpin loops of the OxyS RNA and two hairpin loops of the fhlA mRNA. One of these kissing hairpin interactions involves the Shine–Dalgarno sequence of fhlA (cf. Fig. 5). We will use this fact in the last step of our case study by annotating the hairpin loop sequences. Figures 2 and 3 show a screen shot of Locomotif during the OxyS motif construction.

We strive to locate the OxyS RNA in bacterial genomes with a set of TDMs. Starting with very few constraints, we reduce the number of false positive hits by stepwise inclusion of more details known from the literature and the Rfam consensus structure/sequence.

We incrementally build four matchers, with increasing number of restrictions, which are as follows:

TDM 1
- three stem-loop topology connected with single strands
- stem 1 may contain small internal loops and bulges
- overall motif size constrained to 150 nt
TDM 2
- three stem-loop topology connected with single strands
- one bulge loop and two internal loops in stem 1
- overall motif size constrained to 150 nt

TDM 3
- structure and constraints of TDM 2 plus
  - sizes of single strands, internal loops, bulge loops and hairpin loops are restricted according to the Rfam seed consensus
  - stem 1: local substructure maximal size: 50
  - stem 2: local substructure maximal size: 20
  - stem-loop 3: stem must have 4–8 base pairs, hairpin loop 3–5 bases

TDM 4
- structure and constraints of TDM 3 plus
  - hairpin loop 1 contains sequence motif ‘AACCCUU’
  - hairpin loop 3 contains sequence motif ‘UCCAG’

We apply our matchers to all 2582 intergenic regions of the *E. coli-K12* genome (NC_000913), larger than 20 nt. Including the reverse complement, this accounts for ~1.2 Mb. We liberally define a hit as a sequence folding into the motif with negative free energy. At most one hit is reported for one intergenic region.

As expected, the first matcher, TDM 1, locates a hit in nearly all intergenic regions. Only a few are either too small to harbor the motif, or their foldings have a small positive free energy. The second matcher, TDM 2, reduces the number of hits to 4109, mostly by its implicit minimal length constraint: a hit needs at least 42 nt to fold the desired loop and stem regions. This fact arises from the structural refinements and is deduced by the compiler (see below). Note that it is not explicitly annotated. The third matcher, TDM 3, leaves us with a reasonable number of 59 hits, a number small enough for further examination. Finally the last matcher, TDM 4, which specifies the exact sequences of the interaction sites, produces only one hit, located in the reverse complement of the intergenic region next to the *OxyR* gene. This is the exact locus of the *E. coli* OxyS gene.

Apparently, TDM 4 is perfectly designed for finding *E. coli* OxyS, but what about OxyS in other bacteria? To demonstrate the large scale applicability, we extracted the intergenic regions of all 450 bacterial genomes available at GenBank at this time. We obtained 181 324 regions (total 46 Mb) and applied TDM 4 on the forward and reverse complemented strand. Six hits were found in *E. coli* (AC_000091, NC_004431, NC_000913, NC_002655, NC_007946, NC_008563) and five in different *Shigella* species (NC_007613, NC_007606, NC_004337, NC_007384, NC_008258). Interestingly, only two entries for *Shigella* are annotated in Rfam, although the sequences producing the hits are 100% conserved. No hits were found for *Salmonella* species whose known OxyS genes contain a shorter sequence motif in one of their interaction site.

At this point, we have (1) verified that our matcher actually captures the known motif, (2) found some new instances where the motif has been perfectly conserved and (3) shown that no other (known) bacteria contain an instance of the OxyS motif at this level of conservation. All in all, this has taken three afternoons of work and consumed a total of 10 h computer time. From this point, we can now go off in different directions—either we tune the matcher to an organism of our interest where an *fhlA* homolog is known, relaxing the sequence pattern to only require the complement to the *fhlA* Shine–Dalgarno sequence. Or, we can relax the matcher stepwise both in structure and sequence pattern, to systematically explore the phylogenetic neighborhood of *E. coli*, *Shigella* and *Salmonella*.

A note on the efficiency: these particular examples all have an asymptotic run time of O\((n \ast m)\), *n* being the sequence length and *m* the maximal motif size. More complex motifs, which include multiloops or unrestricted loop regions would generate programs with O\((n \ast m^2)\) time requirements. For unbounded *m*, the asymptotic efficiency is the same as for general folding algorithms.

To give a concrete example: running TDM 4 against all intergenic regions of all 450 bacterial genomes required 1.5 h of computing time on a 1.8 GHz Dual-Opteron machine.

## 5 TRANSFORMING MOTIF GRAPHICS INTO SEARCH PROGRAMS

*Locomotif* employs four distinct language levels as depicted in Figure 6. First, motifs are defined in a graphical motif description language implemented in Java. For transport to the server, the graphical description is translated to an XML file representing both the motif structure and its annotations. In the next step, the XML code is translated to a corresponding ADP grammar. ADP is a domain specific language for expressing dynamic programming algorithms over sequence data (Giegerich et al., 2004a; Steffen and Giegerich, 2005). Finally, using the ADP compiler (ADPC) (Steffen, 2006), plain C code is generated. The resulting motif matcher can then be compiled from C towards the intended target machine, and applied on any number of target sequences.

### 5.1 Translation to XML

XML is suited as an intermediate storage layer, because the natural structure of an XML file can be used to represent RNA motifs. Internal to the *Locomotif* system, an RNA motif is essentially a sequence of building blocks in the order from 5′ to 3′ end, and the same order is seen in the XML code. For generating XML, the *Locomotif* editor traverses through the motif structure. Each building block is then translated to a corresponding JDOM\(^1\) element as it occurs. Using JDOM technology, we construct a global motif tree based on the individual elements and use an output method to obtain the XML code.

\(^1\)JDOM project for using XML from Java, http://jdom.org
5.2 Translation to ADP

In the next step, the XML code is translated to ADP. The ADP language implements algebraic dynamic programming, i.e. a dynamic programming algorithm is specified by a grammar and an evaluation algebra. In our case, the evaluation algebra consists of the thermodynamic model, and is adopted from other tools already implemented with ADP. The motif structure, including all annotations, is expressed as the grammar. Such a grammar for a specialized motif is typically much larger and more sophisticated than the grammars commonly used for RNA folding into arbitrary structures. Each motif element encoded in the XML file is parsed and the appropriate ADP code is created based on language templates for each building block. For details of this translation, the reader is referred to (Reeder and Giegerich, 2006).

5.3 Compilation phase

Finally, the ADP compiler integrates both the motif grammar and the necessary thermodynamic libraries and compiles a C program for motif search. ADPC is responsible for allocating dynamic programming tables and generating concrete recurrences from the grammar. This includes a number of optimizations that would be bothersome and error-prone for a human programmer.

For example, we noted with TDM 2 that a hit requires a minimum of 42 bases to form the motif. This restriction is not annotated explicitly, but is deduced from the implicit building block’s size restrictions. For example, the hairpin building block requires at least five bases: three for the loop region and two for the closing base pair. The ADP compiler discovers such implications and uses them to improve the efficiency of the generated code.

The resulting program is a specific matcher for the motif defined graphically and computes the structure of minimum free energy according to the given grammar and its thermodynamic algebras which are based on (Mathews et al., 1999).

6 CLIENT-SERVER ARCHITECTURE

The Locomotif system is available within a client-server architecture (shown in Fig. 7) that frees our user from dealing with any compilation issues. The resulting matchers can be run on the BiBiServ2 which provides enough storage and computational power to scan large multiple FASTA-files. Nevertheless, the server currently imposes a restriction on the maximum length per sequence of ~10000 bases, due to space requirements of the compiled matcher’s dynamic programming tables. (The downloaded matchers are only limited by your machine’s address space.)

The graphical editor is implemented as a Java application which is available on the webserver. Using Java Web Start, the editor can be run on the client side with any common browser and operating system. All necessary resources are automatically downloaded and updated and no installation routines are needed.

We rely on an intermediate XML layer for transportation by employing an XML schema that checks the content of the DOM document sent to our server. Thereby, we prevent any potential translation or compilation errors resulting from erroneous XML files. The use of XML as a transport layer is also a result of safety considerations. After all, programs provided by the users via this channel will be installed to run on our server. Using a strict XML schema definition adds a certain degree of protection against unsolicited XML submissions.

Bielefeld Bioinformatics Server, http://bibiserv.techfak.unibielefeld.de/locomotif
Once the XML information has been sent to the server, a unique identifier referring to the generated motif matcher is returned to the client. The ID is automatically stored within the running version of the Locomotif editor and can also be saved in a file in case the user wishes to reuse the matcher later. Optionally, the ID can also be sent to an email address and used on a submission web page.

The ADP compiler and the necessary libraries are maintained on the server side. Translation from XML via ADP and C to executable code are performed on the server. The intermediate C code is also made available for download.

As soon as the matcher compilation is finished, the resulting program can be run through the ‘Run Matcher’ interface of the Locomotif editor or via the web interface. Plain sequences or FASTA-files can be specified together with a matcher ID and are uploaded to the server. There, the ID is used to find the appropriate matcher and search the given sequence(s) for the motif it encodes. Results are presented in a popup window within the Locomotif editor or inside the browser.

Since the generated source code is available both on the ADP and C level, experienced programmers can fine-tune the code using features that are currently not supported by the editor. An example would be variable spacers between sequence motifs within one loop. We strongly recommend to perform such tuning on the ADP level. However, programmers enjoying the risks of subscript fiddling may also modify the generated C code. To facilitate this, the C code holds the original ADP code as comments.

7 CONCLUSION

Summing up, Locomotif offers a powerful graphical motif description language, from which motif search programs are generated automatically. Since the graphics are close to the commonly used drawings of RNA secondary structures, we hope that biologists readily adopt it, whether or not they have bioinformatics expertise. Great care has been taken in the interaction design to exclude the drawing of non-sense motifs. All motifs that can be composed can also be compiled and yield reliable matchers, faithful to the motif. Thus, users debugging or refining motifs are debugging or refining their own ideas rather than the generated programs.

A matcher generated from a motif is competitive to hand-written code, and probably more reliable. There is no question about the quality of the program—as the whole approach does not involve any heuristic assumptions, the matcher always finds exactly those sequences (or parts therein) that carry the motif. An evaluation of this new approach must answer the questions (1) whether the description language is powerful enough to describe relevant motifs both in sufficient detail and in sufficient generality, and (2) how well its intended usership adapts to the approach and uses the facilities of the language. Hence, such an evaluation lies in the hands of the community.

Providing the Locomotif system via the Bielefeld Bioinformatics Server, we are sure to collect not only questions and bug reports, but also positive feedback and suggestions for extensions. There are already a few items on our list of planned extensions:

- The most prevalent addition in our eyes is that of an optional construct. One wants to express that a particular substructure may either be represented within the motif, or missing completely. (Currently, two different motifs must be supplied, and two matchers generated and run.) A shaded visualization mode would be needed to represent optional submotifs on the screen.
- Sequence motifs can already allow some variation, using all characters of the IUPAC code. However, allowing general alignments between motif sequences and the...
target—with a specified scoring scheme and threshold—would be much more flexible.

- We plan to add a new building block for pseudoknots, as many RNA motifs are known to contain pseudoknots. Certain pseudoknots can be represented as tree-like structures, and integrate well with our structural graphics and the implementation of matchers via ADP. The class of pseudoknots recognized will be the same as in the program pknotsRG (Reeder and Giegerich, 2004) with the matcher running in $O(n^4)$ time and $O(n^2)$ space. With size restrictions imposed on the pseudoknot, taken from known examples, this will be efficient enough to search, say, all genes in a bacterial genome for programmed ribosomal frameshift signals.

- The implementation of dynamic programming algorithms allows for a certain space-time trade-off (Steffen and Giegerich, 2006). A complex motif may require a large number of tables, leading to prohibitive memory requirements. It is often possible to decrease the number of tables and recompute the intermediate results they would hold from other tables. This, of course, increases runtime. This increase can be a constant factor, but would lead to exponential runtime if too many tables were eliminated. The compiler can analyze how many tables can be avoided without compromising asymptotic efficiency. However, such trade-offs are yet inaccessible to a user of Locomotif.

The first three items are major language extensions. Their implementation effort lies not so much with the generation of enhanced matchers. The necessary techniques are available within the ADP language, and have already been used elsewhere. The effort lies mainly in extending the editor—not only with respect to new building blocks and their visualization, but also with the control of their interactive use to further ensure motif integrity.

Numerous minor improvements can be imagined. In this respect, we expect a continuous flow of input from Locomotif users. Locomotif’s internet address is http://bibiserv.techfak.uni-bielefeld.de/locomotif

ACKNOWLEDGEMENTS

We thank Peter Steffen for providing the ADP compiler used within the Locomotif system and we thank Jan Krüger for help on devising the client-server architecture and installing the system on the BiBiServ. Work of Janina Reeder was supported by DFG graduate college 635 Bioinformatik.

Conflict of Interest: none declared.

REFERENCES

Clote,P. et al. (2005) Structural RNA has lower folding energy than random RNA of the same dinucleotide frequency. RNA, 11, 578–591.