Supplemental material for the paper: Structator: fast index-based search for RNA sequence-structure patterns

Fernando Meyer Stefan Kurtz Rolf Backofen Sebastian Will Michael Beckstette

1 An example of bidirectional RSSP search

As an example of bidirectional search for RSSPs using affix arrays, we search for the RSSP $Q = (P, R)$ in the sequence $S$ of Figures 2 and 3, respectively, of the main document. We recall that $Q = (P, R)$ with $P = \text{NNNGCUNNN}$ and $R = (((\ldots)))$ represents a stem-loop structure of length $m = 10$ and $S = \text{AUAGCUGCUGCUGCA}$ has length 15. We start matching $P$ in $S$ by calling procedure bidir-search of Algorithm 2 as $\text{bidir-search}(\langle 0, 0 \rangle, 2, 3)$. That is, the algorithm matches the first position $P[3] = \text{U}$ of the loop region in left-to-right direction. Given that $X = F$ and $i < j$ (i.e. $0 < 15$) hold, it locates interval $v_x = \langle 0, 1 \rangle$ via binary search in the interval $0 - [0..15]$ of $\text{suffix}_F$. Analogously, the following recursive calls of bidir-search perform right $c$-extensions of $u = \text{U} = P[3..3]$ with characters $P[4] = \text{G}$, $P[5] = \text{C}$, and $P[6] = \text{U}$, by searching in the intervals $1 - [11..14]$, $2 - [12..14]$, and $3 - [12..14]$, respectively. After these extensions, the algorithm has located the affix-interval $v_x = \langle 0, 4 \rangle$ representing all occurrences of $U \text{GCU}$ in $S$ such that $\text{prefix}_v$ matches $u = P[3..6]$. We set $v = v_x$. Next, the algorithm performs a right $c$-extension of $u$ with the pairing position $c \in \varphi(P[7]) = \text{N}$. Therefore, it enumerates all possible $v_x$ such that $\text{prefix}_v = \text{prefix}_{v'}$ for some $d \in \varphi(c)$. We observe that $v_x = \langle 0, 5 \rangle$ with $\text{prefix}_v = \text{UGCU}$ is the only interval satisfying these conditions and conclude $d = \text{G}$. As an additional structural constraint, matches to positions 2 and 7 of $P$ shall form a base pair. To fulfill this constraint the algorithm first switches the search direction by locating the reverse interval $v'_x$ of $v_x$. The left boundary of $v'_x$ is determined with a lookup in table $\text{affk}_F$ as $\text{affk}_F[\text{home}_F([13..14])] = 5$ and the right boundary as 6. Further, we set $\ell_{\text{cp}} = \min \{ \ell_{\text{cp}}[r] \mid 13 < r \leq 14 \} = 6$ and calculate the context of $v'_x$ as $6 - 5 = 1$. Hence, the reverse interval of $v_x$ is determined as $v'_x = \langle 1, 6 \rangle$ with $\overline{\text{prefix}}_{v'_x} = \text{UGCUG}$ and we set $v = v'_x$. Now the only interval satisfying (1) $\overline{\text{prefix}}_{v'_x} = e \overline{\text{prefix}}_{v_x}, e \in \varphi(P[2])$, and (2) the complementarity condition between positions 2 and 7 of $P$, as required by the structure string $R$, is the interval $v_x = \langle 1, 7 \rangle$ with $\overline{\text{prefix}}_{v_x} = \text{CUGCU}$.
representing occurrences of substrings matching $P[2..7]$. Observe that $\overrightarrow{v_2}{[0]} = C$ and $\overrightarrow{v_2}{[5]} = G$ can form a base pair as demanded by $R[2]$ and $R[7]$. Consequently, $\overrightarrow{v_2}$ matches $(P[2..7], R[2..7])$ and therefore we set $v = v_x$. In the next step the algorithm performs another left $c$-extension of $\overrightarrow{v}$ by some $c \in \varphi(P[1] = N)$ leading to interval $v_x = (1, 8 - [5..6], R)$ with $\overrightarrow{v_x} = \text{GCUGCUGC}$ representing occurrences of substrings matching $P[1..7]$. We set $v = v_x$. To match a character $d \in \varphi(c)$ that is complementary to $\overrightarrow{v}[0] = G$ the algorithm performs a right $c$-extension of $\overrightarrow{v}$ using a character $c \in \varphi(P[8])$. Because the context of $v$ is larger than zero, it consumes the context and remains in table $\text{sufr}$. That is, $X = R$. The resulting interval after performing the right $c$-extension is $v_x = (0, 8 - [5..6], R)$ with $\overrightarrow{v_2} = \text{GCUGCUGC}$. Observe that $\overrightarrow{v_2}{[0]} = G$ and $\overrightarrow{v_2}{[7]} = C$ can form a base pair and thus $v_x$ represents occurrences of substrings of $S$ matching $(P[1..8], R[1..8])$. We set $v = v_x$. The next operation is a left $c$-extension by some $c \in \varphi(P[0] = N)$. Hence, the algorithm enumerates all intervals $v_x$ such that $\overrightarrow{v_x} = \overrightarrow{v}d, d \in \varphi(c)$. There are two intervals satisfying these conditions. Namely, $v_{x_1} = (0, 9 - [5..5], R)$ with $\overrightarrow{v_{x_1}} = \text{AGCUUGCUC}$ and $v_{x_2} = (0, 9 - [6..6], R)$ with $\overrightarrow{v_{x_2}} = \text{UGCUUGCUC}$. We set $v_1 = v_{x_1}$ and $v_2 = v_{x_2}$ and continue by processing $v_1$, which represents occurrences of $\overrightarrow{v_1} = \text{AGCUUGCUC}$ in $S$. Because $\overrightarrow{v_1}$ is a unique substring of $S$, for the following right $c$-extension by some $c \in \varphi(P[9] = N)$ we can directly evaluate $S^R[\text{sufr}[5] - 1] = U$. Bases ($\overrightarrow{v_1}{[0]} = A, U$) are complementary, hence we set $v_x = (-1, 9 - [5..5], R)$ and observe that occurrences of substring $\overrightarrow{v_2} = \text{AGCUUGCUC}$ of $S$ matching $(P[0..9], R[0..9])$ and that the boundaries of $Q$ have been reached. With this, in the following recursion the algorithm reports a matching position of $Q$ via a lookup in table $\text{sufr}$ as $\text{sufr}[5] + (-1) = 4 - 1 = 3$, where $-1$ is the context of $v_x$ that has to be added to $\text{sufr}$. Note that, because $X = R$, $3$ is a position in $S^R$. Now the algorithm backtracks to interval $(0, 8 - [5..6], R)$ and continues to perform a right $c$-extension of interval $v_2$ by some $c \in \varphi(P[9])$. Again, $\overrightarrow{v_2} = \text{UGCUUGCUC}$ is a unique substring of $S$ and we can directly evaluate $S^R[\text{sufr}[6] - 1] = A$. Since bases ($\overrightarrow{v_2}{[0]} = U, A$) can pair, we set $v_x = (-1, 9 - [6..6], R)$ with $\overrightarrow{v_2} = \text{UGCUUGCUGCA}$ representing occurrences of substrings of $S$ matching $(P[0..9], R[0..9])$. The boundaries of $Q$ have been reached again and in the following recursion the algorithm reports another matching position of $Q$, precisely $\text{sufr}[6] + (-1) = 1 - 1 = 0$. There are no further intervals to process and the search ends. In summary, bidir-search has found two occurrences of $Q$ in $S$.

2 Comparison of two implementations of bidirectional pattern search

We measured the speedup of Structator running in BIDsearch mode over ONLsearch and compared the results with previously reported measurements [1]. Because the implementation used by Strothmann [1] is not available (personal communication), we calculated relative speedups based on the absolute running times reported in [1]. We note that the measurements of [1] were performed on different hardware. This can, according to our observations, significantly influence the performance of BIDsearch.
See Table S1 for the results of the comparison of BIDsearch with the method of [1]. For a description of the used RSSPs see [1]. The search was performed in the genomes of P. horikoshii (GenBank Acc.: NC_000961, 1.7 MB) and E. coli (GenBank Acc.: AC_000091, 4.5 MB), which were also used in [1]. Additionally we included with P. vampyrus (GenBank Acc.: ABRRP0000000, 1.9 GB) a larger eukaryotic genome in this experiment.

Surprisingly, with the RSSPs ACloop(5), ACloop(10), and ACloop(15) taken from [1], which describe a loop consisting of 5 (10 and 15) repetitions of AC the speedup of the affix array based method of [1] decreased with increasing loop length. This is a behavior which is opposite to our observations (see Figure 8 of the main document). We also noticed that BIDsearch obtained a higher speedup when searching for RSSP Hpin2 in E. coli than the method of [1] but not when searching in the smaller genome of P. horikoshii. This observation remains unclear and cannot be further investigated due to unavailability of the implementation used in [1].

3 A bidirectional search algorithm supporting variable-length RSSPs

Algorithm 2 of the main document matches fixed-length RSSPs. We here present an extension of it also capable of matching RSSPs with loop region allowing a variable number of additional extensions with ambiguous characters N to the left and to the right. In combination, also stem region of variable length is supported. We observe that this extended version is as efficient as the original algorithm supporting fixed-length RSSPs. Additional computation time is only required for the traversal of additional affix-intervals due to the increased sensitivity.

Before describing the algorithm, we define this extension of RSSPs. A variable-length RSSP \( Q \) consists of an RSSP \((P, R)\) and parameters \( \text{maxleftloopextent} (\text{mllex}) \), \( \text{maxrightloopextent} (\text{mrlex}) \), and \( \text{maxstemlength} (\text{msl}) \). \text{mllex} and \text{mrlex} denote the maximum number of respective left and right extensions of the loop region specified in \( R \) and \text{msl} denotes the maximum number of base pairs in the stem. The minimum length of occurrences of \( Q \) is \( m = |P| = |R| \). For examples of variable-length RSSPs, see Figure 12 (E) until (H) of the main document.

To keep the code simple, we split the original algorithm into two procedures. (i) First the loop region of a given variable-length RSSP \( Q \) is matched with procedure bidir-search-loop (see Algorithm 3, Figure S1). (ii) Next, the stem region is matched with procedure bidir-search-stem (see Algorithm 4, Figure S2). Note that bidir-search-stem is very similar to Algorithm 2 of the main document. Prior to the search for \( Q \), the following variables are set: \text{loopstart}, \text{minloopstart}, \text{loopend}, \text{maxloopend}, \text{minbps}, and \text{maxbps}. These variables store the following information. \text{loopstart} (\text{lopend}) stores the position of the base occurring in the left-most (right-most) position of the loop described by the structure string \( R \) in 5’ to 3’ direction, \( \text{minloopstart} = \text{loopstart} - \text{mllex}, \text{maxloopend} = \text{lopend} + \text{mrlex}, \) and \( \text{minbps} (\text{maxbps} = \text{msl}) \) is the minimum (maximum) number of base pairs occurring in \( Q \). It holds: \( \text{minloopstart} \leq \text{loopstart} \leq \text{maxloopstart} \leq \text{loopend} \leq \text{maxloopend} \leq \text{lopend} \leq \text{maxbps} \).
### Table S1: Comparison of speedup of Structator’s BIDsearch over ONLsearch (column BID vs. O) and the speedup of affix array based search over searching on the plain text as reported in [1] (column STR). The respective search times of BIDsearch (column BID) and ONLsearch (column ONL) are shown in milliseconds. For P. vampyrus only measurements for Structator are available.

|              | P. horikoshii (1.7 MB) | E. coli K12 (4.5 MB) | P. vampyrus (1.9 GB) |
|--------------|------------------------|----------------------|----------------------|
| **RSSP**     | **ONL** BID Bvs.O STR  | **ONL** BID Bvs.O STR | **ONL** BID Bvs.O STR |
| Hpin1        | 169.61 65.59 2.59 10.26 | 432.94 141.84 3.05 12.17 | 172,913.36 9,520.39 18.16 - |
| Hpin2        | 33.34 0.27 123.48 155 | 88.61 0.45 196.91 99.25 | 34,702.67 141.84 3.05 12.17 |
| Hloop(5)     | 214.8 166.94 1.29 14.6 | 552.67 372.57 1.48 18.09 | 219,547.76 9,520.39 18.16 - |
| Hloop(10)    | 331.96 1,412.64 0.23 2.13 | 842.32 3,235.11 0.26 2.43 | 335,928.97 48.85 710.39 - |
| ALoop(5)     | 59.07 4.43 13.33 182 | 152.87 9.91 15.43 815 | 64,053.16 825.79 77.57 - |
| ALoop(10)    | 58.71 0.89 65.92 1.3 | 152.01 1.86 81.73 1.38 | 64,199.98 278.76 230.31 - |
| ALoop(15)    | 58.67 0.89 65.92 1.3 | 152.01 1.86 81.73 1.38 | 64,199.98 278.76 230.31 - |

---

**loopstart ≤ loo pend ≤ maxloopend.** Note that minloopstart can be negative. As an example, let \( R = (((...))) \); \( mllex = 4 \); and \( mrlex = 1 \). Then \( loopstart = 3 \), \( minloopstart = -1 \), \( loo pend = 6 \), \( maxloopend = 7 \), and \( minbps = 3 \). To match \( Q \), procedure bidir-search-loop is initially called as \( bidir-search-loop((0,0-\mathbb{[}0..n\mathbb{]}),F),r_0-1,r_0,\text{true}) \), where \((0,0-\mathbb{[}0..n\mathbb{]}),F)\) is an affix-interval, \(r_0\) is any position in the loop region of \( Q \), and parameter true states that the pattern can be extended to the right. Procedure bidir-search-loop makes a call to bidir-search-stem whenever substrings of minimum length \( loo pend−loopstart+1 \) matching the loop in the searched database are found. If \( Q \) has no base pairs, i.e. \( msl = 0 \), it instead immediately reports the matching positions. The call to bidir-search-stem is made as \( bidir-search-stem(v',loopstart-1,loopend+1,0) \), where \( v' \) is the affix-interval representing all occurrences of substring \( \bar{v}' \) in the searched database matching the loop region of \( Q \), positions \( loopstart-1 \) and \( loopend+1 \) denote the inner-most base pair \( (loopstart-1,loopend+1) \) of the pattern, and \( 0 \) is the number of currently matched base pairs. Procedure bidir-search-stem reports matching positions of \( Q \) whenever the boundaries of the RSSP are reached or \( minbps < bpcount < maxbps \) holds.

### References

[1] D. Strothmann. The affix array data structure and its applications to RNA secondary structure analysis. *Theor. Comput. Sci.*, 389(1-2):278–294, 2007.

[2] T. Schnattinger, E. Ohlebusch, and S. Gog. Bidirectional search in a string with wavelet trees. In *Proceedings of the 21st Annual Symposium on Combinatorial Pattern Matching*, volume 6129, pages 40–50. Springer, 2010.

[3] K. Darty, A. Denise, and Y. Ponty. VARNA: Interactive drawing and editing of the RNA secondary structure. *Bioinformatics*, 25(15):1974–1975, 2009.
Algorithm 3: \textit{bidir-search-loop}(affix-interval \( v = (k, l - [i, j], X) \), pos \( r \), pos \( r' \), allowrightext)

1: if \( r' \leq \text{maxloopend} \) and allowrightext = true

\hspace{1em} // perform right extension
2: \hspace{2em} if \( r' > \text{loopend} \) then
3: \hspace{3em} \( \text{chr}' = 'N' \)
4: \hspace{3em} else
5: \hspace{4em} \( \text{chr}' = P[r'] \)
6: \hspace{2em} end
7: \hspace{1em} foreach \( v' \) such that \( d \in \varphi(\text{chr}') \) and \( \text{chr}' = \mathcal{V}d \) do
8: \hspace{2em} if \( msl = 0 \) then // if entire pattern is single stranded
9: \hspace{3em} report match at positions \( \text{suf}_X[i + k, ... , j + k] \)
10: \hspace{3em} return
11: \hspace{2em} else // otherwise loop of length \( r' - r + 1 \) was matched
12: \hspace{3em} \hspace{1em} // so extend stem region
13: \hspace{3em} \hspace{2em} \text{bidir-search-stem}(v', \text{loopstart} - 1, \text{loopend} + 1, 0)
14: \hspace{2em} end
15: \hspace{1em} end
16: \hspace{1em} if \( r' + 1 \leq \text{maxloopend} \) then
17: \hspace{2em} \text{bidir-search-loop}(v', r, r' + 1, true)
18: \hspace{1em} end
19: \hspace{1em} if \( r' + 1 > \text{loopend} \) then
20: \hspace{2em} \text{bidir-search-loop}(v', r, r' + 1, false)
21: \hspace{1em} end
22: \hspace{1em} end
23: \hspace{1em} else if \( r \geq \text{minloopstart} \) then
24: \hspace{2em} // perform left extension
25: \hspace{3em} if \( r < \text{loopstart} \) then
26: \hspace{4em} \( \text{chr} = 'N' \)
27: \hspace{4em} else
28: \hspace{5em} \( \text{chr} = P[r] \)
29: \hspace{4em} end
30: \hspace{3em} end
31: \hspace{3em} foreach \( v' \) such that \( d \in \varphi(\text{chr}) \) and \( \text{chr} = \mathcal{V}d \) do
32: \hspace{4em} if \( r - 1 < \text{loopstart} \) and \( r' > \text{loopend} \) then
33: \hspace{5em} if \( msl = 0 \) then // if entire pattern is single stranded
34: \hspace{6em} report match at positions \( \text{suf}_X[i + k, ... , j + k] \)
35: \hspace{6em} return
36: \hspace{5em} else // otherwise loop of length \( r' - r + 1 \) was matched
37: \hspace{6em} \hspace{1em} // so extend stem region
38: \hspace{6em} \hspace{2em} \text{bidir-search-stem}(v', \text{loopstart} - 1, \text{loopend} + 1, 0)
39: \hspace{5em} end
40: \hspace{4em} end
41: \hspace{3em} bidir-search-loop(v', r - 1, r', allowrightext)
42: \hspace{2em} end
43: \hspace{1em} end
44: \hspace{1em} end

Figure S1: Bidirectional recursive matching of the loop region of a variable-length RSSP using an affix array. Procedure \textit{bidir-search-loop} searches for an RSSP \((P, R)\) defined with additional variables \textit{maxleftloopextent} (\textit{mllex}) and \textit{maxrightloopextent} (\textit{mrlex}) denoting the maximum number of left and right extensions of the loop specified in \( R \), respectively, and \textit{maxstemlength} (\textit{msl}) denoting the maximum number of base pairs. Used variables \textit{loopstart}, \textit{minloopstart}, \textit{loopend}, and \textit{maxloopend} are preset according to structure string \( R \), \textit{mllex}, and \textit{mrlex} (see text). \textit{bidir-search-loop} calls procedure \textit{bidir-search-stem} (see Algorithm 4) whenever substrings of minimum length \( \text{loopend} - \text{loopstart} + 1 \) matching the loop are found.
Algorithm 4: \textit{bidir-search-stem}(affix-interval $v = (b, \ell - [i..j], X)$, pos $r$, pos $r'$, bpcount)

1. if $(r < 0$ and $r' \geq m) \text{ or } (\text{minbps} < \text{bpcount} < \text{maxbps})$ then
2. \hspace{1em} report match at positions $\text{suf} X[i+k, ...,$ $\text{suf} X[j+k]$.
3. end
4. if $(\text{minbps} < \text{bpcount} < \text{maxbps} \text{ or } (r \geq 0 \text{ and } r' < m \text{ and } R[r] = '($ \text{ and } R[r'] = '))$ then
5. if $\text{minbps} < \text{bpcount} < \text{maxbps}$ then
6. \hspace{1em} $\text{chr} = 'N'$
7. \hspace{1em} $\text{chr}' = 'N'$
8. else
9. \hspace{1em} $\text{chr} = P[r]$
10. \hspace{1em} $\text{chr}' = P[r']$
11. end
12. if $X = R$ then
13. \hspace{1em} // perform left extension first
14. \hspace{2em} foreach $v'$ such that $d \in \phi(\text{chr})$ and $\overline{v'} = d\overline{v}$ do
15. \hspace{3em} bidir-search-stem($v'$, $r-1$, $r'+1$, $\text{bpcount} + 1$)
16. end
17. else if $r' < m$ and $R[r'] = ')$ and $(X = F \text{ or } r < 0 \text{ or } R[r] \neq ':')$ then
18. \hspace{1em} foreach $v'$ such that $d \in \phi(P[r])$ and $\overline{v'} = d\overline{v}$ do
19. \hspace{2em} bidir-search-stem($v'$, $r$, $r'+1$, $\text{bpcount}$)
20. end
21. else if $r \geq 0$ and $R[r] = '(:$ then
22. \hspace{1em} foreach $v'$ such that $d \in \phi(P[r])$ and $\overline{v'} = d\overline{v}$ do
23. \hspace{2em} bidir-search-stem($v'$, $r-1$, $r'$, $\text{bpcount}$)
24. end
25. \end{algorithm}

Figure S2: Bidirectional recursive matching of the stem region of a variable-length RSSP using an affix array. Procedure \textit{bidir-search-stem} is called by procedure \textit{bidir-search-loop} (see Algorithm 3) and extends substrings $\overline{v}$ matching the loop region of the RSSP ($P, R$) to substrings matching also the stem. Used variables $\text{minbps}$ and $\text{maxbps}$ are preset according to structure string $R$ and variable $\text{maxstemlength (msl)}$ (see text).
| Organism        | Genome size | sufF | lcPF | lcpeF | aflkF | sufR | lcPR | lcpeR | aflkR |
|-----------------|-------------|------|------|-------|-------|------|------|-------|-------|
| C. elegans      | 100.29      | 401.14 | 100.29 | 6.29 | 401.14 | 401.14 | 100.29 | 6.29 | 401.14 |
| A. thaliana     | 119.67      | 478.67 | 119.67 | 8.85 | 478.67 | 478.67 | 119.67 | 8.85 | 478.67 |
| D. melanogaster | 168.74      | 674.95 | 168.74 | 94.34 | 674.95 | 674.95 | 168.74 | 94.34 | 674.95 |
| C. intestinalis | 173.52      | 694.02 | 173.50 | 28.03 | 694.02 | 694.02 | 173.50 | 28.03 | 694.02 |
| O. sativa       | 374.33      | 1,497.33 | 374.33 | 71.05 | 1,497.33 | 1,497.33 | 374.33 | 71.05 | 1,497.33 |
| M. gallopavo    | 1,087.50    | 4,349.99 | 1,087.50 | 2.01 | 4,349.99 | 4,349.99 | 1,087.50 | 2.01 | 4,349.99 |
| G. gallus      | 1,108.48    | 4,433.93 | 1,108.48 | 98.86 | 4,433.93 | 4,433.93 | 1,108.48 | 98.86 | 4,433.93 |
| D. rerio        | 1,481.32    | 5,925.08 | 1,481.27 | 457.26 | 5,925.08 | 5,925.08 | 1,481.27 | 457.26 | 5,925.08 |
| X. tropicalis   | 1,510.98    | 6,043.63 | 1,510.91 | 310.89 | 6,043.63 | 6,043.63 | 1,510.91 | 310.89 | 6,043.63 |
| P. vampyrus     | 1,999.71    | 7,998.82 | 1,999.71 | 170.84 | 7,998.82 | 7,998.82 | 1,999.71 | 170.84 | 7,998.82 |

Table S2: Sizes in megabytes of the different tables the affix array consists of for the genomes used in Experiment 1. lcpeF and lcpeR are the exception tables storing entries with value larger than 255 which cannot be stored in tables lcPF and lcPR, respectively. In tables lcpeF and lcpeR, each entry consumes 8 bytes.

| Organism        | Genome size | BWI |
|-----------------|-------------|-----|
| C. elegans      | 100.29      | 157.96 |
| A. thaliana     | 119.67      | 188.59 |
| D. melanogaster | 168.74      | 295.37 |
| C. intestinalis | 173.52      | 279.83 |
| O. sativa       | 374.33      | 602.21 |
| M. gallopavo    | 1,087.50    | 1,800.88 |
| G. gallus      | 1,108.48    | 1,757.84 |
| D. rerio        | 1,481.32    | 2,424.81 |
| X. tropicalis   | 1,510.98    | 2,309.24 |
| P. vampyrus     | 1,999.71    | 3,282.55 |

Table S3: Size in megabytes of the bidirectional wavelet index (BWI) [2] for different genomes.
Figure S3: Distribution of speedup factors of BIDsearch over RNABOB (yellow) and RNAMotif (green) when searching for 397 RSSPs in RFAM10 consisting of ∼622 megabases of RNA sequence data. The red and blue curves show the values of one minus the empirical cumulative distribution function of the speedup factors distributions. That is, for a given speedup factor $S$ they show the fraction of RSSPs for which BIDsearch obtained a speedup greater than $S$ over RNAMotif (red curve) and RNABOB (blue curve), respectively. We observed that BIDsearch is more than 50,000 times faster than RNABOB and RNAMotif for the majority of the patterns (see intersection point of dashed lines). Moreover, the total search time required by BIDsearch is dominated by only a small number of patterns describing large unconserved loop regions.
Figure S4: Consensus secondary structure of the CTV_rep_sig family (RFAM Acc.: RF00193) visualized with the VARNA program [3] and SSD in Structator syntax describing this family. The 8 given RSSPs correspond to the colored stem-loops HP1 - HP8. Positions at which sequence information is used in the descriptor are marked with an asterisk.
Table S4: Results of \textit{Structator} searches on RFAM10 (1,446 families; 3,192,599 sequences) using SSDs describing 42 Rfam families. The manually compiled SSDs used in this experiment are available on the \textit{Structator} website. They were designed to be highly specific and consist of 397 RSSPs in total with an average of 9.45 RSSPs per SSD. These are the same 397 RSSPs used in section "Searching large sequence databases" in the main document. Columns 2, 3, and 5 show the number of sequences containing high-scoring global chains, the numbers of true positives (TP), specificity as false positives (FP), and false negatives (FN), respectively. Sensitivity is computed as 
\[
\frac{\#TP}{\#TP + \#FN},
\]
accuracy as 
\[
\frac{\#TP + \#TN}{\#TP + \#FP + \#FN + \#TN},
\]
and precision as 
\[
\frac{\#TP}{\#TP + \#FP}.
\]
Observe that these values strongly depend on the used SSD. The number of RSSPs constituting an SSD is given in column 14. Column 11 shows the minimal required length of a chain to be considered a matching chain. Total running times of \textit{Structator} operating in \textit{BIDsearch} and \textit{ONLsearch} mode are given in columns 12 and 13, respectively. Column 14 shows \textit{BIDsearch}'s speedups over \textit{ONLsearch}. The running time required for chaining of RSSP matches is listed in column 15. Observe that the sum of running times does not match the times needed for searching with the 397 single RSSPs reported in the main document because here each SSD was searched using a separate \textit{Structator} program call.
Figure S5: (A) SSD for HAR1F RNA family consisting of RSSP1, RSSP2, and RSSP3 in Structator syntax. RSSPs were built from stem-loops HP1, HP2, and HP3 shown in (C). (B) RNAMotif descriptor for the same structural elements. Secondary structure drawing shown in (C) was generated with VARNA [3].