Supplemental material for the paper: Fast online and index-based algorithms for approximate search of RNA sequence-structure patterns

Fernando Meyer  Stefan Kurtz  Michael Beckstette

1 Pseudocode for algorithms LESAAAlign and LGSlinkAlign

The pseudocode for algorithm LESAAAlign is given in Algorithm 1 (Figure S1). LESAAAlign traverses the suffix array suf of the target sequence S top down, beginning with the lexicographically smallest suffix $S_{suf[iSuffix]}$, where $iSuffix = 1$ at this stage. During the traversal, it computes the sequence-structure edit distance $dist(Q, S_{suf[iSuffix]}[1..p_{iSuffix}])$ between the RSSP Q and the prefix of length $p_{iSuffix}$ of each suffix $S_{suf[iSuffix]}$, for $1 \leq iSuffix \leq n$. This computation is done by function computeDP in line 5. The input parameters of computeDP are the computed DP matrices, the index $iSuffix$ of the current suffix, the length $iLcp$ of the common prefix between the last processed suffix and the current suffix, and the last computed pattern region $Q[x..y]$ denoted lastRegion in the code. The last two variables are used to avoid recomputation of entries of DP matrices. Function computeDP returns a boolean value, stored in bMatched, stating whether the pattern was matched, and the last newly computed region lastRegion. lastRegion.r is the right boundary of the last computed pattern region $Q[x..y]$ and is used to compute variable iLcpCheck in line 6. iLcpCheck, in turn, is used to check whether suffixes of the suffix array sharing a common prefix can be skipped. If bMatched is true, matches are reported by function reportMatch in lines 8 and 18.

The pseudocode for algorithm LGSlinkAlign is given in Algorithm 2 (Figure S2). LGSlinkAlign traverses the suffix array in two combined strategies: top down and following suffix links. This is managed in the code with two main while-loops, where an outer loop (lines 3 to 47) performs the top down traversal and an inner loop (lines 13 to 46) performs the traversal via suffix links. To keep track of the last processed suffix via top down suffix array traversal, the index of this suffix is stored in variable iSuffixTopDown. To keep the code short, all alignment computations are performed only in the inner loop, distinguishing the strategy by which suffixes are traversed according to the boolean variable bFollowedSuffixLink. This variable is set to true (line 41) when the inner loop iterates and to false (line 4) when the iteration breaks. When bFollowedSuffixLink is false, the same computeDP function used by the LESAAAlign algorithm is applied. Otherwise function computeLastDPColumns is applied. This function does not use lcp information, but takes advantage of the fact that the prefix of the current suffix, determined in line 36 by following a suffix link, is equal to the previously processed suffix prefix, except by its last character. This property of the suffix prefix allows to reuse already computed entries of matrices from the previously processed suffix prefix, requiring for this only one shift of the DP matrices. This is done by function shiftDP in line 42. While traversing the suffix array, processed suffixes are marked in the vtab table. This allows to avoid processing the same suffixes multiple times. In addition to these processed suffixes, non-contiguous suffixes of the suffix array that are known not to contain matches to RSSP Q are also marked in this table. This is possible when pattern Q, for the current suffix, has an unaligned prefix of length iUnalignedPrefixLength > 0. For determining iUnalignedPrefixLength in line 31, value lastRegion.l is used. This value is the left boundary of the last computed pattern region $Q[x..y]$. Marking the additional suffixes in vtab is performed by function markSuffixes (see Figure S3).
Algorithm 1: LESAAlign

```
input: Index tables suf and lcp of sequence S, RSSP Q
output: Matching positions of Q in S
iSuffix := 1
iLcp := 0
lastRegion := undefined
while iSuffix ≤ n do
    (bMatched, lastRegion) := computeDP(DP, iSuffix, iLcp, lastRegion)
    iLcpCheck := lastRegion.r + d
    if bMatched then
        reportMatch(Q, S, iSuffix) //Match found at position suf[iSuffix] of S
        iSuffix := iSuffix + 1
    end
    if iSuffix ≤ n then
        iLcp := lcp[iSuffix]
        while iSuffix ≤ n and lcp[iSuffix] ≥ iLcpCheck do
            if lcp[iSuffix] < iLcp then
                iLcp := lcp[iSuffix] //Store the smallest lcp value of the skipped interval
            end
        end
        if bMatched then
            reportMatch(Q, S, iSuffix) //Match found at position suf[iSuffix] of S
        end
        iSuffix := iSuffix + 1
    end
end
```

Figure S1: Pseudocode for algorithm LESAAlign. For details, see text above and the algorithm description in the main document.

This function receives as parameter a starting index iSuffix, iUnalignedPrefixLength, and the required length iLcp of the common prefixes of the suffixes to be marked. The function then traverses the suffix array top down and bottom up, marking all possible suffixes in vtab.

2 Influence of allowed edit costs and number of indels on search time

We describe an experiment comparing the running times of algorithms LGSlinkAlign, LESAAlign, LScanAlign, and ScanAlign to search in RFAM10.1, similar to the first benchmark described in the main document. This time we set (1) \( K = d \) varying the values in the interval \([0, 7]\), (2) \( K = 7 \) varying \( d \) in the interval \([0, 7]\), and (3) \( d = 0 \) varying \( K \) in the interval \([0, 7]\). We use RSSP \( Q = \text{CARGAYSNVNNNDGCRKYCCHVHRWNRUCYAG}(.(((....((((.....))))..)))).) \) of length \( m = 33 \) describing a stem-loop substructure of Rfam family Cripavirus internal ribosome entry site (Acc.: RF00458) [1]. The secondary structure of this family and the substructure originating the pattern can be visualized in Figure S5, where the substructure is denoted pt4. For the results of this experiment, see Figure S4 and Tables S1, S2, and S3. LGSlinkAlign is the fastest algorithm with measured speedup factors over ScanAlign (LScanAlign) in the range of 160.6 for \( K = d = 0 \) to 3.3 for \( K = d = 7 \) (17.8 for \( K = d = 1 \) to 3.3 for \( K = d = 7 \)). In a comparison between the two online algorithms, LScanAlign is faster than ScanAlign up to a cost threshold of \( K = 6 \) and for any value of \( K \) in case no indels are allowed, i.e. \( d = 0 \). LESAAlign is only faster than the online algorithms for up to \( (K = d) \leq 5 \) and \( K = 7 \) and \( d \leq 3 \). For higher cost thresholds and allowed indels, its performance decreases significantly. We explain this behavior with the increased reading depth in the suffix array implicated by \( K \) and \( d \) and the reduced number of suffixes sharing a common prefix that can be skipped.
Algorithm 2: LGSlinkAlign

input: Index tables suf, lcp, suf⁻¹, and vtab of sequence S, RSSP Q
output: Matching positions of Q in S

iSuffixTopDown := 1
lastRegion := undefined

while iSuffixTopDown ≤ n do //Begin traversing suffix array top down
    bFollowedSuffixLink := false
    iLcp := lcp[iSuffixTopDown]
    while vtab[suf[iSuffixTopDown]] do //Skip already visited suffixes
        iSuffixTopDown := iSuffixTopDown + 1
    end
    if iLcp > lcp[iSuffixTopDown] then //Store the smallest lcp value of the skipped interval
        iLcp := lcp[iSuffixTopDown]
    end
    iSuffix := iSuffixTopDown
    while not vtab[suf[iSuffix]] do
        if bFollowedSuffixLink then //Current suffix was obtained via a suffix link
            (bMatched, lastRegion) := computeLastDPColumns(DP, iSuffix, lastRegion)
        else //Current suffix was obtained via the top-down suffix array traversal
            (bMatched, lastRegion) := computeDP(DPTopDown, iSuffix, iLcp, lastRegion)
        end
        iLcpCheck := lastRegion.r + d
        repeat
            vtab[suf[iSuffix]] := true
            if bMatched then
                reportMatch(Q, S, iSuffix) //Match found at position suf[iSuffix] of S
            end
            iSuffix := iSuffix + 1
            if iSuffix > n or vtab[suf[iSuffix]] then
                break
            end
        until lcp[iSuffix] ≥ iLcpCheck
        iSuffix := iSuffix − 1
        if iUnalignedPrefixLength > 0 then
            markSuffixes(link[iSuffix, iUnalignedPrefixLength], iUnalignedPrefixLength, lastRegion.r + d – iUnalignedPrefixLength)
        end
        if |S[iSuffix]| ≥ m − d then //If suffix is not shorter than the minimum required length
            if not bFollowedSuffixLink then
                DP := DPTopDown
            end
            bFollowedSuffixLink := true
            shiftDP(DP)
        else //Leave large while-loop and traverse suffix array top down
            break
        end
    end
end

Figure S2: Pseudocode for algorithm LGSlinkAlign. For details, see text above and the algorithm description in the main document.
Figure S3: Function `markSuffixes` used by algorithm `LGSlinkAlign` to mark processed suffixes in table `vtab`. For details, see text above.

3 Comparisons between RaligNAtor and RNAMotif in terms of sensitivity and specificity

RNAMotif [2] is one of the most popular tools for approximate matching of RSSPs supporting the operations replacement and mispairing (which corresponds to the arc breaking operation defined in the main document). A number of allowed replacements and mispairings, which we here simply denote errors, can be specified for each part of the structure along with an overall number constraining the entire structure. However, the arc altering and arc removing operations are not supported. Also, insertions and deletions are only supported by using regular expression quantifiers. This means that the user has to know in advance for which positions of the pattern such operations can occur.

In this experiment we first analyze the results of RaligNAtor when searching RFAM10.1 with the tRNA (Acc.: RF00005) RSSP shown in Figure 6 of the main document. In particular, we show the importance of secondary structure information incorporated in the search for homologous sequences by varying the cost of edit operations on base pairs. Secondly, we compare the results obtained by RaligNAtor with the results of RNAMotif version 3.07 when searching with an equivalent RNAMotif pattern. For the used RNAMotif descriptor, see Figure S6.

For the searches with RaligNAtor, we vary the cost threshold $K$ and the number of allowed indels $d$ between 0 and 25 in steps of 5. We use operation costs $\omega_d = \omega_m = \omega_b = \omega_a = 1$ and $\omega_r = 2$. Then we increase the costs of the operations arc breaking, arc altering, and arc removing. More precisely, we set $\omega_d = \omega_m = \omega_b = \omega_a = 1$, $\omega_r = 2$, and $\omega_r = 3$. The results are shown in Table S11. Unsurprisingly, we observe that RaligNAtor’s sensitivity increases with increasing values of $K$ and $d$. However, for low costs of the operations on base pairs, its specificity decreases considerably when $K$ and $d$ are increased from 20 to 25. For high costs of these operations, RaligNAtor is sensitive while maintaining a high specificity.

To search with RNAMotif, we vary the number of allowed errors per substructure between 0 and 25 in steps of 5, constraining the total number of errors to this same number. This means that no indels are allowed, since this requires many different patterns specifying possible indels only for specific pattern positions. The results are shown in Table S11. RNAMotif is highly specific for the complete range of allowed indels, but it is not as sensitive as RaligNAtor. Notably, unlike in the search with RaligNAtor, its sensitivity only marginally increases when the number of allowed errors varies from 20 to 25, with some decrease of its specificity. Similar results can be obtained with RaligNAtor by setting $d = 0$. 

```
Function markSuffixes(iSuffix, iUnalignedPrefixLength, iLcpCheck)
1   //Mark suffixes by traversing suffix array top down
2   iSuffixDown := iSuffix + 1
3   while iSuffixDown \leq n and lcp[iSuffixDown] \geq iLcpCheck do
4       if suf[iSuffixDown] - iUnalignedPrefixLength \geq 1 then
5           vtab[suf[iSuffixDown] - iUnalignedPrefixLength] := true
6       end
7       iSuffixDown := iSuffixDown + 1
8   end
9   //Mark suffixes by traversing suffix array bottom up
10  iSuffixUp := iSuffix - 1
11  while iSuffixUp \geq 1 and lcp[iSuffixUp + 1] \geq iLcpCheck do
12     if suf[iSuffixUp] - iUnalignedPrefixLength \geq 1 then
13        vtab[suf[iSuffixUp] - iUnalignedPrefixLength] := true
14     end
15     iSuffixUp := iSuffixUp - 1
16  end
```
Table S1: Times in minutes required by algorithms ScanAlign, LScanAlign, LESAAlign, and LGSlinkAlign to search with a stem-loop pattern of length 33 in RFAM10.1. Times are influenced by the cost threshold $K$ and the number of allowed indels $d$. For a graphical representation of the measurements, see Figure S4(1).

<table>
<thead>
<tr>
<th>$K$</th>
<th>$d$</th>
<th>#matches</th>
<th>ScanAlign</th>
<th>LScanAlign</th>
<th>LESAAlign</th>
<th>LGSlinkAlign</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>0</td>
<td>201.80</td>
<td>14.69</td>
<td>0.56</td>
<td>1.63</td>
</tr>
<tr>
<td>1</td>
<td>23</td>
<td>270.43</td>
<td>43.83</td>
<td>3.62</td>
<td>2.46</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>71</td>
<td>282.99</td>
<td>67.87</td>
<td>12.39</td>
<td>6.06</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>164</td>
<td>291.66</td>
<td>124.34</td>
<td>43.19</td>
<td>19.90</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>354</td>
<td>300.93</td>
<td>203.04</td>
<td>125.74</td>
<td>43.01</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3,771</td>
<td>323.66</td>
<td>256.16</td>
<td>246.60</td>
<td>66.25</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>86,509</td>
<td>326.85</td>
<td>294.44</td>
<td>348.69</td>
<td>83.96</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1,546,439</td>
<td>339.55</td>
<td>342.66</td>
<td>459.26</td>
<td>104.08</td>
<td></td>
</tr>
</tbody>
</table>

Table S2: Times in minutes required by algorithms ScanAlign, LScanAlign, LESAAlign, and LGSlinkAlign to search with a stem-loop pattern of length 33 in RFAM10.1. Here, the cost threshold $K$ is constant and the number of allowed indels $d$ increases progressively.

<table>
<thead>
<tr>
<th>$K$</th>
<th>$d$</th>
<th>#matches</th>
<th>ScanAlign</th>
<th>LScanAlign</th>
<th>LESAAlign</th>
<th>LGSlinkAlign</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
<td>398</td>
<td>264.68</td>
<td>211.03</td>
<td>101.11</td>
<td>37.93</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>2,873</td>
<td>275.95</td>
<td>240.55</td>
<td>159.24</td>
<td>45.92</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>23,440</td>
<td>281.80</td>
<td>262.52</td>
<td>216.26</td>
<td>56.46</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>103,792</td>
<td>290.80</td>
<td>278.29</td>
<td>268.33</td>
<td>68.04</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>309,464</td>
<td>302.53</td>
<td>295.05</td>
<td>317.23</td>
<td>78.01</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>688,675</td>
<td>325.10</td>
<td>313.40</td>
<td>364.49</td>
<td>86.72</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>1,434,360</td>
<td>333.96</td>
<td>325.80</td>
<td>409.53</td>
<td>95.51</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>1,546,439</td>
<td>339.55</td>
<td>342.66</td>
<td>459.26</td>
<td>104.08</td>
</tr>
</tbody>
</table>

4 RNA family classification using Structator

Structator [3] is an ultra fast tool for RSSP matching. It is the first tool to integrate algorithms for global and local chaining of RNA pattern matches. However, it has limited support to approximate matching, lacking support of the sequence-structure edit operations allowed by RaligNAtor.

Here, we report the number of sequence members obtained by Structator when searching RFAM10.1 with the SSDs of families RF00458 and RF01736. The SSDs are shown in Figures S5 and 8 of the main document. Despite sharing the same pattern syntax with RaligNAtor, we observe the following differences and adaptations.

- **Structator** cannot search for stem-loop patterns with dangling ends. Therefore, we remove the dangling end of the RSSP ies3 belonging to the SSD of family RF00458.
- **As Structator** does not allow for edit operations, parameters cost and indels have no effect in the search. However, a number of allowed mispairings for each pattern can be specified by the user. We allow for each pattern a number of mispairings equal to the value of parameter cost.
- **Structator** has lower sensitivity compared to RaligNAtor when the latter searches with allowed costs greater than zero. For this reason, we chain the matches to the single RSSPs varying the minimum required chain length between 2 and the total number of RSSPs of each SSD.

The results are shown in Table S10. We observe that, in the search with the SSD of family RF00458, Structator cannot find all its true sequence members without increasing considerably the number of false positives. In the search with the SSD of family RF01736, only up to 4 true sequence members can be found. RaligNAtor, in contrast, finds all sequence members of both families and no false positives, as described in the main document.
Figure S4: Running times needed by the different algorithms to search with a stem-loop pattern of length 33 in RFAM10.1. In (1) the cost threshold $K$ and the number of allowed indels $d$ increase equally. In (2) $K = 7$ is constant and $d$ increases from 0 to 7. In (3) $d = 0$ is constant and $K$ increases from 0 to 7. The numbers of resulting matches are given on the x-axes in brackets.

Table S3: Times in minutes required by algorithms ScanAlign, LScanAlign, LESAAlign, and LGSlinkAlign to search with a stem-loop pattern of length 33 in RFAM10.1. Here, indels are not allowed and the cost threshold $K$ increases progressively.

<table>
<thead>
<tr>
<th>$K$</th>
<th>$d$</th>
<th>#matches</th>
<th>ScanAlign</th>
<th>LScanAlign</th>
<th>LESAAlign</th>
<th>LGSlinkAlign</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
<td>261.80</td>
<td>14.60</td>
<td>0.56</td>
<td>1.63</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>7</td>
<td>259.17</td>
<td>31.45</td>
<td>1.78</td>
<td>1.86</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>10</td>
<td>256.27</td>
<td>44.37</td>
<td>3.10</td>
<td>2.52</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>10</td>
<td>257.19</td>
<td>60.41</td>
<td>6.43</td>
<td>3.57</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>11</td>
<td>257.61</td>
<td>90.22</td>
<td>14.52</td>
<td>8.07</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>11</td>
<td>257.61</td>
<td>138.30</td>
<td>33.11</td>
<td>18.90</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>50</td>
<td>257.50</td>
<td>176.02</td>
<td>63.11</td>
<td>31.45</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>398</td>
<td>258.00</td>
<td>202.68</td>
<td>100.82</td>
<td>37.45</td>
</tr>
</tbody>
</table>

Table S4: Times in minutes required by algorithms ScanAlign, LScanAlign, LESAAlign, and LGSlinkAlign to match in RFAM10.1 the single RSSP describing the consensus secondary structure of the tRNA (Acc.: RF00005). Times are influenced by the cost threshold $K$ and the number of allowed indels $d$.
Table S5: Search times in minutes required by algorithms ScanAlign, LScanAlign, LESAAlign, and LGSlinkAlign to match in RFAM10.1 the single RSSP describing the consensus secondary structure of the tRNA (Acc.: RF00005). Here, the cost threshold $K$ is constant and the number of allowed indels $d$ increases progressively.

<table>
<thead>
<tr>
<th>$K$</th>
<th>$d$</th>
<th>#matches</th>
<th>ScanAlign</th>
<th>LScanAlign</th>
<th>LESAAlign</th>
<th>LGSlinkAlign</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0</td>
<td>10,516</td>
<td>1,536.08</td>
<td>121.18</td>
<td>17.69</td>
<td>22.82</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>30,633</td>
<td>1,576.73</td>
<td>156.50</td>
<td>35.67</td>
<td>39.87</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>49,287</td>
<td>1,657.61</td>
<td>188.79</td>
<td>58.99</td>
<td>52.98</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>64,226</td>
<td>1,703.31</td>
<td>222.36</td>
<td>86.39</td>
<td>65.94</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>74,146</td>
<td>1,754.08</td>
<td>256.78</td>
<td>119.47</td>
<td>80.55</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>77,679</td>
<td>1,808.84</td>
<td>287.49</td>
<td>156.48</td>
<td>94.03</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>77,805</td>
<td>1,830.33</td>
<td>319.32</td>
<td>198.94</td>
<td>107.63</td>
</tr>
</tbody>
</table>

Table S6: Search times in minutes required by algorithms ScanAlign, LScanAlign, LESAAlign, and LGSlinkAlign to match in RFAM10.1 the single RSSP describing the consensus secondary structure of the tRNA (Acc.: RF00005). Here, no indels are allowed and the cost threshold $K$ increases progressively.

<table>
<thead>
<tr>
<th>$K$</th>
<th>$d$</th>
<th>#matches</th>
<th>ScanAlign</th>
<th>LScanAlign</th>
<th>LESAAlign</th>
<th>LGSlinkAlign</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1,582.03</td>
<td>21.81</td>
<td>0.53</td>
<td>3.09</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>166</td>
<td>1,601.82</td>
<td>35.39</td>
<td>1.21</td>
<td>2.86</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>439</td>
<td>1,601.20</td>
<td>45.20</td>
<td>2.05</td>
<td>3.00</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1,112</td>
<td>1,601.90</td>
<td>54.83</td>
<td>2.87</td>
<td>3.72</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>2,963</td>
<td>1,606.61</td>
<td>74.29</td>
<td>4.90</td>
<td>5.71</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>6,518</td>
<td>1,601.01</td>
<td>96.93</td>
<td>9.53</td>
<td>11.57</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>10,516</td>
<td>1,601.93</td>
<td>118.26</td>
<td>17.34</td>
<td>21.87</td>
</tr>
</tbody>
</table>

Table S7: Search times in minutes used to investigate the scaling behavior of algorithms ScanAlign, LScanAlign, LESAAlign, and LGSlinkAlign on random subsets of RFAM10.1 of increasing size. See the definition of the searched RSSPs flg1, flg2, and flg3 and further details of this experiment on the main document.

<table>
<thead>
<tr>
<th>RSSP</th>
<th>ScanAlign</th>
<th>LScanAlign</th>
<th>LESAAlign</th>
<th>LGSlinkAlign</th>
</tr>
</thead>
<tbody>
<tr>
<td>flg1</td>
<td>13.13</td>
<td>12.85</td>
<td>1.02</td>
<td>2.66</td>
</tr>
<tr>
<td>flg2</td>
<td>203.67</td>
<td>191.24</td>
<td>97.46</td>
<td>45.46</td>
</tr>
<tr>
<td>flg3</td>
<td>37.42</td>
<td>40.32</td>
<td>20.61</td>
<td>34.55</td>
</tr>
</tbody>
</table>

Table S8: Times in minutes required by algorithms ScanAlign, LScanAlign, LESAAlign, and LGSlinkAlign to match the RSSPs that build the SSD for family Cripavirus internal ribosome entry site (Acc.: RF00458) in RFAM10.1.

<table>
<thead>
<tr>
<th>RSSP</th>
<th>ScanAlign</th>
<th>LScanAlign</th>
<th>LESAAlign</th>
<th>LGSlinkAlign</th>
</tr>
</thead>
<tbody>
<tr>
<td>flg1</td>
<td>288.21</td>
<td>143.23</td>
<td>74.90</td>
<td>27.03</td>
</tr>
<tr>
<td>flg2</td>
<td>310.68</td>
<td>148.17</td>
<td>50.01</td>
<td>22.00</td>
</tr>
<tr>
<td>flg3</td>
<td>156.60</td>
<td>51.74</td>
<td>8.67</td>
<td>5.83</td>
</tr>
</tbody>
</table>

Table S9: Times in minutes required by algorithms ScanAlign, LScanAlign, LESAAlign, and LGSlinkAlign to match the RSSPs that build the SSD for family flg-Rhizobiales RNA motif (Acc.: RF01736) in RFAM10.1.
Figure S5: Consensus secondary structure of family Cripavirus internal ribosome entry site (Acc.: RF00458) showing its four characteristic stem-loop substructures pt2, pt3, pt4, and pt5 and the moderately conserved strand pt1 as drawn by VARNA [4]. The secondary structure descriptor (SSD) for this family, on the right-hand side, consists of five RSSPs ires1, ires2, ires3, ires4, and ires5 describing the strand and stem-loop substructures.

Table S10: Results obtained with Structator [3] when searching with the secondary descriptors of families RF00458 and RF01736 in RFAM10.1. The first column for each family indicates the minimum required length of a chain to be considered a matching chain. #TP, #FP, and #FN stand for number of true positives, false positives, and false negatives, respectively. For additional details, see text above.

Figure S6: RNAMotif descriptor without errors for the tRNA.
**Table S11:** Results of the searches in RFAM10.1 for the tRNA (Acc.: RF00005). For the two series of searches with RaligNAtor using the operation costs above, the sequence-structure pattern shown in Figure 6 of the main document is used. For the searches with RNAMotif varying the number of allowed errors (#Errors), the descriptor shown in Figure S6 is used. These errors comprehend replacements and mispairings. #TP, #FP, and #FN stand for number of true positives, false positives, and false negatives, respectively. Sensitivity is computed as $\frac{\#TP}{\#TP + \#FN}$, specificity as $\frac{\#TN}{\#TN + \#FP}$, accuracy as $\frac{\#TP + \#TN}{\#TP + \#FP + \#FN + \#TN}$, and precision as $\frac{\#TP}{\#TP + \#FP}$. For additional details, see text above.
Figure S7: Results of ROC analyses using RaligNAtor with and without base pairing information and blastn [5] for the Rfam families shown in Table 1 of the main document. ROC curves showing RaligNAtor’s classification performance using (ignoring) base pairing information are shown in green (blue). Blast performance results are shown in red. The ROC curves are sorted by increasing level of sequence identity of the respective family, i.e. in the same order each family is listed in Table 1 of the main document. Additional ROC curves are shown in Figure S8. For details of this experiment, see corresponding description in the main document.
Figure S8: Additional ROC curves. See description of Figure S7 for details.
References


