

procedure PAIn

(*M is the set of base-pairs in RNA profile R . M' is the augmented set. *)

for all intervals (i, j) , $1 \leq i < j \leq n$, all nodes $v \in M'$

if $v \in M$

$$A[i, j, v] = \max \begin{cases} A[i+1, j-1, \text{child}(v)] + \delta(l_v, r_v, t[i], t[j]), \\ A[i, j-1, v] + \gamma(' - ', t[j]), \\ A[i+1, j, v] + \gamma(' - ', t[i]), \\ A[i+1, j, \text{child}[v]] + \gamma(l_v, t[i]) + \gamma(r_v, ' - '), \\ A[i, j-1, \text{child}[v]] + \gamma(l_v, ' - ') + \gamma(r_v, t[j]), \\ A[i, j, \text{child}[v]] + \gamma(l_v, ' - ') + \gamma(r_v, ' - '), \end{cases}$$

else if $v \in M' - M$, and v has one child

$$A[i, j, v] = \max \begin{cases} A[i, j-1, \text{child}[v]] + \gamma(r_v, t[j]), \\ A[i, j, \text{child}[v]] + \gamma(r_v, ' - '), \\ A[i, j-1, v] + \gamma(' - ', t[j]), \\ A[i+1, j, v] + \gamma(' - ', t[i]), \end{cases}$$

else if $v \in M' - M$, and v has two children

$$A[i, j, v] = \max_{i \leq k \leq j} \{A[i, k-1, \text{left_child}[v]] + A[k, j, \text{right_child}[v]]\}$$

end if

end for

FIGURE 1. An algorithm for aligning an RNA profile R with m columns against a database string t of length n . The query consensus structure M has been *Binarized* to get M' . Each node v in the tree corresponds to a base-pair $(l_v, r_v) \in M'$.

1. METHODOLOGY

1.1. The Alignment Procedure. We make the assumption that the base-pairs are non-crossing. For each base-pair $(i, j) \in M$, there is a unique (*parent*) base-pair (i', j') such that $i' < i < j < j'$, and there is no base-pair (i'', j'') such that $i < i'' < i'$, or $j < j'' < j'$. Thus the alignment can be done by recursing on the nodes of the tree. However, the tree can have high degree and not all columns of the profile participate in it. To this end we binarize the tree using the procedure given in [?]. Specifically, we add spurious nodes (base-pairs) to the tree so that every column participates as a tree node, the degree of any node is at most 3, and the number of nodes is $O(m)$, where m is the number of columns in the profile. Further, a node corresponding to a true base-pair $(i, j) \in M$ has at most one child.

Figure 1 describes a dynamic programming algorithm for aligning a sequence to an RNA profile. The RNA profile is described by a tree. Each node v in the tree either corresponds to a base-pair $(l_v, r_v) \in M'$ of the profile, where M' is the augmented list of base-pairs. The alignment of the sequence to the RNA profile is done by recursing on the tree like structure of RNA. Each node in the binarized tree either represents a base-pair/unpaired base (and has its own PSSM), or represents a branching point in a pair of parallel loops. The algorithm maintains the sequence interval being aligned and the current node in the structure tree.

```

procedure mapRetive
(*i and j are the global position in the table assuming that banding is not used. *)
if i & j are within the band of lv and rv
it = i - lv + band
jt = j - rv + band
A[it, jt, v]
else
return initialization value for (i, j, v)
end if

procedure mapSet
if i & j are within the band of lv and rv
it = i - lv + band
jt = j - rv + band
A[it, jt, v] = value
end if

```

FIGURE 2. The mapping functions make the transition to the space saving banding array. Figure 1 assumes that the array A is of size $n * n * m$ while we changed to $band * band * m$. If you are requested to reference something outside the band, the initialization value is calculated at that point, also if you are asked to set something outside the band, nothing happens.

1.2. **Banding.** Banding was used to limit the search space. The band does relies on the idea that we assume that of the location of vertex v is at position (l_v, r_v) in the original sequence, then we can safely assume that it will be close to that position in the next sequence. As a side note, we must make sure that the banding constant we choose is larger than the difference in length between the sequences. For our implementation of the band, at each vertex v in the tree we would only examine those locations for each vertex such that $l_v - band \leq i \leq l_v + band$ and $r_v - band \leq j \leq r_v + band$. The result is that any values outside of these bounds is not updated by the algorithm (but may be referenced). Once we know that certain locations in the table will not be altered after initialization we can use this intuition and only store in memory the locations that will be used. This means that we can go from A being $O(n^2 * m)$ to $O(band^2 * m)$, where $band$ is some banding constant.

Figure 2 shows the methods used to emulate the entire A array when allocating only a small portion. Notice that if the value referenced is not within the banding limit the initialization value is calculated at that time. We can see that only a small amount of overhead is created when using these methods, so the running time of the algorithm is not greatly effected.

1.3. **Multiple Alignment.** When approaching multiple alignments we used a constructive profile solution. We started with one seed sequence, then aligned one sequence to that. Once we had that original alignment, we then used that as the profile to guide the

alignment of a third sequence. Again we used this output alignment as the profile for the next alignment and continued in this fashion.

The assumption was that we only had the structure for one sequence in the set, and that the rest of the sequences needed the structure added. To that end the original alignment was between one sequence with structure and another without. The for the rest we aligned the sequence without structure to the profile which contained the structure.

What we ended up with, was two programs, one for structure-sequence alignment, and one for profile-sequence alignments, both output their results to a profile file for analysis, and both used all of the intuition and methodology discussed above.

2. RESULTS