For each suffix of a structured motif, $\mathcal{M}$, starting at position i with $1 \le i \le |\mathcal{M}|$, we keep its pos-list, $\mathcal{P}_i$, and an index list, $\mathcal{N}_i$. For each entry, say $\mathcal{P}_i[j]$, in the pos-list $\mathcal{P}_i$, the corresponding index entry $\mathcal{N}_i[j]$, points to the first entry, say f, in $\mathcal{P}_{i+1}$ that satisfies the gap range with respect to $\mathcal{P}_i[j]$, i.e., $\mathcal{P}_{i+1}[f] - \mathcal{P}_i[j] - 1 \in [l_i, u_i]$. Note that $\mathcal{N}_{|\mathcal{M}|}$ is never used. Also note that $\mathcal{P}(\mathcal{M}) = \mathcal{P}_1$. Let s be a start position for the structured motif in sequence S, and let s be the j_s-th entry in $\mathcal{P}_1$, i.e., $s = \mathcal{P}_1[j_s]$. Let F store a full position starting from s, and let $\mathcal{F}$ store the set of all full positions. Figure 2 shows the pseudo-code for recovering full positions starting from s. This recursive algorithm has four parameters: i denotes a (suffix) position in $\mathcal{M}$, j gives the j-th entry in $\mathcal{P}_i$, F denotes an intermediate full position, and $\mathcal{F}$ denotes the set of all the full occurrences. The algorithm is initially called with $i = 2$, $j = \mathcal{N}_1[j_s]$, $F = \{s\}$, and $\mathcal{F} = \emptyset$. Starting at the first index in P_i, that satisfies the gap range with respect to the last position in F, we continue to compute all such positions $j' \in [j, |P_i|]$ that satisfy the gap range (line 3). That is, we find all positions j', such that $P_i[j'] - F[i - 1] - 1 = d \in [l_i, u_i]$. For each such position j', we add it in turn to the intermediate full position, and make another recursive call (line 5), passing the first index position $N_i[j']$ in P_{i+1} that can satisfy the gap range with respect to $P_i[j']$. Thus in each call we keep following the indices from one pos-list to the next, to finally obtain a full position starting from s when we reach the last pos-list, $\mathcal{P}_{|\mathcal{M}|}$.

Note that at each suffix position i, since j only marks the first position in $\mathcal{P}_{i+1}$ that satisfies the gap constraints, we also need to consider all the subsequent positions $j' > j$ that may satisfy the corresponding gap range.

Consider the example shown in Fig. 3 to recover the full positions for $\mathcal{M}$ = CCG[0,3]TA[1,3]GAAC. Under each symbol we show two columns. The left column corresponds to the intermediate pos-lists as we proceed from right to left, whereas the right column stores the indices into the previous pos-list. For example, the middle column gives the pos-list $\mathcal{P}(TA[1,3]GAAC) = \{\mathbf{1}, \mathbf{1}, 4, \mathbf{2}, \mathbf{2}, 5, 7, \mathbf{3}, \mathbf{1}, 1\}$. For each position $x \in \mathcal{P}(TA[1,3]GAAC)$ (excluding the sequence identifiers and the cardinality), the right column records an index in $\mathcal{P}(GAAC)$ which corresponds to the first position in $\mathcal{P}(GAAC)$ that satisfies the gap range with respect to x. For example, for position $x = 5$ (at index 6), the first position in $\mathcal{P}(GAAC)$ that satisfies the gap range [1,3] is 10 (since in this case there are 3 gaps between the end of TA at position 6 and start of GAAC at position 10), and it occurs at index 6. Likewise, for each position in the current pos-list we store which positions in the previous pos-list were extended. With this indexed information, full-position recovery becomes straightforward. We begin with the start positions of the occurrences. We then keep following the indices from one pos-list to the next, until we reach the last pos-list. Since the index only marks the first position that satisfies the gap range, we still need to check if the following positions satisfy the gap range. At each stage in the full position recovery, we maintain a list of intermediate position prefixes $\mathcal{F}$ that match up to the j-th position in $\mathcal{M}$. For example, to recover the full position for $\mathcal{M}$ = CCG[0,3]TA[1,3]GAAC, considering start position 1 (with $\mathcal{F} = \{(1)\}$) in sequence 2, we follow index 6 to get position 5 in the middle pos-list, to get $\mathcal{F} = \{(1, 5)\}$. Since the next position after 5 is 7

```
   Full-Position-Recovery(i, j, F, F)
1  if (i > |M|) then
2      Add F to F;
3  foreach (j' ∈ [j, |P_i|]   c   a  (P_i[j'] − F[i − 1] − 1 = d) ∈ [l_i, u_i]) do
4      F[i] ← P_i[j'];
5      Full-Position-Recovery(i + 1, N_i[j'], F, F);
6  if ( =2) then
7      Return F;
```

Figure 2
Indexed Full Position Recovery Algorithm.