

e-CCC-Biclustering: Algorithmic and complexity details

Sara C. Madeira^{*1,2,3}, Arlindo L. Oliveira^{1,2}

¹ Knowledge Discovery and Bioinformatics (KDBIO) group, INESC-ID, Lisbon, Portugal

² Instituto Superior Técnico, Technical University of Lisbon, Lisbon, Portugal

³ University of Beira Interior, Covilhã, Portugal

Email: Sara C. Madeira* - smadeira@kdbio.inesc-id.pt; Arlindo L. Oliveira - aml@inesc-id.pt;

*Corresponding author

Abstract

This document provides supplementary material describing algorithmic and complexity details of *e*-CCC-Biclustering. For clarity we repeat here the details of the main steps of *e*-CCC-Biclustering already presented in the main manuscript. We believe that a complete description of the algorithmic details makes it easier to understand the detailed complexity analysis presented afterwards.

e-CCC-Biclustering: Algorithmic details

In this section we perform a detailed algorithmic analysis of each of the 4 steps of e-CCC-Biclustering presented in Algorithm 1:

1. Reporting all right maximal e-CCC-Biclusters: procedure `computeRightMaximalBiclusters`.
2. Deleting valid models not corresponding to left-maximal e-CCC-Biclusters: procedure `deleteNonLeftMaximalBiclusters`.
3. Deleting valid models representing the same e-CCC-Biclusters: procedure `deleteRepeatedBiclusters`.
4. Reporting all maximal e-CCC-Biclusters: procedure `reportMaximalBiclusters`.

Algorithm 1: e-CCC-Biclustering

Input : A, Σ, e, q_r, q_c

Output: Maximal e-CCC-Biclusters.

```

1  $\{S_1, \dots, S_{|R|}\} \leftarrow \text{alphabetTransformation}(A, \Sigma)$ 
2  $modelsOcc \leftarrow \{\}$  /* List of  $(m, genesOcc_m, numberOfGenesOcc_m)$  */
3 computeRightMaximalBiclusters( $\Sigma, e, q_r, q_c, \{S_1, \dots, S_{|R|}\}, modelsOcc$ )
4 deleteNonLeftMaximalBiclusters( $modelsOcc$ )
5 if  $e > 0$  then
6   deleteRepeatedBiclusters( $modelsOcc$ )
7 reportMaximalBiclusters( $modelsOcc$ )

```

1. Computing valid models corresponding to right-maximal e -CCC-Biclusters

```

Procedure computeRightMaximalBiclusters


---


  Input:  $\Sigma, e, q_r, q_c, \{S_1, \dots, S_{|R|}\}, modelsOcc$ 
  /* The value of  $modelsOcc$  is updated. */
1  $T_{right} \leftarrow \text{constructGeneralizedSuffixTree}(\{S_1, \dots, S_{|R|}\})$ 
2  $\text{addNumberOfLeaves}(T_{right})$  /* Adds  $L(v)$  to each node  $v$  in  $T_{right}$ . */
3 if  $e \neq 0$  then
4    $\text{addColorArray}(T_{right})$ 
   /* Adds  $colors_v$  to every node  $v$  in  $T_{right}$ :  $colors_v[i] = 1$ , if there is a leaf in the
   subtree rooted at  $v$  that is a suffix of  $S_i$ ;  $colors_v[i] = 0$ , otherwise. */
5  $m \leftarrow ""$  /* model  $m$  is a string  $[m[1] \dots m[length_m]]$  */
6  $length_m \leftarrow 0$ 
7  $father_m \leftarrow ""$  /*  $father_m$  is a string  $[m[1] \dots m[length_m - 1]]$  */
8  $numberOfGenesOcc_{father_m} \leftarrow 0$ 
9  $Occ_m \leftarrow \{\}$  /* List of node-occurrences  $(v, v_{err}, p)$  */
10  $\text{addNodeOccurrence}(Occ_m, (\text{root}(T_{right}), 0, 0))$ 
11  $Ext_m \leftarrow \{\}$  /*  $Ext_m$  is the set of possible symbols  $\alpha$  to extend the model  $m$ . */
12 if  $e = 0$  then
13   forall edges  $E(v_i)$  leaving from node  $\text{root}(T_{right})$  to a node  $v_i$  do
14     if  $\text{label}(E(v_i))[1]$  is not a string terminator then
15        $\text{addSymbol}(Ext_m, \text{label}(E(v_i))[1])$ 
16 else
17   forall symbols in  $\Sigma'$  do
   /*  $\Sigma'$  must be in lexicographic order. */
18      $\text{addSymbol}(Ext_m, \Sigma'[i])$ 
19  $length_m \leftarrow 0$ 
20  $\text{spellModels}(\Sigma, e, q_r, q_c, modelsOcc, T_{right}, m, length_m, Occ_m, Ext_m, father_m, numberOfGenesOcc_{father_m})$ 

```

Below we present the algorithmic details of the recursive procedure `spellModels`, which is an adaption of SPELLER [1] able to identify all valid models corresponding to right-maximal e -CCC-Biclusters. These e -CCC-Biclusters are identified by right-maximal valid models (patterns that cannot be extended without losing occurrences) and their corresponding node-occurrences (patterns starting and finish at the same time points in a set of rows in the time series expression matrix).

```

Procedure spellModels
  /* Called recursively. Stores right-maximal e-CCC-Biclusters in modelsOcc. */
  Input :  $\Sigma, e, q_r, q_c, modelsOcc, T_{right}, m, length_m, Occ_m, Ext_m, father_m,$ 
            $numberOfGenesOcc_{father_m}$ 
  /* The value of modelsOcc is updated. */
1 keepModel( $q_r, q_c, modelsOcc, T_{right}, m, length_m, Occ_m, father_m, numberOfGenesOcc_{father_m}$ )
2 if  $length_m \leq |C|$  then
   /*  $|C|$  is the length of the longest model */
3   forall symbols  $\alpha$  in  $Ext_m$  do
4     if  $\alpha$  is not a string terminator then
5        $maxGenes \leftarrow 0$  /* Sum of  $L(v)$  for all node-occurrences  $(v, v_{err}, p)$  in  $Occ_{m\alpha}$  */
6        $minGenes \leftarrow \infty$  /* Minimum  $L(v)$  in all node-occurrences  $(v, v_{err}, p)$  in  $Occ_{m\alpha}$  */
7        $Colors_{m\alpha} \leftarrow \{\}$ 
8       if  $e > 0$  then
9          $Colors_{m\alpha}[i] \leftarrow 0, 1 \leq i \leq |R|$ 
          /*  $colors_{m\alpha}[i] = 1$ , if there is a node-occurrence of  $m$  in  $S_i$ ; */
          /*  $colors_{m\alpha}[i] = 0$ , otherwise */
10         $Ext_{m\alpha} \leftarrow \{\}$ 
11         $Occ_{m\alpha} \leftarrow \{\}$ 
12        forall node-occurrences  $(v, v_{err}, p)$  in  $Occ_m$  do
          /* If  $p = 0$  we are at node  $v$ . Otherwise, we are at edge  $E(v)$  between
             nodes  $father(v)$  and  $v$  at point  $p > 0$ . */
13        if  $p = 0$  then
14          extendFromNodeWithoutErrors( $\Sigma, e, T_{right}, (v, v_{err}, p), m, \alpha, Occ_{m\alpha}, Colors_{m\alpha},$ 
             $Ext_{m\alpha}, maxGenes, minGenes$ )
15          if  $(v_{err} < e)$  then
16            extendFromNodeWithErrors( $\Sigma, e, T_{right}, (v, v_{err}, p), m, \alpha, Occ_{m\alpha}, Colors_{m\alpha},$ 
             $Ext_{m\alpha}, maxGenes, minGenes$ )
17          else
18            extendFromEdgeWithoutErrors( $T_{right}, \Sigma, e, (v, v_{err}, p), m, \alpha, m, Occ_{m\alpha},$ 
             $Colors_{m\alpha}, Ext_{m\alpha}, maxGenes, minGenes$ )
19            if  $x_{err} < e$  then
20              extendFromEdgeWithErrors( $\Sigma, e, T_{right}, (v, v_{err}, p), m, \alpha, Occ_{m\alpha}, Colors_{m\alpha},$ 
             $Ext_{m\alpha}, maxGenes, minGenes$ )
21          if modelHasQuorum( $maxGenes, minGenes, Colors_{m\alpha}, q_r$ ) then
22            spellModels( $\Sigma, e, q_r, q_c, modelsOcc, T_{right}, m\alpha, length_m + 1, Occ_{m\alpha}, Ext_{m\alpha},$ 
             $father_{m\alpha}, numberOfGenesOcc_m$ )

```

Next we present the algorithmic details of the procedures and functions used in the procedure `spellModels` described above. We use the following order:

1. Procedure `keepModel` [called in `spellModels`]
2. Procedure `checkRightMaximality` [called in `keepModel()`]
3. Procedure `extendFromNodeWithoutErrors` [called in `spellModels`]
4. Procedure `extendFromNodeWithErrors` [called in `spellModels`]
5. Procedure `extendFromEdgeWithoutErrors` [called in `spellModels`]

6. Procedure `extendFromEdgeWithoutErrors` [called in `spellModels`]
7. Function `modelHasQuorum` [called in `spellModels`]
8. Procedure `extendModel` [called in `extendFromNodeWithoutErrors()`,
`extendFromNodeWithErrors()`, `extendFromEdgeWithoutErrors` and
`extendFromEdgeWithErrors()`]

Procedure keepModel

```

Input:  $q_r, q_c, modelsOcc, T_{right}, length_m, Occ_m, father_m, numberOfGenesOcc_{father_m}$ 
/* The value of  $modelsOcc$  is updated. */
/* The initial model corresponding to the root should never be kept. */
1 if  $length_m > 0$  then
    /* Check columns quorum. */
2     if  $length_m \geq q_c$  then
3          $genesOcc_m \leftarrow computeGenesInNodeOccurrences(Occ_m, T_{right})$  /* bit vector */
4          $numberOfGenesOcc_m \leftarrow computeNumberOfGenesInNodeOccurrences(genesOcc_m)$ 
        /* Check rows quorum. */
5         if  $numberOfGenesOcc_m \geq q_c$  then
6              $addModelAndOccurrences(modelsOcc, (m, genesOcc_m, numberOfGenesOcc_m))$ 
            /* Check if  $father_m$  satisfied the columns quorum. If so it was kept. */
7             if  $length_{father_m} \geq q_c$  then
                /* Throw away  $father_m$  if it does not correspond to a right-maximal
                e-CCC-Bicluster. */
8              $checkRightMaximality(modelsOcc, numberOfGenesOcc_m, father_m,$ 
                 $numberOfGenesOcc_{father_m})$ 

```

Procedure checkRightMaximality

```

Input:  $modelsOcc, numberOfGenesOcc_m, father_m, numberOfGenesOcc_{father_m}$ 
/* The value of  $modelsOcc$  is updated. */
/* Check if  $father_m$  was the root. If so it was not kept. */
1 if  $length_{father_m} \neq 0$  then
2     if  $numberOfGenesOcc_m = numberOfGenesOcc_{father_m}$  then
3          $deleteModelAndOccurrences(modelsOcc, father_m)$ 

```

Procedure extendFromNodeWithoutErrors

```

Input:  $\Sigma, e, m, (v, v_{err}, 0), \alpha, Occ_{m\alpha}, Colors_{m\alpha}, Ext_{m\alpha}, maxGenes, minGenes$ 
/* The values of  $Occ_{m\alpha}, Colors_{m\alpha}, Ext_{m\alpha}, maxGenes, minGenes$  are updated. */
1 forall edges  $E(v')$  leaving node  $v$  to a node  $v'$  do
2     if  $label(E(v'))[1] = \alpha$  then
        /*  $m$  can be extended with  $\alpha$  without errors */
3          $extendModel(\Sigma, e, m, (v, v_{err}, 0), \alpha, Occ_{m\alpha}, Colors_{m\alpha}, Ext_{m\alpha}, maxGenes, minGenes, E(v'), 0)$ 

```

Procedure extendFromNodeWithErrors

```

Input:  $\Sigma, e, T_{right}, (v, v_{err}, 0), \alpha, Occ_{m\alpha}, Colors_{m\alpha}, Ext_{m\alpha}, maxGenes, minGenes$ 
/* The values of  $Occ_{m\alpha}, Colors_{m\alpha}, Ext_{m\alpha}, maxGenes, minGenes$  are updated. */
1 forall edges  $E(v')$  leaving node  $v$  to node  $v'$  do
2   if  $label(E(v'))[1] \neq \alpha$  then
3     if  $label(E(v'))[1]$  is not a string terminator then
4       if  $v = \text{root}(T_{Right})$  then
5         /* Test if first symbol in  $label(E(v'))$  has column number equal to that of  $\alpha$ .  

6            $C(label(E(v'))[1])$  is the column number of symbol  $label(E(v'))[1]$  */
7         if  $C(label(E(v'))[1]) = C(\alpha)$  then
8           extendModel( $\Sigma, e, T_{right}, (v, v_{err}, 0), \alpha, Occ_{m\alpha}, Colors_{m\alpha}, Ext_{m\alpha}, maxGenes,$ 
9              $minGenes, E(v'), 1$ )
10        else
11          extendModel( $\Sigma, e, T_{right}, (v, v_{err}, 0), \alpha, Occ_{m\alpha}, Colors_{m\alpha}, Ext_{m\alpha}, maxGenes, minGenes,$ 
12             $E(v'), 1$ )

```

Procedure extendFromEdgeWithoutErrors

```

Input:  $\Sigma, e, T_{right}, (v, v_{err}, p), \alpha, Occ_{m\alpha}, Colors_{m\alpha}, Ext_{m\alpha}, maxGenes, minGenes$ 
/* The values of  $Occ_{m\alpha}, Colors_{m\alpha}, Ext_{m\alpha}, maxGenes, minGenes$  are updated. */
1 if  $label(E(v))[p+1] = \alpha$  then
2   extendModel( $\Sigma, e, T_{right}, (v, v_{err}, p), \alpha, Occ_{m\alpha}, Colors_{m\alpha}, Ext_{m\alpha}, maxGenes, minGenes, E(v), 0$ )

```

Procedure extendFromEdgeWithErrors

```

Input:  $\Sigma, e, T_{right}, (v, v_{err}, p), \alpha, Occ_{m\alpha}, Colors_{m\alpha}, Ext_{m\alpha}, maxGenes, minGenes$ 
/* The values of  $Occ_{m\alpha}, Colors_{m\alpha}, Ext_{m\alpha}, maxGenes, minGenes$  are updated. */
1 if  $label(E(v))[p+1] \neq \alpha$  then
2   if  $label(E(v))[1]$  is not a string terminator then
3     extendModel( $\Sigma, e, T_{right}, (v, v_{err}, p), \alpha, Occ_{m\alpha}, Colors_{m\alpha}, Ext_{m\alpha}, maxGenes, minGenes, E(v), 1$ )

```

Function modelHasQuorum

```

Input :  $maxGenes, minGenes, Colors_m, q_r$ 
Output: True or false
1 if  $maxGenes < q_r$  then
2   return false
3 else
4   if  $e = 0$  then
5     return true
6   else
7     if  $minGenes \geq q_r$  then
8       return true
9     else
10      if  $\text{cardinality}(Colors_{m\alpha}) \geq q_r$  then
11        return true
12      else
13        return false

```

Procedure extendModel

```

Input:  $\Sigma, e, T_{right}, (v, v_{err}, p), \alpha, Occ_{m\alpha}, Colors_{m\alpha}, Ext_{m\alpha}, maxGenes, minGenes, E(x), extension_{err}$ 
/* The values of  $Occ_{m\alpha}, Colors_{m\alpha}, Ext_{m\alpha}, maxGenes, minGenes$  are updated. */
1 if  $p > 0$  then
    /* If  $p = 0$ : we at node  $v$  descending edge  $E(x)$ , where  $x$  is the node reached by descending
    edge  $E(x)$  and thus  $father_x = v$ . */
    /* If  $p > 0$ : we are at point  $p$  descending edge  $E(x)$ . In this case,  $E(x)$  is the edge from
     $father_v$  to  $v$  and thus  $x = v$ . */
2    $x = v$ 
3 if  $|label(E(x))| = p + 1$  then
    /* Extension of  $m$  with  $\alpha$  will reach node  $x$  */
    /* Update  $Occ_{m\alpha}$  */
4   addNodeOccurrence( $Occ_{m\alpha}, (x, v_{err} + extension_{err}, 0)$ )
    /* Update  $maxGenes$  and  $minGenes$  */
5    $maxGenes \leftarrow maxGenes + L(x)$ 
    /* When  $e = 0$ ,  $maxGenes = minGenes$  (the number of node-occurrences of  $m\alpha = 1$ ) */
6   if  $L(x) < minGenes$  then
7      $minGenes \leftarrow L(x)$ 
    /* Update  $Colors_{m\alpha}$  */
8   if  $e > 0$  then
9      $Colors_{m\alpha} \leftarrow Colors_{m\alpha} \vee Colors_x$ 
    /* Update  $Ext_{m\alpha}$  */
10  if  $v_{err} = e$  then
    /* Error limit reached */
11    forall edges  $E(x')$  leaving node  $x$  to a node  $x'$  do
12      if  $label(E(x'))[1]$  is not a string terminator then
13        addSymbol( $Ext_{m\alpha}, label(E(x'))[1]$ )
14    else
    /* Errors are still allowed. */
    /*  $C(\alpha)$  is the column number of symbol  $\alpha$ . */
15    forall symbols  $s$  in  $\Sigma_{C(\alpha)+1} = \{f(a, C(\alpha) + 1), \forall a \in \Sigma\}$  do
16      addSymbol( $Ext_{m\alpha}, s$ )
17 else
    /* If  $p = 0$ : we will stay at edge  $E(x)$  between nodes  $v$  and  $x$  at point  $p = 1$ . */
    /* If  $p > 0$ : we will stay at edge  $E(v)$  between  $father(v)$  and  $v$  at point  $p + 1$ . */
    /* Update  $Occ_{m\alpha}$  */
18  addNodeOccurrence( $Occ_{m\alpha}, (x, v_{err} + extension_{err}, p + 1)$ )
    /* Update  $maxGenes$  and  $minGenes$  */
19   $maxGenes \leftarrow maxGenes + L(x)$ 
20  if  $L(x) < minGenes$  then
21     $minGenes \leftarrow L(x)$ 
    /* Update  $Colors_{m\alpha}$  */
22  if  $e > 0$  then
23     $Colors_{m\alpha} \leftarrow Colors_{m\alpha} \vee Colors_x$ 
24  if  $x_{err} = e$  then
    /* Error limit reached. */
25    if  $label(E(x))[p + 2]$  is not a string terminator then
26      addSymbol( $Ext_{m\alpha}, label(E(x))[p + 2]$ )
27  else
    /* Errors are still allowed. */
    /*  $C(\alpha)$  is the column number of symbol  $\alpha$ . */
28  if  $C(\alpha) + 1 \leq |C|$  then
    /* Extension will not be a string terminator. */
29    forall symbols  $s$  in  $\Sigma'_{C(\alpha)+1} = \{f(a, C(\alpha) + 1), \forall a \in \Sigma\}$  do
30      addSymbol( $Ext_{m\alpha}, s$ )

```

2. Deleting valid models not corresponding to left-maximal e -CCC-Biclusters

```

Procedure deleteNonLeftMaximalBiclusters


---


  Input:  $modelsOcc$ 
  /* The value of  $modelsOcc$  is updated. */
1  $T_{left} \leftarrow createTrie()$ 
  /* Array which will store references to nodes in  $T_{left}$  */
2  $R_{nodes} \leftarrow \{\}$ 
3 foreach  $model$  and occurrences  $(m, genesOcc_m, numberOfGenesOcc_m)$  in  $modelsOcc$  do
4    $m_r \leftarrow ReverseModel(m)$ 
5    $nodeRepresentingModel \leftarrow addReverseModelToTrie(T_{left}, m_r)$ 
   /* Each node in  $T_{left}$  stores two integers: 1) the number of genes in the model it
   represents,  $genes_v$  (0 if it does not represent the end of a model); and 2) the
   maximum number of genes in the subtree rooted at  $v$ ,  $maxGenes_{subtree_v}$  (computed
   later). Both these values are initialized with 0. */
6    $addNumberOfGenes(nodeRepresentingModel, numberOfGenesOcc_m)$ 
7    $addReferenceToNode(R_{nodes}, nodeRepresentingModel)$ 
8 forall nodes  $v$  in  $T_{left}$  do
   /* Performed using a depth-first search (dfs) */
9   if  $genes_v > 0$  then
   /* Node  $v$  represents a model and is potentially left-maximal. */
10    Mark  $v$  as "left-maximal"
11   else
12    Mark  $v$  as "non left-maximal"
13   Compute the maximum number of genes in the subtree rooted at  $v$ 
14 foreach node  $v$  in  $T_{left}$  do
   /* Performed using a depth-first search (dfs) */
15   if  $genes_v > 0$  and  $genes_v = maxGenes_{subtree_v}$  then
16    Mark  $v$  as "non left-maximal"
17  $p_{modelsOcc} \leftarrow 0$ 
18 foreach  $model$  and occurrences  $(m, genesOcc_m, numberOfGenesOcc_m)$  in  $modelsOcc$  do
19   if  $R_{nodes}[p_{modelsOcc}]$  is marked as "non-left maximal" then
20      $deleteModelAndOccurrences(modelsOcc, m)$ 
21    $p_{modelsOcc} \leftarrow p_{modelsOcc} + 1$ 

```

3. Deleting valid models representing the same *e*-CCC-Biclusters

```

Procedure deleteRepeatedBiclusters


---


  Input: modelsOcc
  /* The value of modelsOcc is updated. */
1  $H \leftarrow \text{createHashTable}()$ 
2 foreach model and occurrences ( $m$ ,  $\text{genesOcc}_m$ ,  $\text{numberOfGenesOcc}_m$ ) in modelsOcc do
3    $\text{firstColumn}_m = C(m[1])$ 
4    $\text{lastColumn}_m = C(m[\text{length}_m])$ 
5    $\text{key} \leftarrow \text{createKey}(\text{firstColumn}, \text{lastColumn}, \text{genesOcc}_m)$ 
6    $\text{value} \leftarrow (\text{firstColumn}, \text{lastColumn}, \text{genesOcc}_m)$ 
7   if  $\text{containsKey}(H, \text{key})$  then
8      $\text{value}_{\text{key}} \leftarrow \text{getValue}(H, \text{key})$ 
9     if  $\text{value} = \text{value}_{\text{key}}$  then
10      /*  $H$  already has a value representing the same CCC-Bicluster */
11       $\text{deleteModelAndOccurrences}(\text{modelsOcc}, m)$ 
12    else
13       $\text{insertKeyValue}(\text{key}, \text{value})$ 
14    else
15       $\text{insertKeyValue}(\text{key}, \text{value})$ 

```

4. Reporting all maximal *e*-CCC-Biclusters

```

Procedure reportMaximalBiclusters


---


  Input: modelsOcc
1 foreach model and occurrences ( $m$ ,  $\text{genesOcc}_m$ ,  $\text{numberOfGenesOcc}_m$ ) in modelsOcc do
2    $\text{firstColumn}_m = C(m[1])$ 
3    $\text{lastColumn}_m = C(m[\text{length}_m])$ 
4    $\text{print}(m, \text{firstColumn}_m, \text{lastColumn}_m, \text{genesOcc}_m)$ 

```

e-CCC-Biclustering: Detailed complexity analysis

In this section we perform a detailed complexity analysis of each of the 4 steps of *e*-CCC-Biclustering presented in Algorithm 1:

1. Reporting all right maximal *e*-CCC-Biclusters: procedure `computeRightMaximalBiclusters`.
2. Deleting valid models not corresponding to left-maximal *e*-CCC-Biclusters: procedure `deleteNonLeftMaximalBiclusters`.
3. Deleting valid models representing the same *e*-CCC-Biclusters: procedure `deleteRepeatedBiclusters`.
4. Reporting all maximal *e*-CCC-Biclusters: procedure `reportMaximalBiclusters`.

Given an $|R|$ by $|C|$ discretized gene expression matrix, we show that, the asymptotic complexity of the algorithm is $O(\max(|R|^2|C|^{1+e}|\Sigma|^e, |R||C|^{2+e}|\Sigma|^e))$, where e is the maximum number of allowed errors allowed per gene in the expression pattern defining each *e*-CCC-Bicluster. However, in most cases of interest $|R| \gg |C|$ and the complexity becomes $O(|R|^2|C|^{1+e}|\Sigma|^e)$. When $e = 0$, CCC-Biclustering [2, 3] can be used to obtain $O(|R||C|)$.

Procedure `computeRightMaximalBiclusters`

Given a discretized matrix A with $|R|$ rows and $|C|$ columns the alphabet transformation performed using the procedure `alphabetTransformation` returns the set of strings $S_i = \{S_1, \dots, S_{|R|}\}$ (each with length $|C|$) needed to compute the generalized suffix tree T_{right} in $O(|R||C|)$ time.

The generalized suffix tree T_{right} (for the set of $|R|$ strings S_i with length $|C|$) can be constructed in $O(|R||C|)$ using Ukkonen's algorithm [4] and standard data structures used in suffix trees since the increase in the alphabet size from $|\Sigma|$ to $|C||\Sigma|$ due to the alphabet transformation does not affect the $O(|R||C|)$ construction and manipulation of T_{right} [2, 3].

Adding the number of leaves, $L(v)$, to each node v in T_{right} is performed by a depth-first search *dfs*, which is linear on the number of nodes in T_{right} and therefore takes $O(|R||C|)$ time.

Adding the color array, a bitset of length $|R|$ needed only when $e > 0$, to each of the $O(|R||C|)$ nodes in T_{right} takes $O(|R|^2|C|)$ time.

All the remaining operations before the call to procedure `spellModels` take constant time, except the initialization of Ext_m , which is $O(|C||\Sigma|)$.

Consider now the complexity of the procedure `spellModels` when $e > 0$.

The procedure `keepModel` is $O(|R|^2|C|)$. The function `computeGenesInNodeOccurrences` takes $O(|R|^2|C|)$ since the number of node-occurrences in Occ_m is bounded by the maximum number of nodes in T_{right} , $O(|R||C|)$, and the manipulation of the color arrays takes $O(|R|)$. Given the bit vector with the genes, the function `computeNumberOfGenesInNodeOccurrences` is $O(|R|)$: in order to compute the genes in the node-occurrences of a model m (returned as a bit vector, $genesOcc_m$), the procedure `computeGenesInNodeOccurrences` performs a bitwise or between the bit vectors $colors_v$ of all node-occurrences (v, v_{err}, p) of m in case $e > 0$, and uses the value of $L(v)$ when $e = 0$ (in this case Occ_m has only one node-occurrence). The function `computeNumberOfGenesInNodeOccurrences` is then used to compute the number of genes. All the remaining operations, including the procedure `checkRightMaximality` takes constant time.

Since the number of valid models is $O(|R||C||N(e, S_i)|)$, where $|N(e, S_i)| \leq |C|^e|\Sigma|^e$, since $|S_i| = |C|$, the manipulation of the color array takes $O(|R|)$ time per model and all other operations take constant time, `spellModels` is $O(|R|^2|C|^{1+e}|\Sigma|^e)$, when $e > 0$.

Lets now look at the case where $e = 0$. In this case, `computeGenesInNodeOccurrences` also takes $O(|R|)$. Although, each model has now only one node-occurrence, the *dfs* needed to get the genes and the creation of the bit vector to be returned are both $O(|R|)$. The procedure `computeNumberOfGenesInNodeOccurrences` is also $O(|R|)$, `checkRightMaximality` takes also constant time, the color array does not exist and $|N(e, S_i)| = 1$. As such, when $e = 0$, `spellModels` is $O(|R|^2|C|)$.

In this context, the complexity of `computeRightMaximalBiclusters` is bounded by the complexity of `spellModels`, and is thus $O(|R|^2|C|^{1+e}|\Sigma|^e)$. Note that, when $e = 0$ the linear time complexity of $O(|R||C|)$ achieved by the CCC-Biclustering algorithm [2,3] is better than the $O(|R|^2|C|)$ bound obtained with e -CCC-Biclustering.

Procedure `deleteNonLeftMaximalBiclusters`

Since the number of models in $modelsOcc$ is $O(|R||C|^{1+e}|\Sigma|^e)$ and the size of the models is $O(|C|)$, the trie T_{left} can be constructed and manipulated in $O(|R||C|^{2+e}|\Sigma|^e)$.

Reversing the model m and inserting the reversed model m_r in the trie T_{left} , performed using the procedures `reverseModel` and `addReverseModelToTrie` take $O(|C|)$ time each. Storing the number of genes in the node v in T_{left} identifying the last symbol in m_r and storing the reference to this node in the array R_{nodes} take constant time. This enables the construction of T_{left} with the information about the number of genes where m occurs in $O(|R||C|^{2+e}|\Sigma|^e)$.

Since the number of nodes in T_{left} is $O(|R||C|^{2+e}|\Sigma|^e)$ each node v will be initially marked as either “left-maximal” or “non left-maximal” and the maximum number of genes stored in its subtree, $maxGenes_{subtree_v}$,

can be computed in $O(|R||C|^{2+e}|\Sigma|^e)$ using a *dfs*.

The remaining nodes v representing “non left-maximal” models can then be marked using another *dfs* performed again in $O(|R||C|^{2+e}|\Sigma|^e)$ where a simple constant time comparison is performed between the values of $genes_v$ and $maxGenes_{subtree_v}$ stored in each node identifying a model ($genes_v > 0$).

Finally, for each of the $O(|R||C|^{1+e}|\Sigma|^e)$ models stored in *modelsOcc* we just have to verify if its corresponding node v in T_{left} is marked as “non left maximal”. This verification, performed using the reference to node v the stored in R_{nodes} at position $p_{modelsOcc}$, and the potential removal of the model from *modelsOcc* are both performed in constant time.

In this context, the procedure `deleteNonLeftMaximalBiclusters` has a worst case complexity of $O(|R||C|^{2+e}|\Sigma|^e)$.

Procedure `deleteRepeatedBiclusters`

The complexity of `deleteRepeatedBiclusters`, depends only on the number of valid models (assuming the hashtable accesses work in essentially constant time), $O(|R||C|^{1+e}|\Sigma|^e)$, and the time needed to compute the hash function, $O(|R|)$, leading to a worst case complexity of $O(|R|^2|C|^{1+e}|\Sigma|^e)$.

Procedure `reportMaximalBiclusters`

Since the number of genes in $genesOcc_m$ is $O(|R|)$ and computing the first and last column of m takes constant time, the reporting procedure `reportMaximalBiclusters` is $O(|R|^2|C|^{1+e}|\Sigma|^e)$.

References

1. Sagot MF: **Spelling approximate repeated or common motifs using a suffix tree**. In *Proc. of Latin'98*, Springer Verlag, LNCS 1380 1998:111–127.
2. Madeira SC, Oliveira AL: **A Linear Time Biclustering Algorithm for Time Series Gene Expression Data**. In *Proc. of 5th Workshop on Algorithms in Bioinformatics*, Springer Verlag, LNCS/LNBI 3692 2005:39–52.
3. Madeira SC, Teixeira MC, Sá-Correia I, Oliveira AL: **Identification of Regulatory Modules in Time Series Gene Expression Data using a Linear Time Biclustering Algorithm**. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 21 Mar 2008. *IEEE Computer Society Digital Library*. *IEEE Computer Society*, 24 March 2008 [<http://doi.ieeecomputersociety.org/10.1109/TCBB.2008.34>].
4. Ukkonen E: **On-line construction of suffix trees**. *Algorithmica* 1995, 14:249–260.