Efficient Space and Time Multicomparison of Genomes *

M. Huerta and X. Messeguer

Dep. LSI, Universitat Politècnica de Catalunya, Barcelona Mòdul C5-C6, C/Jordi Girona, 1-3, E-Barcelona 08034, Spain. E-mail: peypoch@lsi.upc.es

Abstract. The comparison of genomes is based on substrings, called MUMs, that appear only once in each genome and are maximal. The set of MUMs determines the skeleton from which a global comparison can be established (see Fig. 1(a)).
The search of MUMs is handled with Suffix-trees which are those trees that store all suffixes of a given sequence. The searching algorithm builds a generalized Suffix-tree for each pair and finds the MUMs between them. In this process, most of the time (several minutes) is spent in the construction of trees. We introduce, in this paper, a slight modification of Suffix-trees, denoted as TSuffix-trees, that allow search of MUMs of many sequences. We build a tree of only one genome and we perform the search without making a bottom-up traversal. These both facts imply great savings in time and space as compared to existing programs. We remark that time is linear with respect to the length of all genomes but space is linear with respect to the length of only the smallest genome.

Keywords: MUM, Pattern matching, On-line, Suffix trees.

1 Introduction

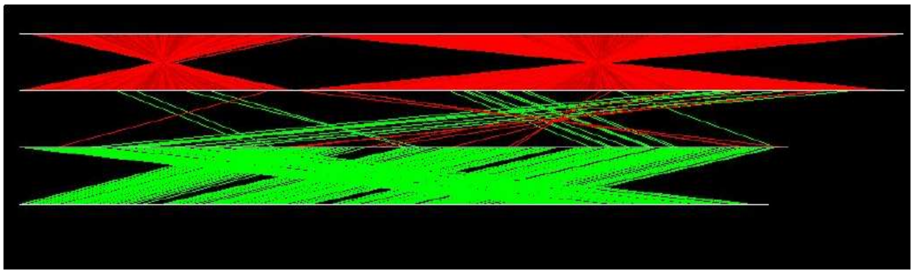
The comparison of genomes, nowadays, is very important for researchers. For this reason, during the last two years, some new computer programs have been designed to handle this task. It must be pointed out that, few years ago, there were only very few number of sequenced genomes and therefore most of the comparisons were made only between sequences of thousands of bases. Nowadays, there are many genomes available in databases having millions of bps, such as the human, fly or mouse genomes, and this size implies an increase of about 1000 times in the length of sequences. The quadratic cost of the dynamic programming used to align sequences and this 1000 times increase of the length of sequences imply that the computation time has grown 10^6 times and therefore earlier methods are unapplicable in practice and new computer tools must be designed. Moreover, as the comparision of genomes has a strong biological sense when they are very similar, then it is a good assumption that many portions of the genomes are identical. Thus the power of the dynamic programming applied to whole genomes is not necessary and the comparison of genomes with less costly techniques is possible.

The most fruitful idea [DKF+99] proposed till now is the detection of the substrings shared by genomes because they form the skeletton of the comparison. As we need to be sure that these substrings have a biological significance, they are forced to be *Maximal-Unique-Matchings*[DKF+99], in short MUMs, defined by the following strong conditions:

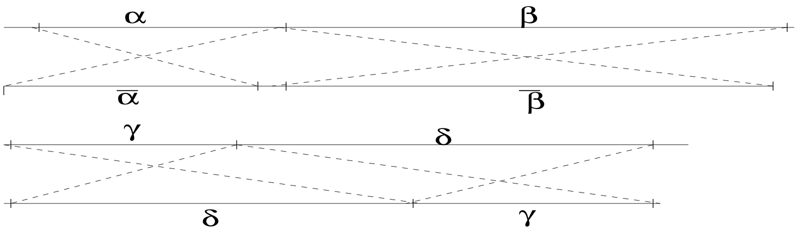
Definition 1. Given a set of sequences, a string is a Maximal-Unique-Matching of this set, if and only if:
(i) It is a substring of each sequence, or a "matching".
(ii) It appears only once in each sequence, hence it's "unique".
(iii) It is not a substring of another unique match, so it's "maximal". Note that it cannot be extended without incurring a mismatch.

For instance, given the sequences *ababaabs* and *aabaat*, there are only two MUMs: *abaa* and *aab*. Figure 1(a) pictures the comparison between the following four bacteria: *Chlamydophila pneumoniae AR39* (124742 bp), *Chlamydia pneumoniae* (1247805 bp), *Chlamydia muridarum* (1084689bp) and *Chlamydia trachomatis* (1057413bp). Horizontal lines represent genomes and lines connecting two genomes denote

* Partially supported by project DGES PB98-0926 (AEDRI; Algorithms and Data Structures for Information Retrieval)



(a) Image of the comparison between the genomes of four bacteria.



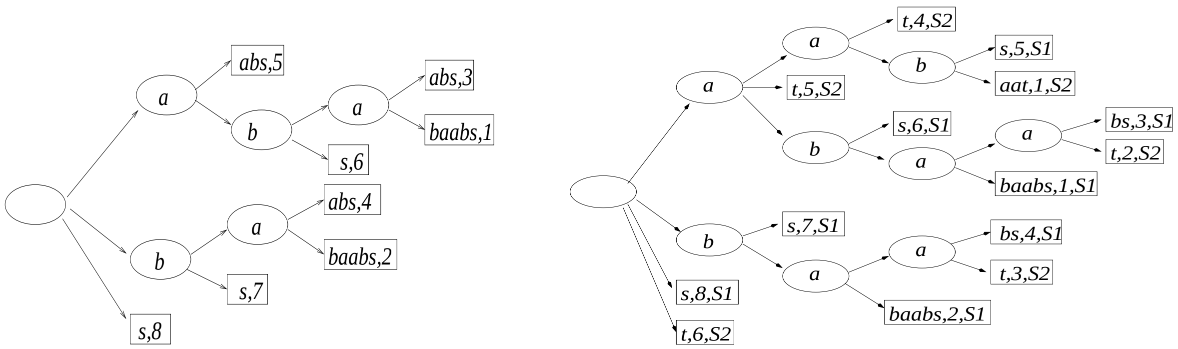
(b) Skeleton of the pairwise alignment between the genomes of four bacteria

Fig. 1. Comparison of genomes of four bacteria. This picture has been made with the program MALGEN located at the web site "http://www.lsi.upc.es/∼alggen".

MUMs(direct or inverse identity between segments of length 40 bp at least). Note that the complete set of lines suggests how genomes can be aligned. For instance, in figure 1(b) the segments $\overline{\alpha}$ and $\overline{\beta}$ are the inverse of α and β respectively, then the second genome is a transposition of the first one. The same relation holds between the last two genomes.

Until now, the search of MUMs [DKF$^+$99] has been made using Suffix-trees [Gus97]. The Suffix-tree of a sequence contains all the suffixes of the sequence (see Fig 2(a)). If two genomes have to be compared then a generalized Suffix-tree is built (see Fig 2(b)) and those paths starting at the root and ending in nodes with two leaves, one of each sequence, are searched (they are the MUMs), and so on for more genomes. This process consumes a linear time with respect to the length of genomes but has a great drawback: the time and space spent in the construction of the Suffix-tree is so high that it represents 95% of the global time to search MUMs.

Software in [mum99,DKF$^+$99] compares two genomes following these ideas. As most of the computing time was devoted to the Suffix-tree construction, the recently appeared second version [mum02,DPCS02] only constructs Suffix-tree for the first genome, then it has reduced the time to almost half. But its search finds unique-matchings strings only respect to the first genome. The paper [HKO02] improves the search of MUMs between two genomes by using Suffix-arrays during the bottom-up traversal algorithm. Recently, in [HKO02] a software has been proposed for the multialignment of genomes whose input is the generalized Suffix-tree for all genomes and, using Suffix-arrays[HKO02], it makes an efficient bottom-up search, but in this case it finds common maximal-matching strings (without the uniquess property) and they do not take into account the time spent for the Suffix-tree construction because the software takes it as the input of the program.



(a) This tree represents the suffixes of the string *ababaabs*. For instance, the suffix *baabs*, that starts in position 4, is spelt out by the path *ba* followed by leaf $[abs, 4]$.

(b) Generalized Suffix-tree of the strings *ababaabs* and *aabaat*. Note that in this case the leaves contain the number of the sequence to which they belong.

Fig. 2. Suffix-trees

Now it is clear that the main drawback of these programs is the time and space spent in Suffix-trees. But there is another drawback too, the bottom-up search. For this reason the programs that need to do it should improve this process using Suffix-arrays.

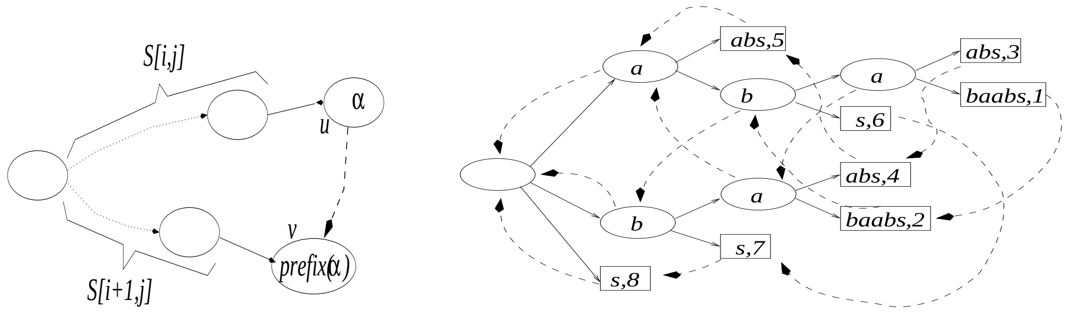
In this paper we propose a new algorithm that reduces notably the time and space required for constructing Suffix-trees and searching MUMs. Specifically, we construct the Suffix-tree of the shortest genome and while reading the remaining genomes, the list of MUMs is being updated, thus the space spent is linear respect to the first genome. In some sense we can say that it is an on-line algorithm: once $i(i > 1)$ first genomes have been read, we have the common MUMs between these i genomes. Thus, we only construct the Suffix-treeof one genome and we do not need to make the bottom-up traversal of the tree. These improvements are based on:

1. **Building time:** we use a new, more efficient design of Suffix-trees [Hue02]: the labels are stored in the nodes (not in the edges) and the suffix-links pointers have been modified with respect to previous designs [Ukk95]. This implementation allows that leaves have independent pointers and this fact is the starting-point of our work and explains why E. Ukkonen implementation does not holds.
2. **Search time:** We only build a tree of the shortest genome and we use it as a map for searching, updating and marking the MUMs while reading other genomes.

   D. Gusfield applies this idea to solve longest common substring between two strings problem in a space-time-efficient way in [Gus97] pag.135, but he only needs to store and update an integer (the maximum of a set of lengths) while we need to store and update a set of strings (the MUMs).

This paper is mainly devoted to the second improvement, which performs the most important reduction of time and space. A detailed explanation of the first one can be read in [Hue02].

The rest of the paper is organized following the main facts pointed in this introduction. In section 2, we show the new design of Suffix-trees and in section 3, we introduce TSuffix-trees. Section 4 discusses the algorithm for searching MUMs between two genomes and the last section 5 extends this search to many genomes.



(a) Definition of suffix-links. (b) suffix-links of the Suffix-tree of Figure 2(a).

Fig. 3. The suffix-links.

2 Suffix-trees

Let S be a sequence of length n composed by characters of a finite alphabet. We denote $S[i]$ the character located at position i and $S[i,j]$ the substring $S[i]S[i+1]\ldots S[j]$. Then the set of suffixes of S is the set of substrings $S[i,n]$ for $1 \leq i \leq n+1$.

Definition 2. [Gus97] We define the path-label of a node u as the concatenation, in order, of the labels of the nodes belonging to the path from the root to the node u, including the label of the node u.
This definition can be extended to the path-label of a leaf or a character.

Definition 3. The Suffix-tree of the sequence S is a rooted tree T such that:
(i) T has n leaves numbered 1 to n.
(ii) Each node and leaf, except the root, is labeled with a nonempty substring of S.
(iii) Two sons of the same node cannot start with the same character.
(iv) Given a leaf i, the path-label$(i) = S[i,n]$ suffix of S.

We say that a node u *spells* a string *out* when this string is the path-label(u). For instance, in Figure 2(a) the node with label a (having sons $[abs,3]$ and $[baabs,1]$) spells aba out. With the same sense we say that a character spells a string out and that a path of the tree spells a string out.

We assume that the tree is an explicit Suffix-tree for the clarity of the paper, but all lemmas of this paper also hold with some minor changes for implicit Suffix-tree.

Suffix-trees can be built in linear time with respect to the length of the sequence [Ukk95]. This bound is reached due to the existence of pointers called suffix-links . We want to remark here that the definition of these pointers is one of the major differences with previous designs of Suffix-trees: we locate the pointer into the descendant node while E. Ukkonen put it into the parent node. Thus leaves have independent pointers and this fact will be the starting-point of TSuffix-trees data structure and its transversal-path defined in the following section.

Definition 4. Assume that there is a suffix-link that points node u to node v and that the parent of u spells $S[i,j]$ (see Fig. 3(a)), then
(i) the path-label of the parent of v is the suffix $S[i+1,j]$,
(ii) the label of v is a prefix of the label of u.

For instance, see the left tree in figure 3(b): leaf 1 whose parent spells aba out points to leaf 2 because the parent of leaf 2 spells ba out (i) and the label $baabs$ of leaf 2 is a prefix of the label $baabs$ of leaf 1 (ii). Leaf 2 points to a node with label b because the string spelt out by the parent of this node is a (i) and the label b of the node is a prefix of the label of leaf 2 (ii).

We recall now the insertion algorithm. We assume that the characters of $S_1[1..i]$ have been processed and there exists a pointer, called Active-point, to a character of the tree, called Active-character, such that:

A1: The path-label of the Active-character, denoted Active-prefix, spells the prefix $S_1[j,i]$ of the suffix $S_1[j,n]$.
A2: The prefix part $[k,i]$ of suffixes $[k,n]$ for $1 \leq k < j$ cannot be spelt without reaching leaf k.
A3: The leaves of suffixes $[k,n]$ for $1 \leq k < j$ have been previously inserted into the tree.
This property explains that even we have read and processed character $S_1[i]$, the first j leaves have only been inserted.

The algorithm acts in order to maintain these assumptions. The new character read of S, $S[i+1]$, is compared to the next character after the Active-character, denoted Current-character. (**Note:** the Current-character could not be well defined. This could happens when the Active-character was the last one of a node because there exist several possibilities for the Current-character, one for each son. But, abusing the language, we assume that if some character holds the equality then the Current-character represents this option.)

O.1: If $S[i+1]$ is equal to the Current-character then the Active-point is advanced to point the Current-character. Now the path-label of the new Active-character spells $[j,i+1]$ out.
O.2: Otherwise, as $S[j,i+1]$ cannot be spelt out in the tree, then a new node and a new leaf j are inserted at the Active-point. Then the value of j is incremented by one and the corresponding suffix-link is traversed for the recursively application of the algorithm.

Now we address the following two questions: given the Suffix-tree of the first sequence how the MUMs can be found (i) and how can be updated (ii) while reading a second one?

(i) Recall that the building algorithm reads a character of the sequence, looks over the tree searching for a specific prefix and inserts a new leaf if the prefix does not exists. Suppose that we mimic this behavior (as D.Gusfield for computing the matching statistics [Gus97],pag. 133) but reading a character of the second sequence and looking over the Suffix-tree of the first sequence. If the prefix, that is a string of the second sequence, does not exists in the Suffix-tree of the first sequence what does it means? We have found a match which is maximal respect to the left and right sides.
If this divergence occurs while looking over a leaf it means that we have found a match that maximal to the left and right sides and therefore unique for the first sequence.
(ii) Once this unique matching (respect to the first sequence) has been found how can we test for its maximality anduniqueness respect to the second one and where can it be marked if it is a MUM?
We will prove that there exist a path through the tree, denoted transversal-path, and a list of MUMsindexed from this path, where matchings can test both properties and can be stored if they are MUMs.

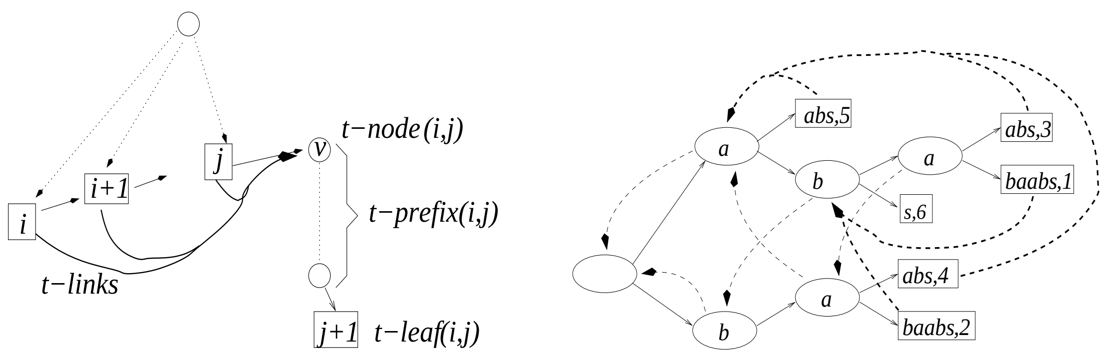
The next sections develop these ideas.

3 TSuffix-trees

In this section we define a slightly modified Suffix-tree, denoted TSuffix-tree, which allows the search and mark of MUMs on-line. This goal is accomplished due to the existence of a transversal path through the tree in which MUMs are marked and the direct access of leaves to this transversal path by means of a new pointers denoted transitive links.

Definition 5. We say that leaves $i, \ldots, j$ form a chain, for short $(i,j) - chain$ if
(i) None leaf points to i.
(ii) Leaf k points to leaf $k+1$ for $i \leq k < j$.
(iii) Leaf j points to an internal node.

Then the numbers of its first and last leaves determine the chain. For instance, in figure 3(b) leaves 1 and 2 form the $(1,2) - chain$ and leaves 3, 4 and 5 the $(3,5) - chain$.



(a) Meaning of the transitive definitions.

(b) New t-link pointers (wide black arrows) of the Suffix-tree of (Fig 2(a))

Fig. 4. Transitive definitions.

Lemma 6. *The leaves of a chain have the same label.*

Proof. This follows from the definition of suffix-links applied to leaves . □

We define now the new pointer called *transitive-link*, for short t-link:

Definition 7. Given a $(i, j) - chain$ such that the last leaf j points to node v, we define for each leaf of the chain a new pointer called transitive-link, for short t-link, that points to v (see Fig. 4(a)).
We say that v is the transitive node, for short t-node, of the complete chain.

Note that for leaf j the t-link is the suffix-link. Figure 4(b) pictures these new pointers for the Suffix-tree of Fig. 2(a). Note that the transitive links substitute the suffix-links of leaves while the suffix-links of nodes are preserved.

Lemma 8. *Given a $(i, j) - chain$ whose leaves are labeled with α, then*
(i) There is a leaf in the subtree rooted at t-node(i,j) whose path-label respects this subtree is α.
(ii) The number of this leaf, called transitive leaf and for short t-leaf, is $j + 1$.

Proof. (i) and (ii) follow from the definition of suffix-links applied to leaf j. □

This previous lemma can be paraphrased saying that the label of a leaf can be spelt out through a top-down path starting at t-node and ending at t-leaf. This fact implies that for every leaf, a prefix part of the label is spelt out through some internal nodes of the tree and a suffix part is spelt out in another leaf, the t-leaf. We separate the prefix part:

Definition 9. We call the prefix part as the *transitive prefix*, for short t-prefix.

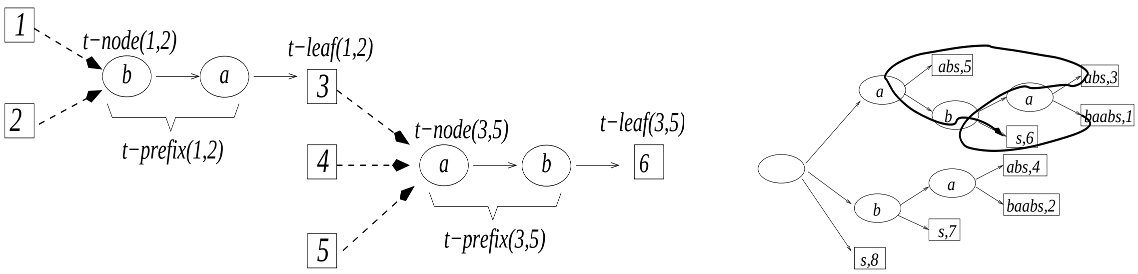
Then it can be proved by induction over the t-leaves that

Lemma 10. *The label of a leaf is spelt out by the concatenation of the t-prefixes of the consecutive chains.*

For instance, suppose that we want to spell the label *baabs* of leaf 2 in Fig. 4(b). Following the t-link of leaf 2 and spelling the t-prefix *ba* we reach leaf 3. Then, following the t-link and spelling the t-prefix *ab* we reach the last leaf 6 with the terminal character *s*.

The structure induced by chains and transitive links is represented in Figure 5(a).

Definition 11. Given the leaf 1, the concatenation of the consecutive t-prefixes until the last leaf of the tree determines a path called *transverse-path* of the tree.



(a) Structure induced by the *chains* and *transitive-links*.

(b) transversal-path of the TSuffix-tree.

Fig. 5. The transversal-path.

Figure 5(a) shows the transversal-paths of leaf 1 and figure 5(b) pictures the transversal-path into the TSuffix-tree.

Figure 5(a) can mislead about the straightness of the transversal-path because it suggests that all t-nodes are different. As figure 5(b) shows, a node can be the t-node of different chains into the transversal-path but we need this straightness because we want that the information stored into a t-node related with a chain was only accessed coming from this chain. But this goal can be easily reached applying the following trick: for each chain we insert a new node pointed by the leaves of the chain and that points to the respective t-node. Then we consider that this trick allows to us assume that there exists a t-node for every chain and that two chains have different t-nodes.

Definition 12. We call TSuffix-trees those Suffix-trees that incorporates transitive-links and the trick.

We relate now the transversal-path with MUMs:

Lemma 13. *Each MUM has a non-empty suffix portion into the transversal-path.*

Proof. If a MUM does not hols the lemma then it ends before reach leaf one and it cannot be a MUM. □

Finally, given a TSuffix-tree we can assume that:

- there is a transversal path that contains the last character into the first sequence of each MUM.
- the first character of each leaf contained into the transversal-path is directly accessed from the leaf.

In the next section we will work with TSuffix-tree, but this is more a theoretical assumption that a practical one. From a practical point of view it is not necessary to update Suffix-tree pointers by t-link pointers neither apply the trick-node in a pre-process because we can do that while searching the MUMs.

4 Maximal unique matching (MUM)

MUMs are characterized by three records:

- the starting position of the MUM in the first sequence.
- the last position of the MUM in the first sequence.
- the last position in the second sequence.

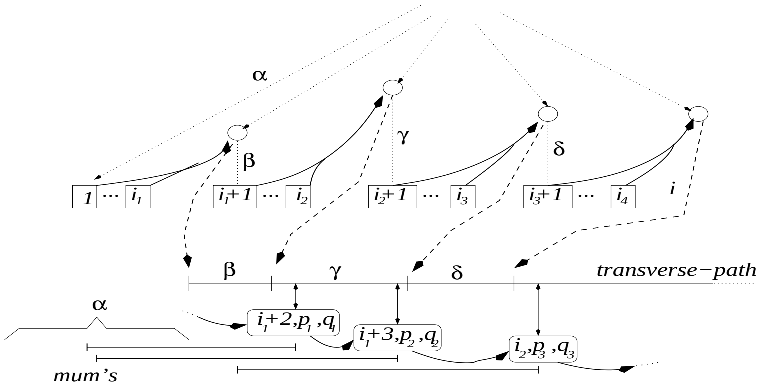


Fig. 6. Meaning of the transversal-path: it is accessed coming from a leaf at the position of the first character of the leaf. The first record of the MUMs, for instance $i_1 + 3$, denotes the leaf from which they have accessed the transversal-path, the second record, for instance P_2, determines the position at which the MUM is linked.

The first record indicates the number of leaf determining the suffix of the first sequence from which the MUM is a prefix. As two MUMs cannot be prefixes of the same suffix, we take this first record as the identifier of the MUM. Then the existence of MUM i implies the existence of a character in the label of the leaf i whose path-label was the MUM. The second record indicates the location where the MUM ends and by lemma 13 also determines the location into the transversal-path where the MUM will be marked.

Now we address the assumptions of the search algorithm. We assume that T is the TSuffix-tree of the sequence S_1, that we have read the symbols $S_2[1..i]$ and that there is an Active-point pointing to the Active-character of the TSuffix-tree and an integer $j \le i$ such that:

B1: the Active-prefix spells $S_2[j, i]$ out (recall that the Active-prefix is the path-label of the Active-character). Note: in TSuffix-trees the spelling starts in the root, but can ends before or after reaching a leaf. In the last case the spelling continues through the transversal-path.
B2: T cannot spells $S_2[k, i]$ out for $1 \le k < j$.
B3: The MUMs that are prefix of the suffixes $1, 2 \ldots j-1$ of sequence S_2 have been found and marked.
B4: There exists a list of MUMs sorted by the second record(see Fig. 6).

The firsts three properties correspond to the Suffix-trees properties A1-A3.

Depending on the properties that the Active-prefix holds while running the algorithm, we define two states $\mathcal{M}$ and $\mathcal{UM}$:

$\mathcal{M}$**:** The Active-prefix is only a match. This fact means that we have not reached a leaf yet.
$\mathcal{UM}$**:** The Active-prefix is an unique-matching respect the first sequence. This fact means that we have reached a leaf.

We design the algorithm in order to maintain the assumptions B1-B4. The sentences of the algorithm depend on the state in which it is, but there is one common feature that we explain first:

Equality-test: this is the first action that is made at each step of the algorithm. This test compares the Current-character, the character following the Active-character, with the new character read $S_2(i+1)$. The option EQ means that both are equals, otherwise we write DIF.
Note: as in Suffix-trees the Current-character could not be well defined, but, abusing the lenguage, we assume that if some possible character holds the equality then the Current-character represents this one option.

Now we are ready for to explain the algorithm:

If *state*= $\mathcal{M}$ **then** make the Equality-test:

If EQ **then** if the Current-character belongs to a leaf then update *state* with $\mathcal{UM}$ and Active-leaf with the number of the leaf. Finally the Active-point is advanced to point the Current-character.
Note: if the unique-matching becomes a MUM, Active-leaf will be the first record and identifier of the MUM.

If DIF **then** the algorithm remains in state $\mathcal{M}$ and the new Active-point is determined following the algorithm of Suffix-trees (Option O.2).

If *state* =$\mathcal{UM}$ **then** make the Equality-test:

If EQ **then** advance the Active-point (Option O.1).

If DIF **then** the string Active-prefix can be divided in two parts: a prefix part spelt before to reach the Active-leaf and a suffix part spelt into the Active-leaf.
Then we spell this suffix part through the transversal-path and compare Active-leaf with the identifier of previously stored MUMs (see Fig. 6). Recall that MUMs are stored in the position determined by the second record (the last character).
MUMs with a smaller identifier continue being MUMs because they start before it , equal or greater MUMs are absorbed by the Active-prefix and can be removed, becoming thus the Active-prefix a MUM.
If the Active-prefix has been spelt and no greater or equal MUMs have been reached, it is even possible to test if it is a MUM: just compare the Active-leaf with the next MUM of the list. If it is greater than the Active-leaf then it is maximal on the left side and it is MUM. If it is smaller or equal there exists a longer previous MUM that includes the Active-prefix, but in the second case we need to mantain the maximal-matching in the list to discard future MUMs contained in it.
Finally, if the Active-prefix is a MUM insert it into the list and move the Active-point as in a Suffix-tree (Option O.2).

The value of j is incremented by one.

We can resume the algorithm saying that we have two structures, the Suffix-tree of the first sequence, and the output list of MUMs. Both structures are perfectly indexed along the transversal-path in both senses, then we can access to a node of the list and return back to the same position into the transversal-path. This facts avoid us to make a bottom-up search.

Theorem 14. *Let S_1 and S_2 be genomic sequences, then the list of MUMs between them can be found on-line in linear time $O(|S_1| + |S_2|)$ and linear space $O(|S_1|)$.*

5 Extension to multiple genomes

We are ready to extend these ideas to the comparision of multiple genomes. We assume that we have constructed the Suffix-tree of the first genome, that we have read the second genome and that we have found the list of MUMs. If we follow the same algorithm while reading the third genome, we will find the set of MUMs between the third and the first genome and thus the common substring between the list and the set will become the MUMs between the three genomes. As the set of new MUMs are stored in the same output list as the previous ones, it is easy to detect the MUMs of the three sequences.

Now, by induction it is easy to prove the following theorem:

Theorem 15. *Let $S_1, \ldots S_n$ be genomic sequences, then the list of MUMs between all them can be found in linear time $O(|S_1| + \ldots + |S_n|)$ and linear space $O(|S_1|)$.*

References

[DKF+99] A.L. Delcher, S. Kasif, R.D. Fleischmann, J. Peterson, O. White, and L. Salsberg. Alignment of whole genomes. *Nucleic Acid Research*, 27(11):2369–2376, 1999.

[DPCS02] A.L. Delcher, A. Phillippy, J. Carlton, and L. Salsberg. Fast algorithm for large-scale genome alignment and comparison. *Nucleic Acid Research*, 30(11):2478–2483, 2002.

[Gus97] D. Gusfield. *Algorithms on strings, trees and sequences.* Cambridge University Press, 1997.

[HKO02] M. Höhl, S. Kurtz, and E. Ohlebush. Efficient multiple genome alignment. *Bioinformatics*, 1(1), 2002.

[Hue02] M. Huerta. Suffix tree construction with slide nodes, research report. http://www.lsi.upc.es. 2002.

[mum99] Mummer. 1999.

[mum02] Mummer. 2002.

[Ukk95] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, (14):249–260, 1995.