Building PQR Trees in Almost-Linear Time

Guilherme P. Telles and João Meidanis

Technical Report - IC-03-026 - Relatório Técnico

November - 2003 - Novembro

Building PQR Trees in Almost-Linear Time

Guilherme P. Telles [*] João Meidanis [†]

Abstract

In 1976, Booth and Leuker invented the PQ trees as a compact way of storing and manipulating all the permutations on n elements that keep consecutive the elements in certain given sets $C_1, C_2, \ldots, C_m$. This problem finds applications in DNA physical mapping, interval graph recognition, logic circuit optimization and data retrieval, for instance. In 1995, Meidanis and Munuera created the PQR trees, a natural generalization of PQ trees. The difference between them is that PQR trees exist for every set collection, even when there are no valid permutations. The R nodes encapsulate subsets where the consecutive ones property fails. In this note we present an almost-linear time algorithm to build a PQR tree for an arbitrary set collection.

1 Introduction

Given a collection of m subsets $C_1, C_2, \ldots, C_m$ of a set U of n elements, the *consecutive ones problem* consists in answering whether there is a permutation of the elements in U that keep the elements of each C_i consecutive.

Many problems can be stated as the consecutive ones problem, such as DNA physical mapping [9], interval graphs recognition [4], logic circuit optimization [3] and data retrieval [5].

The consecutive ones problem was solved by a polynomial algorithm devised by Fulkerson and Gross [4] in 1965. Booth and Leuker [1] invented the PQ trees in 1976, a data structure to solve the problem and compactly represent every valid permutation for U subject to $C_1, C_2, \ldots, C_m$. They also gave a linear algorithm for building PQ trees. PQ trees themselves are useful in solving other problems not directly related to the consecutive ones problem, such as planarity test [1] and interval graphs isomorphism [6]. In 1995, Meidanis and Munuera [7] created the PQR trees, a natural generalization of PQ trees, and gave a quadratic algorithm to build the PQR trees. In a continuation of this work, Meidanis, Porto and Telles [8] extended the algebraic theory behind this problem. PQR trees can be used to solve the problems PQ trees solve, with the additional advantage that when there is no valid permutation PQR trees will point out specific subcollections responsible for the failure of the consecutive ones property [8].

For a long time it was unknown whether the extra R nodes introduce enough complexity into the structure as to make a linear time algorithm impossible. One major problem in this

[*]Institute of Mathematical and Computer Sciences, University of São Paulo, CP 668, 13560-970, São Carlos-SP, Brazil. E-mail: gpt@icmc.usp.br

[†]Institute of Computing, University of Campinas, CP 6176, 13083-970, Campinas-SP, Brazil. E-mail: meidanis@ic.unicamp.br

respect was related to the movement of uncolored ("white") nodes (see Section 4), which did not seem to be bounded by the number of black or gray nodes. Here we show how to implement the algorithm so as to never have to move uncolored nodes. Another important issue was to merge two Q or R nodes in time $O(1)$. We solved this issue by representing the children of such nodes as a union-find structure, where only one designated child point to the parent, and the other siblings point to this designated child directly or indirectly, composing a union-find structure. This leads to $O(1)$ merging, but the price to pay is $O(\alpha(r))$ to find the parent, and an overall almost-linear time bound, where r is the number of ones in the input matrix.

The algorithm given here is not a simple extension of Booth and Leuker's algorithm, but relies on deeper properties of the trees uncovered by the new theory. It uses a fewer number of better organized patterns than the PQ algorithm. The correctness proof also draws heavily on the theory developed and the amortized analysis we present is based on a cleaner potential function. We believe that this contributes significantly towards a definite solution to this problem.

This article is organized as follows. Section 2 gives basic definitions and Section 3 introduces the basics on PQR theory. Our algorithm and analysis come in Sections 4 and 5. Our conclusions appear in Section 6.

2 Definitions

The term **collection** is used here as a synonym for set of sets. Hereafter, collections will always be denoted by calligraph capitals, such as $\mathcal{C}$. To simplify notation, we sometimes write a set as a list of its elements in any order. For example, $A = \{k, l, m, n\}$ can be written as $A = lnkm$. A **permutation** of a finite set U, $|U| = n$, is a one-to-one mapping $\alpha : \{1, 2, \ldots, n\} \mapsto U$. Given a permutation α of the elements of U, and a subset A of U, we say that A is **consecutive** in α when the elements of A appear consecutively in α. For example, if $U = abcdef$ and $\alpha = efcbda$, the subset $A = cdb$ is consecutive in α, while $B = efa$ is not. Given a pair $(U, \mathcal{C})$, with $\mathcal{C} \subseteq \mathcal{P}(U)$, we say that a permutation α of U is **valid** (with respect to $\mathcal{C}$) if all sets $A \in \mathcal{C}$ are consecutive in α. The pair $(U, \mathcal{C})$ has the **consecutive ones property** (C1P) if there is at least one valid permutation.

A PQR tree T is a rooted tree with four types of nodes – P, Q, R and leaves – subject to the following restrictions:

- The leaves are in one-to-one correspondence with the elements of the set U.
- Every P node has at least two children.
- Every Q node has at least three children.
- Every R node has at least three children.

An example of a PQR tree for the set $U = abcdefghijklmnop$ and the collection $\mathcal{C} = \{abcdef, bc, cd, de, ce, ghijklmn, no, op, hijkl, lm\}$ appears in Figure 3.

A PQ tree is just a PQR tree without R nodes [8]. PQ trees are a compact way of representing all valid permutations for a collection with the C1P [1]. Reading the leaves from left to right gives a valid permutation. All other valid permutations are given by equivalent trees, obtained from one another by:

- arbitrary permutations of the children of a P node,
- reversal of the children of a Q node,

In general, regardless of whether the tree has the C1P, a PQR tree is a compact way of representing its completion (see Section 3).

A node v in a PQR tree is identified with the set of leaves having v as ancestor. We use v to denote either the node or the set of its descendent leaves (elements of U) interchangeably.

Given two sets A and B, we define the following operations:

- The **nondisjoint union** of A and B, denoted $A \uplus B$, is equal to $A \cup B$ provided that $A \cap B \neq \emptyset$.
- The **noncontained difference** of A and B, denoted $A \between B$, is equal to $A \setminus B$ provided that $B \not\subseteq A$.

Two sets A and B are **orthogonal**, denoted $A \perp B$, if either $A \subseteq B$, or $B \subseteq A$, or else $A \cap B = \emptyset$. Similarly, a collection $\mathcal{C}$ is orthogonal to a set A if every set in $\mathcal{C}$ is orthogonal to A.

3 PQR Theory

Meidanis and Munuera [7] created a theory, later consolidated by Meidanis, Porto, and Telles [8], that greatly helps reasoning about the problem and lies at the heart of the algorithm we present here. In this section we recall basic findings of this theory that will be used in the sequel.

The main points of the theory are:

- the realization that the set operations *intersection*, *noncontained difference*, and *nondisjoint union* produce consecutive sets from consecutive sets and can be used to enrich the input set collection $\mathcal{C}$ into a normalized, complete collection $\overline{\mathcal{C}}$. The PQR tree associated to $\mathcal{C}$ is essentially unique and depends only on $\overline{\mathcal{C}}$.
- the realization that the concept of *orthogonality* between sets can be used to characterize the nodes that appear in a PQR tree for $\mathcal{C}$, through the collection $\mathcal{C}^{\perp}$, which holds every set orthogonal to $\mathcal{C}$.

To formalize these statements, it is important to have a way of obtaining the complete collection $\mathit{Compl}(T)$ associated to a PQR tree T from the tree itself. The collection $\mathit{Compl}(T)$ is defined as follows, where $\mathit{Desc}_T(S)$ is the set $\bigcup_{v \in S} v$.

1. The *trivial sets* $\emptyset$, U, and $\{x\}$ for all $x \in U$ are contained in $\mathit{Compl}(T)$,
2. $\mathit{Desc}_T(S)$ is in $\mathit{Compl}(T)$ if S is the set of all children of a P node of T,
3. $\mathit{Desc}_T(S)$ is in $\mathit{Compl}(T)$ if S is a set of consecutive children of a Q node of T,
4. $\mathit{Desc}_T(S)$ is in $\mathit{Compl}(T)$ if S is an arbitrary set of children of an R node of T,
5. No other sets are in $\mathit{Compl}(T)$.

The following results from an earlier work [8] are important here:

Theorem 1 *A collection $\mathcal{C}$ corresponds to a PQR tree T if and only if*

$$\overline{\mathcal{C}} = Compl(T).$$

Theorem 2 *If T is a PQR tree for a collection $\mathcal{C}$ then the nodes of T correspond exactly to the sets in $\overline{\mathcal{C}} \cap \mathcal{C}^{\perp}$.*

In the analysis that follows, given a PQR tree T corresponding to a collection $\mathcal{C}$, we denote by $T + S$ any PQR tree corresponding to the collection $\mathcal{C} \cup \{S\}$.

4 Almost-Linear Time Algorithm

The algorithm for PQR tree construction in this section is an on-line algorithm, in the sense that, given a set U and a collection $C \subseteq \mathcal{P}(U)$, it builds a PQR tree T for the collection $\mathcal{C}$ processing (or adding) one set in $\mathcal{C}$ at a time.

The algorithm for adding a set S to a tree T appears below. Here the LCA is the least common ancestor node of all leaves in S.

```
1. mark the leaves corresponding to $S$
2. color the tree and find the $LCA$
3. restructure the tree, getting rid of gray nodes
4. adjust the $LCA$
5. uncolor the tree
```

What follows is a better explanation of each step.

Step 1 Leaves corresponding to the elements of S are colored black.

Step 2 Coloring of other nodes of T is done as follows:

- a node v is colored black when $v \subseteq S$,
- a node v is colored gray when $v \not\perp S$,
- a node v is left uncolored ("white") when either $v \cap S = \emptyset$ or $S \subset v$,

with the following exception: the LCA will always be left uncolored, even if by the above rules it would have to be colored black or gray.

This coloring can be accomplished simultaneously with finding the LCA of all black leaves as described in the original Booth and Leuker paper [1].

Notice that the nodes colored gray will not be nodes of any tree $T + S$, since they are not orthogonal to S. This motivates the next step of the algorithm, which restructures the tree until no gray node remains.

From this point on in the paper, let us denote the set of black, gray and white children of a node v respectively by $B(v)$, $G(v)$ and $W(v)$.

Template	Operations
PP	Transform P node into Q node PQ template
PQ	Prepare the LCA Reverse Q node Move children away from LCA
PR	Prepare the LCA Move children away from LCA
QP	Transform P node into Q node QQ template
QQ	Reverse the LCA Reverse Q node Merge into the LCA
QR	Reverse the LCA Reverse Q node Merge into the LCA
RP	Transform P node into Q node RQ template
RQ	Merge into the LCA
RR	Merge into the LCA

Table 1: Templates for Step 3.

Step 3 This step repeatedly kills, merges, or uncolors gray nodes, sometimes also creating new nodes when needed, until no gray nodes remain in the tree. This step is therefore the iteration of a simpler task, which is the processing of one gray node.

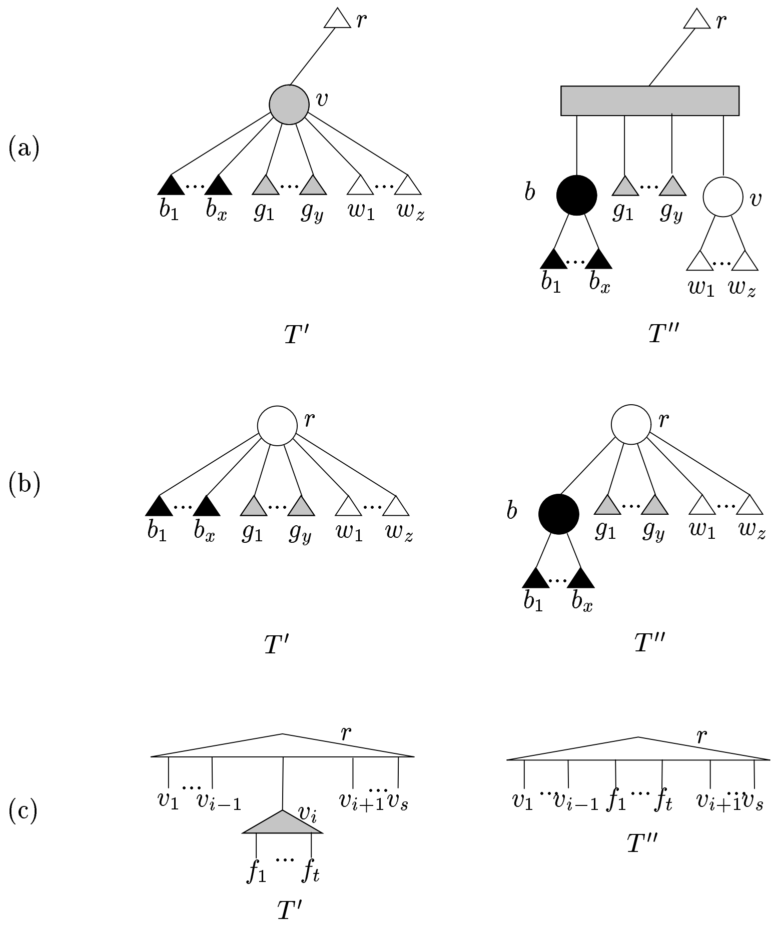
```
while there is a gray child $v$ of the $LCA$
    process $v$ using the appropriate template
end while
```

Templates are selected based on the type of the LCA and on the type of its child being processed. Every template is associated with a set of operations or other templates, as shown in Table 1. The operations that compose templates are illustrated in Figures 1 and 2, where a triangle represents a node which type is either Q or R, and r is the label for the LCA. Operations "Reverse Q node" and "Reverse the LCA" are not illustrated in the figures. The first consists in reversing a Q node when its leftmost child is white and its rightmost child is gray or black or when its leftmost child is gray and its rightmost child is black. In other words, the goal is to leave a node in the left end at least as "dark" as the one in the right end. The second consists in reversing the LCA when it is a Q node using the same rules for "Reverse Q node", but with respect to the left and right neighbors of the child of the LCA being processed. When the child v being processed has only one neighbor, then if this neighbor is black or gray it should be on the left of v; when it is white, it should be on the right of v.

Notice also that while it seems that we have 9 templates, nodes of type Q and R are treated rather similarly, and one can use only the 4 templates PP, PQ, QP, QQ (treating an R node as if it were of type Q) without affecting the correctness of the algorithm and without a significant time penalty either.

Step 4 After Step 3, there are no gray nodes left in the tree. Therefore, all maximal black nodes are children of the LCA. We have to adjust the LCA as follows, according to its type.

- the LCA is a P-node: if the LCA has 2 or more black children, and at least one white child, create a new child of type P of the LCA and move all black maximal nodes to this new node.
- the LCA is a Q-node: if all maximal black nodes are consecutive, do nothing; otherwise



(a)
r
v
b_1 b_x g_1 g_y w_1 w_z
T'
r
b
g_1 g_y
v
b_1 b_x
w_1 w_z
T''
(b)
r
b_1 b_x g_1 g_y w_1 w_z
T'
r
b
g_1 g_y w_1 w_z
b_1 b_x
T''
(c)
r
v_1 v_{i-1} v_i v_{i+1} v_s
f_1 f_t
T'
r
v_1 v_{i-1} f_1 f_t v_{i+1} v_s
T''

Figure 1: Operations that compose PQR templates: (a) Transform P node into Q node, (b) Prepare the LCA, (c) Merge into the LCA.

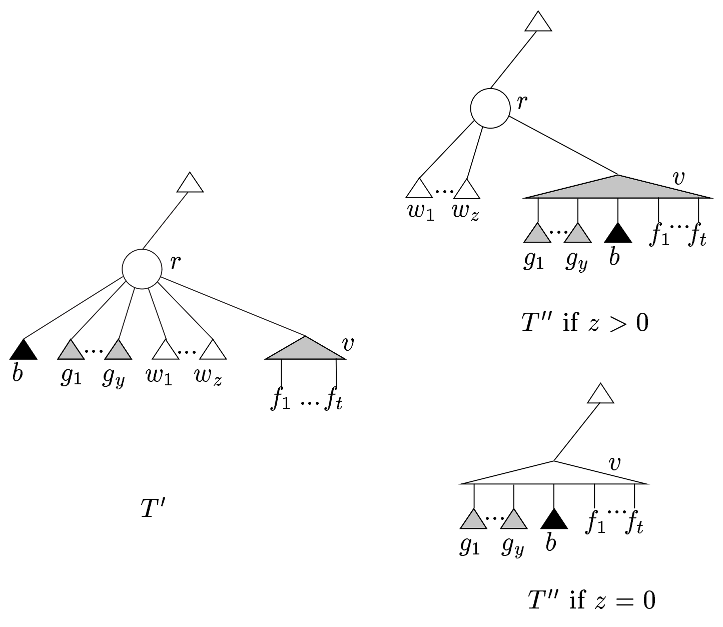


Figure 2: Operation "Move children away from the LCA", that composes PQR templates.

change the type of the LCA to R.

- the LCA is a R-node: do nothing.

Step 5 Uncolor all black nodes.

Figure 3 shows the execution of the algorithm for the tree shown and $S = ghm$.

5 Correctness and complexity

5.1 Correctness

To prove that the algorithm is correct it is necessary to show that, if T' and T'' are the colored trees before and after any given step in the algorithm, then

$$\overline{Compl(T') \cup S} = \overline{Compl(T'') \cup S} \tag{1}$$

This is easy to see for Steps 2, 4, and 5. For Step 3, we need to show the invariant (1) holds for all the operations on PQR trees described in the previous section. To illustrate the proof method, we will show how it applies to operation Prepare the LCA. The nodes in T' all appear also in T'', with the same types. Therefore $\overline{Compl(T') \cup S} \subseteq \overline{Compl(T'') \cup S}$. On the other hand, we can write

$$b = (r \cap S) \between g_1 \between \dots \between g_y,$$

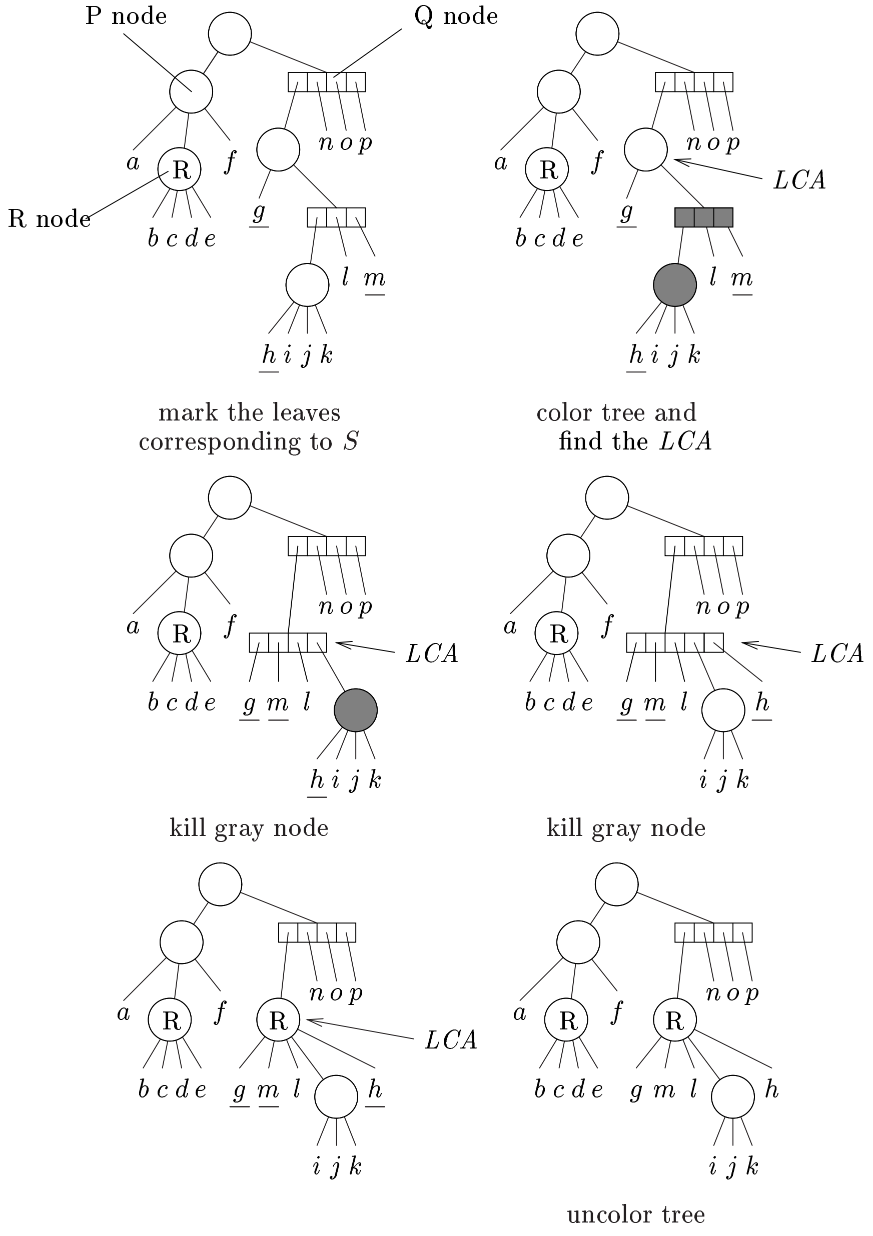

P node
Q node
R node
a b c d e f g h i j k l m n o p
R
LCA
mark the leaves corresponding to S
color tree and find the LCA
kill gray node
kill gray node
uncolor tree

Figure 3: Execution of the algorithm for the tree shown and $S = ghm$.

which shows that $\overline{Compl(T'') \cup S} \subseteq \overline{Compl(T') \cup S}$, completing the proof.

Noticing that after the last step we have $S \in Compl(T'')$, we have actually

$$\overline{Compl(T'') \cup S} = Compl(T + S),$$

showing that we end up with the right tree after adding S.

5.2 Complexity

In this section we will assume that we are given a collection $C_1, C_2, \ldots, C_m$ of subsets of $U = \{1, 2, \ldots, n\}$ to construct their PQR tree, and that $r = \sum_j |C_j|$.

The main operations involved in the algorithm are: node creation, node killing, node reversal, node moving, node merging, node coloring, and node uncoloring. We will consider first the operations done in Step 3, which is the most intricate step.

Each node must know its parent because of the way coloring is done (bottom to top). To guarantee the almost-linear time bound, we must implement the set of children of a Q or R node as a union-find structure[2]. One designated child will hold a pointer to their parent, and all other siblings will point to other siblings which are closer to the designated child. This permits execution of the "Merge into the LCA" operation in $O(1)$ time, but the price to pay is an $O(\alpha(r))$ bound for finding the parent of a node.

We will show that the algorithm performs $O(r)$ operations, with cost $O(\alpha(r))$ each. Each time a set S is added, work proportional to $|S|$ is allowed. It turns out that sometimes a small set is added and the tree changes significantly, meaning that we cannot guarantee that adding S can be done within $O(|S|)$ time. However, an amortized $O(|S|)$ bound holds. We will show this by exhibiting a tree potential function that records the part of the $O(|S|)$ not actually used for work yet and which is available to future operations.

In what follows, a *move* operation includes moving and merging nodes.

Theorem 3 *The amount of operations involved in Step 3 is dominated by the move operations.*

Proof:

Leaves are never created or killed. Internal nodes are created with zero children, and only internal nodes with at most one child are killed. Therefore, to kill a node its children must be moved elsewhere first. Likewise, the creation of a node is always followed by movements to bring children in. The reversal of a node is always followed by a movement as well.

Node coloring occurs when a new node is colored black. Therefore, coloring is dominated by creation. Uncoloring occurs when all colored children of a P-node are moved elsewhere. Therefore, uncoloring is dominated by movement. □

Theorem 4 *The total number of moves in Step 3 of adding a set S to a tree T is bounded by:*

$$\begin{aligned} m_3(T,S) \quad \leq \quad & |B(r)| + |G(r)| + \\ & \sum_{v \text{ gray P-node}} (|B(v)| + |G(v)|) + \\ & \sum_{v \text{ gray Q/R-node}} 1, \end{aligned}$$

type of LCA	type of v	movements
P	P	$\lvert B(r)\rvert + \lvert G(r)\rvert + \lvert B(v)\rvert + \lvert G(v)\rvert$
P	Q/R	$\lvert B(r)\rvert + \lvert G(r)\rvert$
Q/R	P	$\lvert B(v)\rvert + \lvert G(v)\rvert$
Q/R	Q/R	1

Table 2: Upper bound on number of movements in each template case of Step 3.

where r is the LCA.

Proof: See Table 2. Notice that in every case, v ceases to exist or becomes uncolored. Therefore, each gray node is processed exactly once. Notice also that the LCA is of type P at most in one iteration (the first one), and therefore enters the calculation at most once. $\square$

The main result of this section relies on the concept of *tree potential.* The potential of a tree T is defined as follows.

$$pot(T) = \sum_{v \text{ P-node of } T} |v| + \sum_{v \text{ Q/R-node of } T} 1 + .$$

Theorem 5 *The gain in potential $\Delta pot(T,S) = pot(T+S) - pot(T)$ when adding a set S to a tree T satisfies*

$$\Delta pot(T,S) \quad \leq \quad \sum_{v\in B(r)} |v| - \sum_{v \textit{ gray P-node}} \sum_{u\in G(v)} |u| - \sum_{v \textit{ gray Q/R-node}} 1,$$

where r is the LCA.

Proof: The nodes in $T+S$ that are not in T are (at most):

- P-node with all black children of r
- P-nodes with all black children of gray P-nodes
- P-nodes with all uncolored children of gray P-nodes
- Q-node child of an LCA of type P.

The nodes in T which are not in $T+S$ are exactly the gray nodes. Therefore,

$$\begin{aligned}\Delta pot(T,S) \quad &\leq \quad \sum_{v\in B(r)} |v| + \sum_{v \text{ gray P-node}} \sum_{u\in B(v)\cup W(v)} |u| + 1 - \\ &\qquad \sum_{v \text{ gray P-node}} |v| - \sum_{v \text{ gray Q/R-node}} 1 \\ &\leq \quad \sum_{v\in B(r)} |v| - \sum_{v \text{ gray P-node}} \sum_{u\in G(v)} |u| - \sum_{v \text{ gray Q/R-node}} 1.\end{aligned}$$

$\square$

Theorem 6 *The total number of operations work(T, S) involved in adding S to T satisfies*

$$\mathit{work}(T, S) = O(|S| + m_3),$$

where $m_3 = m_3(T, S)$ is as in Theorem 4.

Proof: Steps 1, 4 and 5 are $O(|S|)$. Step 2 is $O(|S| + m_3)$, as proved by Booth and Leuker [1]. In their paper they call $\mathit{prunned}(T, S)$ the subtree of all gray nodes plus the LCA, and show that Step 2 is $O(|\mathit{prunned}(T, S)|)$. It turns out that $|\mathit{prunned}(T, S)| = O(m_3)$, because every gray node v satisfies $|B(v)| + |G(v)| \geq 1$. □

Theorem 7

$$\mathit{work}(T, S) + \Delta \mathit{pot}(T, S) = O(|S|).$$

Proof: It suffices to show that

$$m_3 + \Delta \mathit{pot}(T, S) = O(|S|).$$

From previous formulas, by straightforward algebraic manipulation,

$$\begin{aligned} m_3 + \Delta \mathit{pot}(T, S) &\leq |B(r)| + |G(r)| + \sum_{v \in B(r)} |v| + \sum_{v \text{ gray P-node}} |B(v)| \\ &\leq 3|S|. \end{aligned}$$

This last inequality is due to the following reasons:

$$|B(r)| + \sum_{v \text{ gray P-node}} |B(v)| \leq |S|$$

since $B(r) \cup \bigcup_{v \text{ gray P-node}} B(v)$ is a disjoint union contained in S;

$$|G(r)| \leq |S|,$$

since the sets of $G(r)$ are disjoint; finally,

$$\begin{aligned} \sum_{v \in B(r)} |v| &= \left| \bigcup_{v \in B(r)} v \right| \\ &\leq |S|, \end{aligned}$$

since the sets of $B(r)$ are disjoint and contained in S.

□

From this last theorem we can assess the total work needed to add sets $C_1, C_2, \ldots, C_m$ to a universal tree T_0 (a tree with just one P internal node with all leaves as its children), which results in the PQR tree T for the collection of these given sets. Adding the number of operations for adding all the sets, we end up with:

$$\text{total ops} + \mathit{pot}(T) - \mathit{pot}(T_0) = O(r + m),$$

or

$$\text{total ops} = O(r + m + n),$$

since $\mathit{pot}(T) \geq 0$, and $\mathit{pot}(T_0) = n$. As we saw, each operation can be performed in $O(\alpha(r))$, and therefore the almost-linear bound follows.

6 Conclusions

This work represents a significant contribution towards a cleaner and more general solution to the consecutive ones problem. Based on the theory developed by Meidanis and Munuera [7], later extended by Meidanis, Porto, and Telles [8], we propose a new algorithm to build PQR trees corresponding to a set collection, in time proportional to the collection's size.

This algorithm is more intuitive than the original PQ tree algorithm proposed by Booth and Lueker, and, in addition, constructs tress for all input collections, regardless of their consecutive ones status. The resulting tree will point out specific subcollections responsible for the failure of the consecutive ones property in the collections as a whole.

Several practical applications modeled by the consecutive ones problem require the ability to deal with input data with errors. It is important to explore the possibility of using PQR trees to address this issue.

Acknowledgments

We acknowledge the financial support of Brazilian agencies FAPESP and CNPq.

References

[1] K.S. Booth and G.S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976.

[2] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, 2001. ISBN 0-262-03293-7 (MIT Press); ISBN 0-07-013151-1 (McGraw-Hill).

[3] A.G. Ferreira and S.W. Song. Achieving optimality for gate matrix layout and PLA folding: a graph theoretic approach. In I. Simon, editor, *Latin'92*, volume 583 of *Lecture Notes in Computer Science*, pages 139–153, São Paulo, Brasil, 1992. Springer-Verlag.

[4] D.R. Fulkerson and O.A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965.

[5] S.P. Ghosh. File organization: the consecutive retrieval property. *Communications of the ACM*, 15(9):802–808, 1972.

[6] G.S. Lueker and K.S. Booth. A linear time algorithm for deciding interval graph isomorphism. *Journal of the ACM*, 26(2):183–195, 1979.

[7] J. Meidanis and E.G. Munuera. A simple linear time algorithm for binary phylogeny. In N. Ziviani, J. Piquer, B. Ribeiro, and R. Baeza-Yates, editors, *Proceedings of the XV Intern. Conf. of the Chilean Computer Science Society*, pages 275–283, Arica, Chile, 1995.

[8] J. Meidanis, O. Porto, and G.P. Telles. On the consecutive ones property. *Discrete Applied Mathematics*, 88:325–354, 1998.

[9] J.C. Setubal and J. Meidanis. *Introduction to computational molecular biology.* PWS Publishing Co., 1997.