Building PQR Trees in Almost-Linear Time

Guilherme P. Telles [a,1], Joao Meidanis [b,1]

[a] *Institute for Mathematical and Computer Sciences, University of Sao Paulo, PO Box 668, 13560-970, Sao Carlos, Sao Paulo, Brazil.*

[b] *Scylla Bioinformatics, R. James Clerk Maxwell, 360, 13069-380, Campinas, Sao Paulo, Brazil and Institute of Computing, University of Campinas, PO Box 6176, 13083-970, Campinas, Sao Paulo, Brazil.*

Abstract

The problem of finding permutations that keep consecutive the elements in certain given sets $S_1, S_2, \ldots, S_m$ appears frequently, for example in DNA physical mapping, interval graph recognition and data retrieval. Such permutations are called *valid*. When there are valid permutation for the given sets, a PQR tree stores them all compactly. When there are no valid permutations, the R nodes of a PQR tree point out subsets responsible for the failure. This feature is interesting for problems whose input data may have errors, such as DNA physical mapping. In this paper we present an almost-linear time algorithm for PQR tree construction. Our algorithm replaces the pattern matching scheme of PQ trees, the predecessor of PQR trees, with simpler code that is easier to implement.

1 Introduction

Given a collection of m subsets $S_1, S_2, \ldots, S_m$ of a set U, the *consecutive ones problem* consists in answering whether there is a *valid permutation* of

[1] Supported by Brazilian agencies CNPq and Fapesp.

the elements in U, that is, a permutation that keeps the elements of each S_i consecutive. Many other problems can be stated as the consecutive ones problem, such as DNA physical mapping [5], interval graph recognition [3], logic circuit optimization [2] and data retrieval [4].

The consecutive ones problem was solved by a polynomial time algorithm devised by Fulkerson and Gross [3] in 1965. In 1976, Booth and Leuker [1] invented the PQ tree, a data structure that compactly represents every valid permutation, and gave a linear time algorithm for building it. In 1995, Meidanis and Munuera [6] created the PQR tree, a natural generalization of the PQ tree, and gave a quadratic algorithm to build it. When there are no valid permutations for an instance of the consecutive ones problem, the PQ tree is null but the PQR tree is not and its R nodes will point out subcollections that cannot have valid permutations. This feature is specially interesting in applications where experimental errors may occur and the problem to be solved becomes harder, as in DNA physical mapping [5]. In such cases PQR trees may conduce to a solution that is closer to the solution that would be obtained if the data had no errors.

In 1998, Meidanis, Porto and Telles [7] extended the algebraic theory behind the consecutive ones problem. In this work we introduce an algorithm for building PQR trees in almost-linear time. The algorithm uses a small number of patterns that rely on properties of the trees unveiled by the PQR theory. The correctness proof also draws heavily on the theory. We believe that this contributes significantly towards a better solution for this problem.

This article is organized as follows. Section 2 gives basic definitions and introduces the basics of PQR theory. The algorithm and its analysis come in Section 3. Our conclusions appear in Section 4.

2 Definitions and PQR Theory

A set of sets is called **collection** and denoted by calligraphic capital letters. A **permutation** of a finite set U, $|U| = n$, is a one-to-one mapping $\alpha : \{1, 2, \ldots, n\} \mapsto U$. The power set of a set U is denoted $\mathcal{P}(U)$. A **trivial** subset of U is a set with 0, 1, or n elements. Given a permutation α of the elements of U and a subset A of U, A is **consecutive** in α when the elements of A appear consecutively in α. For example, if $U = \{a, b, c, d, e, f\}$ and $\alpha = [e, f, c, b, d, a]$, $A = \{c, d, b\}$ is consecutive in α. Observe that there is no order relation defined for A. Another example is the set $B = \{e, f, a\}$, that is not consecutive in α. Given a pair $(U, \mathcal{C})$, $\mathcal{C} \subseteq \mathcal{P}(U)$, a permutation α of U is **valid** (with respect to $\mathcal{C}$) if every set $A \in \mathcal{C}$ is consecutive in α. A

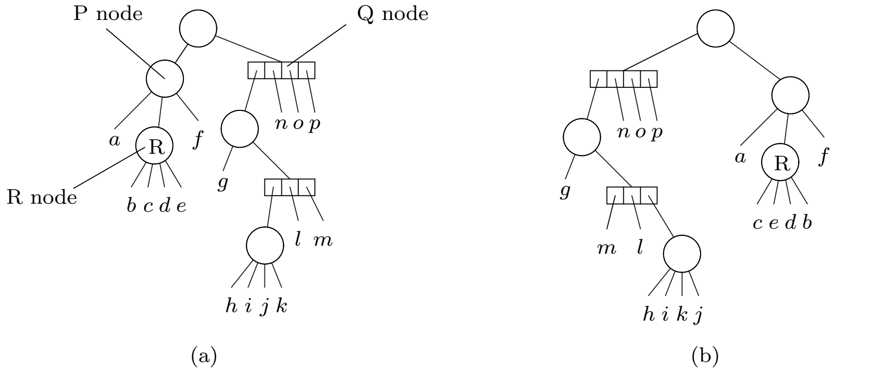

Fig. 1. (a) A PQR tree over the set $U = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p\}$ and the collection $\mathcal{C} = \{a, b, c, d, e, f\}, \{b, c\}, \{c, d\}, \{d, e\}, \{c, e\}, \{g, h, i, j, k, l, m, n\}, \{n, o\}, \{o, p\}, \{h, i, j, k, l\}, \{l, m\}\}$. (b) A tree equivalent to the one in (a). The pair $(U, \mathcal{C})$ does not have the consecutive ones property. Observe that the R node in the trees indicates that the sets $\{b, c\}, \{c, d\}, \{d, e\}$ and $\{c, e\}$ are responsible for the C1P failure. It is not possible to find a permutation of U that keeps these sets consecutive at the same time.

pair $(U, \mathcal{C})$ has the **consecutive ones property** (C1P) if there is at least one valid permutation. The size of an instance for the consecutive ones problem is $n + m + r$, where $n = |U|, m = |\mathcal{C}|, r = \sum_{S \in \mathcal{C}} |S|$.

A **PQR tree over a set** U 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 U, every P node has at least two children, and every Q or R node has at least three children. An example of a PQR tree appears in Figure 1a. Two PQR trees are **equivalent** when one can be obtained from the other by zero or more of the following operations: arbitrary permutations of the children of a P or an R node; reversal of the children of a Q node. Examples are the trees in Figures 1a and 1b.

The PQR theory [6,7] greatly helps reasoning about the C1P and lies at the heart of the algorithm we present here. The main points of the theory are:

- the realization that the operations *intersection*, *noncontained difference*, and *nondisjoint union* produce consecutive sets from consecutive sets and can be used to enrich the input collection $\mathcal{C}$ into a *complete* collection $\overline{\mathcal{C}}$. The PQR tree for $\mathcal{C}$ is unique up to equivalence 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 $\mathcal{C}$.

Formally, given two sets A and B, the **nondisjoint union** of A and B is equal to $A \cup B$ provided that $A \cap B \neq \emptyset$, and the **noncontained difference** of A and B is equal to $A \setminus B$ provided that $B \not\subseteq A$. The sets A and B are **orthogonal** if $A \subseteq B$, $B \subseteq A$, or $A \cap B = \emptyset$. A collection $\mathcal{C}$ is orthogonal to a set A if every set in $\mathcal{C}$ is orthogonal to A. Given $\mathcal{C} \subseteq \mathcal{P}(U)$, the collection $\mathcal{C}^{\perp}$ is the collection of every set in $\mathcal{P}(U)$ that is orthogonal to $\mathcal{C}$.

A collection is **complete** when it contains the trivial sets and is closed under the operations *intersection*, *nondisjoint union* and *noncontained difference*. Given a collection $\mathcal{C} \subseteq \mathcal{P}(U)$, its completion $\overline{\mathcal{C}}$ is the smallest complete collection that contains $\mathcal{C}$. Analogously, the completion of a PQR tree T, denoted by $Compl(T)$, is a collection that contains the empty set, the set of all leaves in T, the sets $\{x\}$ such that x is a leaf of T, and the sets $\bigcup_{v \in S} v$ for every set S such that S is a set of all children of a P node of T, a set of consecutive children of a Q node of T, or an arbitrary set of children of an R node of T.

We say that T is a PQR tree for $\mathcal{C}$ when $\overline{\mathcal{C}} = Compl(T)$. In general there are several PQR trees for the same collection $\mathcal{C}$, and therefore we cannot speak of **the** PQR tree for $\mathcal{C}$. However, all these trees are equivalent, hence we can unambiguously define an equivalence class of trees for a given collection $\mathcal{C}$ [7].

The following results from an earlier work [7] are important here. The first one states that for every collection there is a PQR tree, the second characterizes the nodes in a PQR tree and the third relates the C1P to the PQR trees.

Theorem 2.1 *For any collection $\mathcal{C}$ over a set U, there is a PQR tree such that $Compl(T) = \overline{\mathcal{C}}$.*

Theorem 2.2 *If T is a PQR tree for a collection $\mathcal{C}$ then the nodes of T correspond exactly to the sets in $\overline{\mathcal{C}} \cap \mathcal{C}^{\perp}$.*

Theorem 2.3 *If T is a PQR tree for a collection $\mathcal{C}$ and T has no R nodes, then the set of all permutations of U valid with respect to $\mathcal{C}$ can be obtained reading the leaves of every tree equivalent to T from left to right.*

3 Almost-Linear Time Algorithm

Our algorithm for PQR tree construction starts with a tree that corresponds to the empty collection (a single internal node of type P with all leaves as its children). It then adds one set S_i, $1 \leq i \leq m$, from $\mathcal{C}$ at a time, in arbitrary order. After the addition of a set S_i, the resulting tree is a tree in the equivalence class of PQR trees for $\{S_1, \ldots, S_i\}$. The algorithm for adding a set S_i to a tree T appears below.

1. color the leaves corresponding to S_i
2. color the tree and find the LCA
3. restructure the tree, eliminating gray nodes
4. adjust the LCA
5. uncolor the tree

In steps 1 and 2 tree nodes are colored depending on whether they are contained in S_i (black), contain or are disjoint from S_i (white) or have an strict intersection with S_i (gray). The least common ancestor (LCA) of all leaves in S_i is found as the nodes are marked [1]. The nodes colored gray will not be nodes of any resulting tree, since they are not orthogonal to S_i.

Step 3 repeatedly kills, merges, or uncolors gray nodes, sometimes also creating new nodes, until no gray nodes remain. This step is the ordered application of the reversal of Q nodes and of the operations illustrated in Figure 2. Reversal depends on the colors of two children of a node. Selecting an operation depends on the the type of the LCA and on the type of the gray node being processed.

After Step 3, there are no gray nodes left in the tree. Therefore, all black subtrees are children of the LCA. Step 4 adjusts the LCA according to its type, applying the operation "Prepare the LCA" if it is a P node, or checking for black children consecutiveness (and occasionally changing its type to R) if it is a Q node.

To prove that the algorithm is correct we have shown [8] that if T' and T'' are respectively the uncolored trees before and after any given step in the algorithm, then the invariant $\overline{Compl(T') \cup \{S_i\}} = \overline{Compl(T'') \cup \{S_i\}}$ holds. This is easy to see for Steps 1, 2, and 5. For Steps 3 and 4, we showed that the invariant holds for all the operations on PQR trees described previously. Based on these facts, we concluded that if T' and T'' are the trees before and after the iteration that adds set S_i then $\overline{Compl(T') \cup \{S_i\}} = \overline{Compl(T'') \cup \{S_i\}}$. After Step 4, $S_i \in Compl(T'')$, which implies $\overline{Compl(T'') \cup \{S_i\}} = Compl(T'')$, showing that tree T'' is in fact in a PQR tree for $\{S_1, \ldots, S_i\}$.

For the complexity analysis, we have shown [8] first that the operations of moving a node dominate the running time. Then we have shown that the cost of every operation is $O(\alpha(r,r))$, where α is the inverse Ackermann function. Finally, we used amortized analysis to show that the number of operations needed to add sets $S_1, \ldots, S_m$ to the tree for the empty collection is $O(n+m+r)$, and then the almost-linear bound follows.

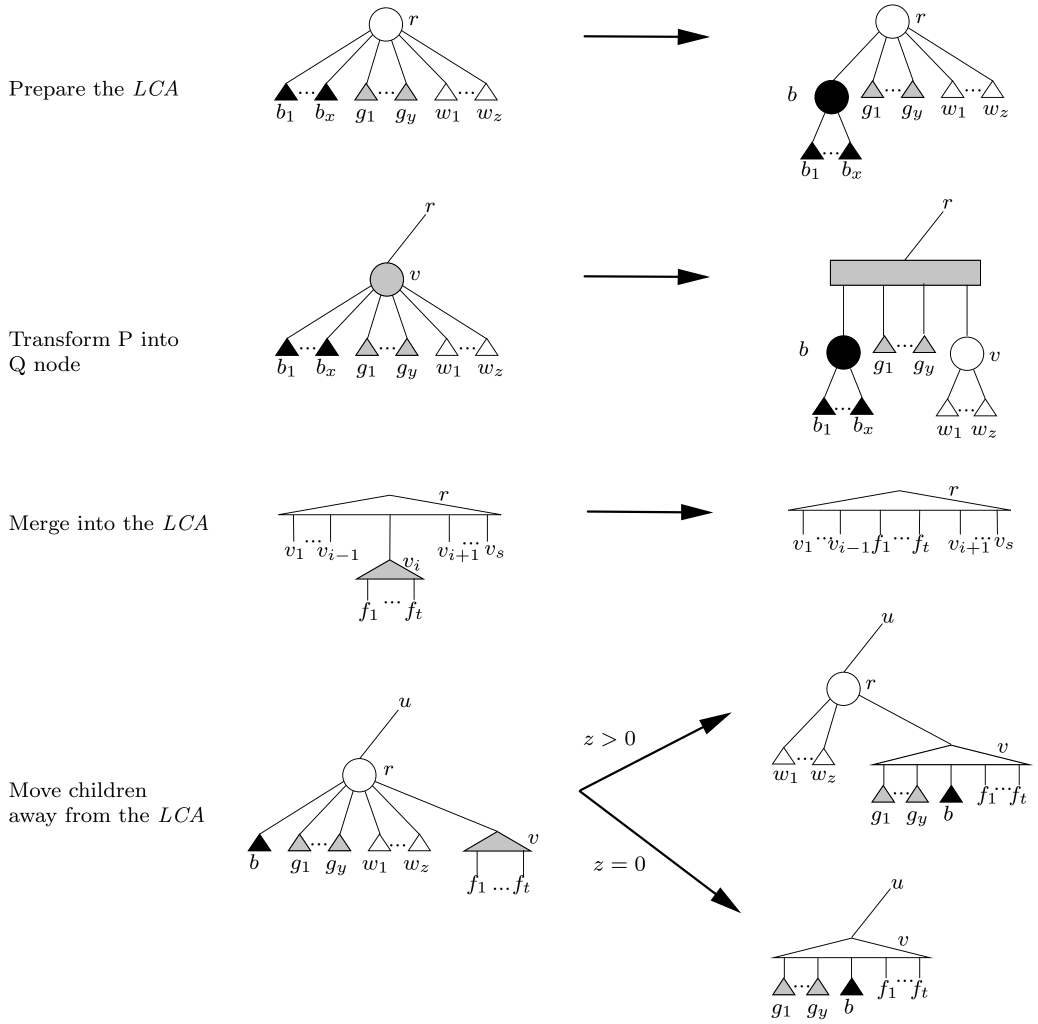

Fig. 2. Operations for removing gray nodes from a PQR tree. A triangle represents a node whose type is either Q or R, r is the label for the LCA and v is the label for the gray child being processed.

4 Conclusions

This work is a contribution towards a cleaner and more general solution to the consecutive ones problem. Based on the PQR theory, we propose a new algorithm to build PQR trees in time almost-linear in the collection's size.

Our algorithm replaces the entire pattern matching scheme of the predecessor PQ trees with simpler code, which depends only on the types of a gray node and of the LCA, and is much easier to implement. In addition, it constructs trees for all input collections, regardless of their consecutive ones status. The R nodes of the resulting tree points out subcollections responsible for the failure of the consecutive ones property if that is the case, as illustrated in Figure 1.

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. We expect our contribution, besides being an interesting theoretical result, will lead to practical algorithms for dealing with noisy data.

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] A.G. Ferreira and S.W. Song. Achieving optimality for gate matrix layout and PLA folding: a graph theoretic approach. In *Proceedings of Latin'92*, pages 139–153, São Paulo, Brasil, 1992.

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

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

[5] W.-L. Hsu. On physical mapping algorithms: an error tolerant test for the consecutive ones property. In *Proceedings of the 3rd Int. Conf. on Computing and Combinatorics*, pages 242–250, 1997.

[6] J. Meidanis and E.G. Munuera. A simple linear time algorithm for binary phylogeny. In *Proceedings of the XV Int. Conf. of the Chilean Computer Science Society*, pages 275–283, Arica, Chile, 1995.

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

[8] G.P. Telles and J. Meidanis. Building PQR trees in almost-linear time. Technical Report TR-03-26, Instituto de Computacao, UNICAMP, 2003.