New Algorithms for Structure Informed Genome Rearrangement

Eden Ozery
Ben Gurion University of the Negev, Israel

Meirav Zehavi
Ben Gurion University of the Negev, Israel

Michal Ziv-Ukelson
Ben Gurion University of the Negev, Israel

Abstract

We define two new computational problems in the domain of perfect genome rearrangements, and propose three algorithms to solve them. The rearrangement scenarios modeled by the problems consider Reversal and Block Interchange operations, and a PQ-tree is utilized to guide the allowed operations and to compute their weights. In the first problem, Constrained TreeToString Divergence (CTTSD), we define the basic structure-informed rearrangement based divergence measure. Here, we assume that the gene order members of the gene cluster from which the PQ-tree is constructed are permutations. The PQ-tree representing the gene cluster is ordered such that the series of gene IDs spelled by its leaves is equivalent to the reference gene order. Then, a structure-informed gene rearrangement measure is computed between the ordered PQ-tree and the target gene order. The second problem, TreeToString Divergence (TTSD), generalizes CTTSD, where the gene order members are not necessarily permutations and the structure-informed rearrangement based divergence measure is extended to also consider up to d_S and d_T gene insertion and deletion operations, respectively, when modelling the PQ-tree informed divergence process from the reference order to the target order.

The first algorithm solves CTTSD in $O(n\gamma^2 \cdot (m_p \cdot 1.381^\gamma + m_q))$ time and $O(n^2)$ space, where γ is the maximum number of children of a node, n is the length of the string and the number of leaves in the tree, and m_p and m_q are the number of P-nodes and Q-nodes in the tree, respectively. If one of the penalties of CTTSD is 0, then the algorithm runs in $O(nm\gamma^2)$ time and $O(n^2)$ space. The second algorithm solves TTSD in $O(n^2\gamma^2{d_T}^2{d_S}^2m^2(m_p \cdot 5^\gamma\gamma + m_q))$ time and $O(d_Td_Sm(mn + 5^\gamma))$ space, where γ is the maximum number of children of a node, n is the length of the string, m is the number of leaves in the tree, m_p and m_q are the number of P-nodes and Q-nodes in the tree, respectively, and allowing d_T deletions from the tree and d_S deletions from the string. The third algorithm is intended to reduce the space complexity of the second algorithm. It solves a variant of the problem (where one of the penalties of TTSD is 0) in $O(n\gamma^2{d_T}^2{d_S}^2m^2(m_p \cdot 4^\gamma\gamma^2n(d_T + d_S + m + n) + m_q))$ time and $O(\gamma^2nm^2d_Td_S(d_T + d_S + m + n))$ space.

The algorithm is implemented as a software tool, denoted MEM-Rearrange, and applied to the comparative and evolutionary analysis of 59 chromosomal gene clusters extracted from a dataset of 1,487 prokaryotic genomes.

2012 ACM Subject Classification Applied computing → Computational biology

Keywords and phrases PQ-tree, Gene Cluster, Breakpoint Distance

Digital Object Identifier 10.4230/LIPIcs.WABI.2022.11

Supplementary Material The code for our tool, the data used in experiments, and the log file produced by the run of the reported benchmark, can be found on GitHub.
Software (Source Code and Data): `https://github.com/edenozery/MEM-Rearrange`

Funding The research was supported by the Israel Science Foundation (ISF) grants no. 939/18 and 1176/18, and by the Frankel Center for Computer Science at Ben Gurion University.

Acknowledgements Our sincere thanks go to the anonymous WABI 2022 referees who, with their careful reading and incisive comments, helped improve this paper.

1 Introduction

Recent advances in pyrosequencing techniques, combined with global efforts to study infectious diseases, yield huge and rapidly-growing databases of microbial genomes [32, 34]. These big new data statistically empower genomic-context based approaches to functional and evolutionary analysis: the biological principle underlying such analyses is that groups of genes that are located close to each other across many genomes often code for proteins that interact with one another, suggesting a common functional association.

Groups of genes that are co-locally conserved across many genomes are denoted *gene clusters*. The order of the genes in distinct genomic occurrences of a gene cluster may not be conserved. A specific order of the genes of a gene cluster, that is co-linearly conserved across many genomes, is denoted a *gene order* of the gene cluster. The distinct genomes in which a gene order occurs are denoted *instances* of the gene order. Gene clusters in prokaryotic genomes often correspond to (one or several) operons; those are neighbouring genes that constitute a single unit of transcription and translation.

In this paper, our biological goal is to study the evolution of gene clusters in prokaryotes, by computing the divergence between pairs of gene orders that belong to the same gene cluster, based on genome rearrangement scenarios. When defining this computational task as an optimization problem, one needs to take into account that parsimony considerations may not be sufficient: driven by the objective to keep the genome small and efficient, in spite of the high rate of gene shuffling in the prokaryotic genome, only gene orders that are reinforced by conveying some advantage in adaptation will be kept in the genome. This calls for a structure-informed genome rearrangement divergence measure that will interleave parsimony considerations with some learned structural and functional information regarding the gene cluster under study. Such a measure could more accurately assess the degree of divergence from one order of a gene cluster to another, and provide further understanding of gene-context level environmental-specific adaptations [20, 1].

To this end, we propose a new structure-informed rearrangement-based divergence measure and provide efficient algorithms to compute it. According to our approach, information regarding the structure of the gene cluster is learned from the known gene orders of the gene cluster and represented by a PQ-tree (defined in Section 2). The PQ-tree is then utilized to both guide the allowed operations and to compute their weights.

PQ-trees have been advocated as a representation for gene clusters [10, 6, 38]. A PQ-tree describes the possible permutations of a given sequence, and can be constructed in polynomial-time [25]. The PQ-tree representing a given gene cluster describes its hierarchical inner structure and the relations between instances of the gene cluster succinctly, assists in predicting the functional association between the genes in the gene cluster, yields insights into the evolutionary history of the gene cluster, and provides a natural and meaningful way of visualizing complex gene clusters. We refer the reader to Figure 6 (Section 5.2) for two exemplifications of gene clusters and their representative PQ-trees. The figure also illustrates a couple of gene orders per each exemplified gene cluster.

The biological assumptions underlying the representation of gene clusters as PQ-trees is that operons evolve via progressive merging of sub-operons, where the most basic units in this recursive operon assembly are colinearly conserved sub-operons [19]. In the case where an operon is assembled from sub-operons that are colinearly dependent, the conserved gene order could correspond, e.g., to the order in which the transcripts of these genes interact in the metabolic pathway in which they are functionally associated [35]. Thus, rearrangement events that shuffle the order of the genes (or of smaller sub-operons) within this sub-operon could affect the function of its product. On the other hand, inversion events in which the genes participating in this sub-operon remain colinearly ordered with respect to the transcription order, have less of an affect on the interactions between the sub-operon's gene products.

The case of colinearly conserved sub-operons is represented in the PQ-tree by a Q-node (marked with a rectangle), and by a *Reversal* operation in the corresponding pairwise gene order rearrangement scenario. In the case where an operon is assembled from sub-operons that are not colinearly co-dependent, convergent evolution could yield various orders of the assembled components [19]. This case is represented in the PQ-tree by a P-node (marked with a circle), and by a *Block Interchange* operation in the corresponding pairwise gene order rearrangement scenario.

Background on Structure Informed Rearrangement Scenarios. A generic formulation of genome rearrangement problems is, given two genomes and some allowed edit operations, to transform one genome into the other using a minimum number of edit operations [17, 11, 14, 26]. A famous algorithmic result related to genome rearrangements concerns the problem of sorting signed permutations by Reversals. This problem aims at computing a shortest sequence of Reversals that transforms one signed permutation into another, and can be solved in polynomial time [22, 7, 30]. It was later generalized to handle, still in polynomial time, multichromosomal genomes with linear chromosomes, using rearrangements such as Translocations, chromosome Fusions and Fissions [21, 24]. Then, a general operation called Double Cut-and-Join (DCJ), was introduced in [36] for handling problem instances where the common intervals are organized in a nonlinear structure. A DCJ can be, among others, a Reversal, a Translocation, a Fusion or a Fission, but two consecutive DCJ operations can also simulate a Block-Interchange or a Transposition.

Previous works proposed related forms of structure-informed genome rearrangements by considering rearrangement scenarios on two permutations that preserve their common intervals (groups of co-localized genes). Such scenarios, which may not be shortest among all scenarios, are called perfect [18]. Computing a Reversal scenario of minimum length that preserves a given subset of the common intervals of two signed permutations is NP-hard [18] and several papers have explored this problem, describing families of instances that can be solved in polynomial time [2, 3, 28, 16], fixed parameter tractable algorithms [3, 5], heuristics [8], and an exponential time algorithm (which works also in the more general weighted case) [23]. We note that all the perfect scenarios mentioned above considered only Reversal operations, while for our settings Block-Interchange operations should also be considered.

In [4] the notion of perfect scenario was extended to the Perfect DCJ model, thus capturing additional operations in perfect scenarios, including Cut, Join and Block-Interchange. When considering the perfect rearrangement scenarios that best fit our problem, this is the model that is most relevant to our settings, as the other previous works do not include the Block-Interchange operation. The operations considered by the Perfect DCJ model are very general, which renders the problem computationally intractable in its general setting. Indeed, Berard et al. [4] thus only obtain positive algorithmic results for special cases that, in particular, do not encompass the structure of a PQ-tree, and with a parameter that can often be of the magnitude of the entire input size. For us, the aforementioned special cases are too restricted, and cannot model the problem we have at hand. On the other hand, fortunately, for us, considering Cut and Join operations is an unjustified overhead. Specifically, we seek to model the considered evolutionary scenarios by a formulation that is more specific to our biological problem, in order to increase the divergence measure accuracy as well as tighten the parameters driving the complexity of the algorithms for the problem. Since, in our problem, we are dealing with prokaryotic gene clusters and the data in our experiment is typically confined to one chromosome per genome, we need not consider Cut and Join operations. In addition, the intervals in prokaryotic gene clusters follow a strongly conserved hierarchy, naturally modeled by the PQ-tree learned from the members of the gene cluster. In terms of divergence measure accuracy, we would like to enforce the PQ-tree structure as a constraint to the considered rearrangements. Furthermore, while the Perfect DCJ model is unweighted (and simply counts the number of DCJ operations applied), we use the PQ-tree as a guide affecting the weights of the applied rearrangement operations. In terms of tightening the parameters driving the complexity of the computation, the PQ-tree constraint enables us to use dynamic programming algorithms and to reduce the parameter from n to the out degree of the tree. In particular, this means that the more hierarchical the input is, the smaller our parameter is likely to be, and the faster our algorithm is – in other words, the running time of our algorithm naturally scales with the amount of structure given by the PQ-tree.

Our Contribution. We propose a new, two-step approach to structure-informed gene order rearrangement: In the first step, given the gene orders of the gene cluster under study, the internal topology properties of a gene cluster are learned from its corresponding gene orders and a PQ-tree is constructed accordingly. Then, in the second step, given a reference gene order and a target gene order, a structure informed rearrangement is computed from the reference to the target, such that colinear dependencies among genes and between sub-operons, as learned by the PQ-tree, are taken into account by the penalties assigned to the rearrangement operations.

To this end, we define two new theoretical problems and propose three algorithms to solve them. In the first problem, denoted Constrained TreeToString Divergence (CTTSD), we define the basic structure-informed rearrangement based divergence measure. Here, we assume that the gene order members of the gene cluster from which the PQ-tree is constructed are permutations. The rearrangement operations considered by this problem include (weighted) Reversals and Block-Interchange operations. In this problem, the PQ-tree representing the gene cluster (Figure 6.A) is ordered such that the series of gene IDs spelled by its leaves is equivalent to the reference gene order (Figure 6.B). Then, a structure-informed, weighted genome rearrangement measure is computed from the ordered PQ-tree to the target gene order (Figure 6.C).

The second problem, denoted TreeToString Divergence (TTSD), is a generalization of the first problem, where the gene order members are not necessarily permutations and the genome rearrangement measure is extended to also consider up to d_S gene insertion operations and up to d_T gene deletion operations.

The first fixed parameter tractable (FPT) algorithm (in Section 3) solves CTTSD in $O(n\gamma^2 \cdot (m_p \cdot 1.381^\gamma + m_q))$ time and $O(n^2)$ space, where γ is the maximum number of children of a node, n is the length of the string and the number of leaves in the tree, and m_p and m_q are the number of P-nodes and Q-nodes in the tree, respectively. If one of the penalties of CTTSD is defined to be 0, then the algorithm runs in $O(nm\gamma^2)$ time and $O(n^2)$ space.

The second FPT algorithm (in the full version) solves TTSD in $O(n^2\gamma^2 {d_T}^2 {d_S}^2 m^2 (m_p \cdot 5^\gamma \gamma + m_q))$ time and $O(d_T d_S m(mn + 5^\gamma))$ space, where γ is the maximum number of children of a node, n is the length of the string, m is the number of leaves in the tree, m_p and m_q are the number of P-nodes and Q-nodes in the tree, respectively, and allowing d_T deletions from the tree and d_S deletions from the string.

While our first algorithm is simple and intuitive (based on one dynamic programming and two greedy procedures), for our second algorithm (based on three dynamic programming procedures), more technical ingredients are required. For example, one challenge is the need to compute a vertex cover in a graph that is not fully known by any single entry of our dynamic programming table. Specifically, when we consider a single entry, some of the relevant vertices are not yet processed, and for those that are processed, we cannot store enough information (for the sake of efficiency) so as to deduce which edges exist between them.

The third FPT algorithm (in the full version) is intended to reduce the space complexity of the second algorithm. It solves a variant of the problem (where one of the penalties of TTSD is defined to be 0) in $O(n\gamma^2{d_T}^2{d_S}^2m^2(m_p \cdot 4^\gamma\gamma^2n(d_T + d_S + m + n) + m_q))$ time and $O(\gamma^2nm^2d_Td_S(d_T + d_S + m + n))$ space. This algorithm employs the principle of inclusion-exclusion for the sake of space reduction, which, to the best of our knowledge, is not commonly used in the study of problems in computational biology.

The proposed algorithms are implemented as a software tool, denoted MEM-Rearrange, and applied to the comparative and evolutionary analysis of 59 chromosomal gene clusters extracted from a dataset of 1,487 prokaryotic genomes (in Section 5). Our preliminary results, based on competitive analysis of the 59 gene clusters, indicate that our proposed measure correlates well with an index that is computed by comparing the class composition of the genomic instances of the two compared gene orders. The correlations yielded by our measure show some advantage over the correlations computed for the basic (not structurally informed) Signed Break-point Distance [9], an advantage that grows monotonically with increase in the number of Q-nodes in the PQ-tree. Two of the gene clusters from the benchmark are used to illustrate how our approach can be applied to the study of rearrangement-based adaptations.

Roadmap. The rest of the paper is organized as follows. In Section 2, we formally define the terminology used throughout the paper, and, in particular, the two problems studied in this paper. Some of the more standard definitions are deferred to the full version of this paper. Due to space constraints, our second algorithm, which solves the TTSD problem, is given only in the full version. In Section 3, we present our first algorithm, which solves the CTTSD problem. Due to space constraints, some details – in particular, pseudocode and complexity analysis – are deferred to the full version. Our third algorithm, which reduces the space complexity of our second algorithm to be polynomial in a special case, is also deferred to the full version. In Section 4, we specify the details of our data set construction and experiment. In Section 5, we compare the performance of our proposed rearrangement measure versus that of signed break-point distance on a benchmark of 59 chromosomal gene clusters and provide a detailed exemplification of the difference between the two rearrangement measures on two of the gene clusters from the benchmark.

2 Preliminaries

Let $S = s_1...s_n$ be a string, $S[i] = s_i$, and $S[i : j] = s_i...s_j$.

PQ-Tree: Representing the Pattern. A PQ-tree is a rooted tree with three types of nodes: *P-nodes*, *Q-nodes* and leaves. The children of a P-node can appear in any order, while the children of a Q-node must appear in either left-to-right order or right-to-left order. The possible reordering of the children nodes in a PQ-tree can potentially create many equivalent PQ-trees. Booth and Lueker [10] defined two PQ-trees T and T' as *equivalent* (denoted $T \equiv T'$) if and only if one tree can be obtained by legally reordering the nodes of the other, as described above. A generalization of their definition, to allow insertions and deletions, is defined as follows. Two PQ-trees T, T' are *quasi-equivalent* with parameter d, denoted by $T \cong_d T'$, if and only if T' can be obtained by (a) randomly permuting the children of the P-nodes of T, (b) reversing the children of the Q-nodes of T, (c) deleting up to d leaves of T. Denote by T_x the subtree of a PQ-tree T rooted in the node x. Denote by $\mathsf{Leaves}(x)$ the set of leaves of T_x, $\mathsf{span}(x) = |\mathsf{Leaves}(x)|$, and for a set of nodes U, $\mathsf{span}(U) = \sum_{v \in U} \mathsf{span}(v)$. Denote by $\mathsf{children}(x)$ the set of children of x, and let $root_T$ be the root node of T.

Given a PQ-tree T, we denote the label of a leaf x by $\mathsf{label}(x)$. The *frontier* T, denoted $F(T)$, is the sequence of labels on the leaves of T read from left to right. In addition, for each node in a PQ-tree (internal node or leaf), we define a unique "color", which will help us distinguish and map between nodes of two quasi-equivalent PQ-trees. Specifically, we use colors to keep track of which operations are performed on a tree when reordering it. From now on, when we say that two PQ-trees T, T' are quasi-equivalent, we assume that the equivalence is with parameter d. In addition, we assume that the PQ-trees T, T' are colored as follows: each color is assigned to one node in T and at most one node in T', and the nodes in T' have the same colors as their corresponding nodes in T. In addition, we say that the *frontier* of T', $F(T')$, is *derived* from T, and we call this a *derivation.* We also say that T' is ordered as $F(T')$. When a string S is derived from T_x, we also say that S is derived from x. Given two quasi-equivalent PQ-trees T, T' and two nodes $x \in T$ and $x' \in T'$, x and x' are *equivalent nodes* if they share the same color.

Break-Point Distances. A *signed string* is a string where each character is assigned a sign ('+' or '-'). Specifically, we suppose to have a function sign that returns the sign of the character in each position in S. A *gene mapping* $\mathcal{M}$ of two strings, $G = g_1, \dots, g_n$ and $H = h_1, \dots, h_m$, is a set of pairs $(g_i, h_j) \in G \times H$ such that $g_i = h_j$ and every position in G and H is in exactly one pair in $\mathcal{M}$. Now, we can define the notion of a signed break-point:

▶ **Definition 1** (Signed Break-Point). *Given two signed strings $G = g_1, \dots, g_n$, $H = h_1, \dots, h_m$ and a gene mapping $\mathcal{M}$, a* signed break-point *between G and H is a pair of consecutive genes $g_i g_{i+1}$ in G (resp. $h_j h_{j+1}$ in H) such that g_i and g_{i+1} (resp. h_i and h_{i+1}) belong to $\mathcal{M}$, say, $(g_i, h_j), (g_{i+1}, h_k) \in \mathcal{M}$ (resp. $(h_i, g_j), (h_{i+1}, g_k) \in \mathcal{M}$), but neither $k = j + 1$, $\mathsf{sign}_G(i) = \mathsf{sign}_H(j)$ and $\mathsf{sign}_G(i+1) = \mathsf{sign}_H(k)$, nor $k = j - 1$, $\mathsf{sign}_G(i) = -\mathsf{sign}_H(j)$ and $\mathsf{sign}_G(i+1) = -\mathsf{sign}_H(k)$.*

Denote by $\mathsf{NUM}_{\mathsf{SBP}}(G, H, \mathcal{M})$ the number of signed break-points of G between G and H with respect to $\mathcal{M}$. The *signed break-point distance* between G and H, denoted by $\mathsf{d}_{\mathsf{SBP}}(G, H)$, is the minimum of $\mathsf{NUM}_{\mathsf{SBP}}(G, H, \mathcal{M})$ among all gene mappings $\mathcal{M}$. For example, let $S_1 = +a + b + c + d$ and $S_2 = +a - b - d - c$. Then, there exists exactly one gene mapping of S_1 and S_2. The signed break-points of S_1 are the pairs $(a, b), (b, c)$, and $\mathsf{d}_{\mathsf{SBP}}(S_1, S_2) = 2$. To accommodate deletions, we will use the notion of signed break-point distance with respect to strings obtained from the given ones after deleting characters.

Problem Preliminaries. Given an internal node x in a PQ-tree T, let $\mathsf{sign}(x)$ be the majority sign of $\mathsf{Leaves}(x)$. If the number of negative signs is equal to the number of positive signs, we abuse notation and consider $\mathsf{sign}(x)$ as $+$ as well as $-$. Given a node x in a PQ-tree, let $S(x)$ denote the signed string of colors of the nodes in $\mathsf{children}(x)$ as they are ordered in the tree (from left to right). For example, consider Figure 1.a where letters represent colors; then, $S(z) = +b + c$, $S(y) = +a + z + d$ and $S(x) = +y + e + f$.

Towards defining the divergence from an (ordered) PQ-tree T to a string S, we define a penalty for taking an action on an internal node of a PQ-tree, denoted by $\Delta_{\mathsf{violation}}$, which is a combination of several types of penalties. The first type concerns large units that "jump" while reordering the children of a P-node x to have the same order as those of its equivalent node x'. If the size of a unit that jumps is t, then we penalize this operation by $(t-1)/2$. In particular, a leaf does not get penalized. To do so, we build a graph $G[x,x']$, whose vertex set is the children of x. Each $c \in \mathsf{children}(x)$ has a weight of $(\mathsf{span}(c)-1)/2$. This weight represents the penalty for a jump of a large unit; we divide by 2 so that this penalty, even when the unit size is 2, will be smaller than the penalty for a break-point. $G[x,x']$ has an edge for each pair of children that *change their signed order*, defined as follows.

▶ **Definition 2** (Change Of Signed Order). *Let x, x' be two equivalent P-nodes of two quasi-equivalent PQ-trees T and T' with parameter d, $S(x) = c_1, \dots, c_n$ be the signed string of colors of the nodes in $\mathsf{children}(x)$ that were not deleted, as they ordered in T, $S(x') = c'_1, \dots, c'_{n'}$ be the signed string of colors of the nodes in $\mathsf{children}(x')$, as they ordered in T', and $\mathcal{M}$ be the gene mapping of $S(x)$ and $S(x')$. Given two nodes y (with color c_i and y' is the equivalent node of y in T') and z (with color c_j and z' is the equivalent node of z in T') in $\mathsf{children}(x)$, say, $j > i$, and such that $(c_i, c'_k), (c_j, c'_t) \in \mathcal{M}$ for some c'_k and c'_t, y and z* change their signed order *if the following are both false: (1) $t > k$, $\mathsf{sign}(y) = \mathsf{sign}(y')$ and $\mathsf{sign}(z) = \mathsf{sign}(z')$ (2) $t < k$, $\mathsf{sign}(y) = -\mathsf{sign}(y')$ and $\mathsf{sign}(z) = -\mathsf{sign}(z')$.*

Let $G = (V, E)$ be a graph with a vertex weight function w. The *minimum weight of a vertex cover* of G is the minimum weight among all vertex covers of G (where a *vertex cover* is a set $S \subseteq V$ such that every edge of G has at least one endpoint in S, and its weight is the sum of weights of its vertices). After building $G[x,x']$, we find the minimum weight of a vertex cover of $G[x,x']$ in order to sum all penalties for the units that jumped while reordering the children of a P-node. Observe that by computing the minimum weight of a vertex cover, we identify a "best" (in terms of penalty) set of nodes that jump. The vertices of the vertex cover itself are considered to be the units that jump.

▶ **Definition 3** ($\Delta^P_{\mathsf{jump}}(x,x')$). *Given two equivalent P-nodes x and x' of two quasi-equivalent PQ-trees with parameter d, and the weight t of a minimum weighted vertex cover of $G[x,x']$, the* jump violation *between x and x', denoted by $\Delta^P_{\mathsf{jump}}(x,x')$, is t.*

Given two equivalent nodes x, x' of two quasi-equivalent PQ-trees T and T' (where x is not deleted), let $\mathsf{isFlipped}(x,x')$ return 1 if x and x' "flipped", and 0 otherwise. That is, if x and x' are leaves, $\mathsf{isFlipped}(x,x') = 1$ if $\mathsf{sign}(x) = -\mathsf{sign}(x')$; otherwise, $\mathsf{isFlipped}(x,x') = 0$. If x and x' are internal nodes, $\mathsf{isFlipped}(x,x') = 1$ if for each child $y \in \mathsf{children}(x)$ that is not deleted, $\mathsf{isFlipped}(y,y') = 1$ where y' is the child of x' equivalent to y, and the order of $\mathsf{children}(x')$ in T' is the reversal of the order of $\mathsf{children}(x)$ in T; otherwise, $\mathsf{isFlipped}(x,x') = 0$. Given an internal node x, denote by $\tilde{\mathcal{S}}_d(x)$ the set of signed strings where $\tilde{S} \in \tilde{\mathcal{S}}_d(x)$ if $\tilde{S}$ is obtained from $S(x)$ by deleting up to d leaves from T_x. Now, we are ready to define the penalty for violation between x and x':

▶ **Definition 4** ($\Delta^d_{\mathsf{violation}}(x,x')$). *Given two equivalent internal nodes x and x' of two quasi-equivalent PQ-trees with parameter d, and input numbers δ^Q_{ord} and δ^Q_{flip}, the* violation *between x and x', denoted by $\Delta^d_{\mathsf{violation}}(x,x')$, is defined as follows.*

- *If x is a P-node, $\Delta^{\mathsf{P}}_{\mathsf{violation}}(x,x') = \min\limits_{\tilde{S} \in \tilde{\mathcal{S}}_d(x)} \mathsf{d_{SBP}}(\tilde{S}, S(x')) + \Delta^P_{\mathsf{jump}}(x,x')$.*
- *If x is a Q-node, $\Delta^{\mathsf{Q}}_{\mathsf{violation}}(x,x') = \delta^Q_{\mathsf{ord}} \cdot \min\limits_{\tilde{S} \in \tilde{\mathcal{S}}_d(x)} \mathsf{d_{SBP}}(\tilde{S}, S(x')) + \mathsf{isFlipped}(x,x') \cdot \delta^Q_{\mathsf{flip}}$.*

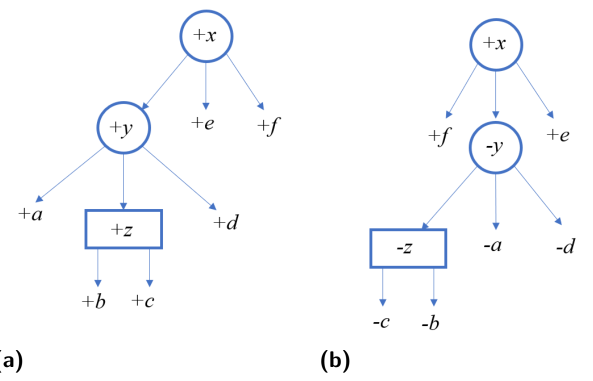

Figure 1 (a) A PQ-tree T. (b) A PQ-tree that is equivalent to T and is ordered as S. Note that a P-node is denoted by a circle and a Q-node is denoted by a rectangle.

When d is clear from the context, we will use the notation $\Delta_{\mathsf{violation}}$ instead. In Definition 4, one of the penalties for reordering the children of a Q-node x is the flip penalty ($\delta^{Q}_{\mathsf{flip}}$), and it is performed if x flipped. Here, notice the case where x flipped and all of its non-deleted children flipped as well (including the P-nodes). Then, we penalise each of the non-deleted children and x as well for flipping, but, the event is flipping only x. Thus, we unnecessarily penalise the non-deleted children. Therefore, we define a "flip correction" procedure:

▶ **Definition 5** (FlipCorrection(x, x')). *Given two equivalent internal nodes x and x' of two quasi-equivalent PQ-trees with parameter d, the* flip correction *between them, denoted by* FlipCorrection(x, x'), *is 0 if not all of* children(x) *flipped or deleted, and otherwise it is the sum of the flip penalties of all of the Q-nodes and leaves in* children(x) *that were not deleted.*

Finally, we sum up the violations (and corrections) of all internal nodes to define the divergence from an ordered and labeled PQ-tree to a signed string:

▶ **Definition 6** ($\mathsf{mDiverg}_{d_T,d_S}(T, S)$). *Let T be an (ordered and leaf-labeled) PQ-tree, and S be a signed string. Let $\mathcal{O}_T$ be the set of all quasi-equivalent T' PQ-trees of T with parameter d_T (i.e. $T \cong_{d_T} T'$) such that T' is ordered as a substring of S obtained by deleting up to d_S characters from S. For any $T' \in \mathcal{O}_T$, let $\mathcal{M}_{T'}$ be the unique*[1] *mapping of each node in T' to its equivalent node in T. Then,*

$$\mathsf{mDiverg}_{d_T,d_S}(T, S) = \min_{T' \in \mathcal{O}_T} \sum_{(x,x') \in \mathcal{M}_{T'}} (\Delta_{\mathsf{violation}}(x, x') - \mathsf{FlipCorrection}(x, x')).$$

In case $d_T = 0$ and $d_S = 0$, let $\mathsf{mDiverg}(T, S) = \mathsf{mDiverg}_{d_T,d_S}(T, S)$.

For example, consider the PQ-tree T in Figure 1, which is ordered as $+a+b+c+d+e+f$, and the signed string $S = +f - c - b - a - d + e$. To order T as S, flip the Q-node z, and pay $\Delta^{\mathsf{Q}}_{\mathsf{violation}}(z, z') = \delta^{Q}_{\mathsf{flip}}$. For the P-node y, swap between a and z, so $\mathsf{d_{SBP}}(S(y), S(y')) = 1$; hence, $\Delta^{\mathsf{P}}_{\mathsf{violation}}(y, y') = 1$ (there is no jump of a large unit, because the leaf a can jump). Finally, for the P-node x, $\Delta^{P}_{\mathsf{jump}}(x, x') = 0$ (order the children by moving the leaf f to the left-most position), and $\mathsf{d_{SBP}}(S(x), S(x')) = 1$. Thus, $\Delta^{\mathsf{P}}_{\mathsf{violation}}(x, x') = 1$. In total, $\mathsf{mDiverg}(T, S) = \Delta^{\mathsf{Q}}_{\mathsf{violation}}(z, z') + \Delta^{\mathsf{P}}_{\mathsf{violation}}(y, y') + \Delta^{\mathsf{P}}_{\mathsf{violation}}(x, x') = \delta^{Q}_{\mathsf{flip}} + 2$.

[1] Note that uniqueness follows from our use of colors.

Problem Definitions

Constrained TreeToString Divergence. The input to CTTSD consists of two signed permutations of length n, $S_1 = \sigma_1 \dots \sigma_n \in \Sigma^n$, $|\Sigma| = n$, such that $\sigma_i \neq \sigma_j$ for all $1 \leq i < j \leq n$, and $S_2 = \lambda_1 \dots \lambda_n \in \Sigma^n$ such that $\lambda_i \neq \lambda_j$ for all $1 \leq i < j \leq n$; a PQ-tree T ordered as S_1 with m_p P-nodes and m_q Q-nodes; and two numbers δ^Q_{ord} and δ^Q_{flip}. The output is $\mathsf{m_{Diverg}}(T, S_2)$ or "NO" if S_2 cannot be derived from T.

TreeToString Divergence. Generalizing CTTSD, in TTSD we do not assume that the strings are permutations, and we allow deletions. The input to TTSD consists of two signed strings, $S_1 = \sigma_1 \dots \sigma_m \in \Sigma^m_T$ and $S_2 = \lambda_1 \dots \lambda_n \in \Sigma^n_S$; a PQ-tree T ordered as S_1 with m_p P-nodes and m_q Q-nodes; $d_T \in \mathbb{N} \cup \{0\}$, which specifies the number of allowed deletions from T; $d_S \in \mathbb{N} \cup \{0\}$, which specifies the number of allowed deletions from S_2; and two numbers δ^Q_{ord}, δ^Q_{flip} indicating the penalty of the events of changing order and flipping, respectively, a Q-node. The output is $\mathsf{m_{Diverg}}_{d_T,d_S}(T, S_2)$ or "NO" if S_2 cannot be derived from T with respect to d_T and d_S.

3 Constrained TreeToString Divergence Algorithm

In this section, we present a greedy FPT algorithm to solve CTTSD. Our algorithm consists of three components: the main algorithm, and two procedures called P-Mapping and Q-Mapping. We first present and explain the main algorithm and the procedures. Afterwards, we demonstrate the execution of the algorithm. We remark that, since we consider a string and its reversal (with all signs negated) to be equal, we run the algorithm twice: once on the input string, and once on its reversal.

The Main Algorithm. Recall that the input to CTTSD consists of two signed permutations S_1 and S_2 of length n, two numbers δ^Q_{ord} and δ^Q_{flip}, and a PQ-tree T ordered as S_1. If S_2 can be derived from T, then the output of the algorithm is the divergence from T to S_2, $\mathsf{m_{Diverg}}(T, S_2)$. Otherwise, the output is "NO" (specifically, the algorithm returns ∞).

The main algorithm constructs a 2-dimensional DP table $\mathcal{A}$ of size $m' \times n$ where $m' = n + m_p + m_q$ is the number of nodes in T. For each node x in T and index ℓ, $\mathcal{A}$ has an entry $\mathcal{A}[x, \ell]$. In the algorithm, for each node x, we keep two indices ℓ and r (denoted by $x.\ell$ and $x.r$ respectively) such that $S_2[x.\ell : x.r]$ is derived from T_x. Then, the purpose of an entry of the DP table, $\mathcal{A}[x, x.\ell]$, is to hold the divergence from the subtree T_x to the substring $S_2[x.\ell : x.r]$ of S_2. That is, $\mathcal{A}[x, x.\ell] = \mathsf{m_{Diverg}}(T_x, S_2[x.\ell : x.r])$. If any substring of S_2 starting at position ℓ cannot be derived from T_x, then $\mathcal{A}[x, \ell] = \infty$.

Some entries of the DP table define illegal derivations. Such are derivations where the length of the frontier of the subtree is larger than the length of the longest subsrting starting at the specified index ℓ. These entries are called *invalid entries* and their value is defined as ∞ throughout the algorithm. Formally, an entry $\mathcal{A}[x, \ell]$ is invalid if $\mathsf{span}(x) > n - \ell + 1$.

The algorithm first initializes the entries of $\mathcal{A}$ that are meant to hold divergences of derivations of every possible substring of S_2 (a single character) from the leaves of T. Specifically, for a leaf x, if it did not flip, we put 0 in the corresponding entry. If x did flip, we put δ^Q_{flip}. After that, we update $x.\ell$ and $x.r$. As described in the initialization, if $\mathsf{label}(x) = S_2[\ell]$, $S_2[\ell]$ ($S_2[\ell : \ell]$) is derived from $\mathsf{label}(x) = S_2[\ell]$, then we put 0 or δ^Q_{flip} in $\mathcal{A}[x, \ell]$ and we put ℓ in $x.\ell$ and $x.r$.

After the initialization, all other entries of $\mathcal{A}$ are filled as follows. Go over the internal nodes of T in postorder. For every internal node x, go in descending order over every index $1 \leq \ell \leq n$ that can be a start index for the substring of S_2 derived from T_x (in case of invalid entry, we continue to the next iteration). For every x and ℓ, use the algorithm for P-mapping or Q-mapping according to the type of x. Both algorithms receive the following input: the node x, S_2, start and end indices ℓ, e of a substring of S_2, and the collection of derivations of the children of x (entries of $\mathcal{A}$ that have already been computed and hold the divergence of a derivation). In addition, the Q-Mapping algorithm receives as input the penalty parameters δ^Q_{ord} and δ^Q_{flip}. After being called, both algorithms return the divergence from T_x to $S_2[\ell : e]$, that is, $\mathsf{m}_{\mathsf{Diverg}}(T_x, S_2[\ell : e])$.

Finally, having filled the DP table, $\mathcal{A}[root_T, 1]$ holds the divergence from T to S_2 ($\mathsf{m}_{\mathsf{Diverg}}(T, S_2)$), and so we return $\mathcal{A}[root_T, 1]$.

P-Node Mapping: The Algorithm. Recall that the input consists of a P-node x, a string S_2, two indices ℓ and e, and a set of derivations $\mathcal{D}$. Notice that each value in $\mathcal{D}$ is the divergence from the subtree rooted in a child c of x to $S_2[c.\ell : c.e]$, where $S_2[c.\ell : c.e]$ is a substring of $S_2[\ell : e]$ that is derived from T_c. These values, $\mathcal{A}[c, c.\ell]$ for each $c \in \mathsf{children}(x)$, were calculated in earlier iterations and saved in $\mathcal{D}$. If $S_2[\ell : e]$ can be derived from T_x, then the output of the algorithm is the divergence from T_x to $S_2[\ell : e]$, $\mathsf{m}_{\mathsf{Diverg}}(T_x, S_2[\ell : e])$. Otherwise, the output is "NO" (specifically, the algorithm returns ∞). Denote by T'_x the quasi-equivalent PQ-tree of T_x ordered as $S_2[\ell : e]$. Note that if T'_x exists, then it is unique (because we deal with permutations and forbid deletions).

The algorithm first checks if the interval $[\ell, e]$ can be "completed" by all of the intervals defined by the indices of the children of x. Specifically, we check if there is any order of the children of x, say, ordered as $c_1, \ldots, c_{|\mathsf{children}(x)|} \in \mathsf{children}(x)$, such that $c_1.\ell = \ell$, $c_{|\mathsf{children}(x)|}.e = e$, and for each $1 \leq j \leq |\mathsf{children}(x)| - 1$, $c_j.e + 1 = c_{j+1}.\ell$. If there is no such order, then the interval $[\ell, e]$ cannot be completed, and so $S_2[\ell, e]$ cannot be derived from T_x. In this case, we return ∞. Otherwise, $S_2[\ell : e]$ can be derived from T_x by reordering the children of x according to the unique order that completes the interval $[\ell, e]$. Second, we sum up all of the values in $\mathcal{D}$ (and store the sum in the variable *childrenDist*). Next, we calculate the violation between x and its equivalent node x' in T'_x, $\Delta^{\mathsf{P}}_{\mathsf{violation}}(x, x')$, according to Definition 4 (and store the result in the variable *violation*). Finally, we return the sum of *childrenDist* and *violation*, which is the divergence from T_x to $S_2[\ell : e]$, $\mathsf{m}_{\mathsf{Diverg}}(x, S_2[\ell : e])$ (according to Definition 6).

Q-Node Mapping: The Algorithm. Recall that the input consists of a Q-node x, a string S_2, two indices ℓ and e, a set of derivations $\mathcal{D}$ and penalty parameters δ^Q_{ord} and δ^Q_{flip}. Notice that each value in $\mathcal{D}$ is the divergence from the subtree rooted in a child c of x to $S_2[c.\ell : c.e]$, where $S_2[c.\ell : c.e]$ is a substring of $S_2[\ell : e]$ and is derived from T_c. These values, $\mathcal{A}[c, c.\ell]$ for each $c \in \mathsf{children}(x)$, were calculated in earlier iterations and saved in $\mathcal{D}$. If $S_2[\ell : e]$ can be derived from T_x, then the output of the algorithm is the divergence from T_x to $S_2[\ell : e]$, $\mathsf{m}_{\mathsf{Diverg}}(T_x, S_2[\ell : e])$. Otherwise, the output is "NO" (specifically, the algorithm returns ∞). Denote by T'_x the (unique) quasi-equivalent PQ-tree of T_x ordered as $S_2[\ell : e]$.

The algorithm first checks if the interval $[\ell, e]$ can be "completed consecutively" by all of the intervals defined by the indices of the children of x. Specifically, we check if there is a consecutive order of the children of x, say, ordered as $c_1, \ldots, c_{|\mathsf{children}(x)|} \in \mathsf{children}(x)$, such that $c_1.\ell = \ell$, $c_{|\mathsf{children}(x)|}.e = e$, and for each $1 \leq j \leq |\mathsf{children}(x)| - 1$, $c_j.e + 1 = c_{j+1}.\ell$. As apposed to a P-node, here the order of the children completing the interval $[\ell, e]$ must be consecutive with respect to their indices (the same order as the children of x in T or the reverse order). If there is no such order, then the interval $[\ell, e]$ cannot be completed consecutively, and so $S_2[\ell : e]$ cannot be derived from T_x. In this case, we return ∞. Otherwise, $S_2[\ell : e]$ can be derived from T_x by keeping the order of the children of x, or flipping it. Second, we sum up all of the values in $\mathcal{D}$ (and store the sum in the variable *childrenDist*). Next, we calculate the violation between x and its equivalent node x' in T'_x, $\Delta^{\mathsf{Q}}_{\mathsf{violation}}(x, x')$, according to Definition 4 (and store the result in the variable *violation*). Afterwards, we calculate the flip correction according to Definition 5 (and store the result in the variable *childrenFlipCorrection*). Finally, we return the sum of *childrenDist* and *violation* minus *childrenFlipCorrection*, which is the divergence from T_x to $S_2[\ell : e]$, $\mathsf{m}_{\mathsf{Diverg}}(x, S_2[\ell : e])$ (according to Definition 6).

Example. Consider the following input: $S_1 = +a+b+c+d+e+f$, $S_2 = +f-c-b-a-d-e$, PQ-tree T ordered as S_1, $\delta^Q_{\mathsf{ord}} = 3$ and $\delta^Q_{\mathsf{flip}} = 3$. We iterate through the nodes of the tree in post-order, thus we initiate the leaves before their parents. For each leaf $x \in \mathsf{Leaves}(root_T)$, if $\mathsf{label}(x) = S_2[\ell]$, then $\mathcal{A}(x, \ell) = 0$; otherwise, $\mathcal{A}(x, \ell) = \infty$.

Figure 2 describes the PQ-tree T and its quasi-equivalent PQ-tree T' ordered as S_2, after the initialization of the leaves only. Notice that the order of the initialization is in fact different, in postorder; for simplicity, we show the tree where only the leaves are initialized. In addition, in the figures, the equivalent nodes of T and T' are shown as the same nodes. But in the explanations, in order to distinguish between them, for each node $x \in T'$, we denote it by x'. The pair of numbers shown in the figure near a node represent its ℓ and r values. In addition, the sign $+$ or $-$ near a node represents its sign. For example, for $a \in T$, $a.\ell = a.r = 4$ and $\mathsf{sign}(a) = +$. For each internal node, the character assigned to it represents its color.

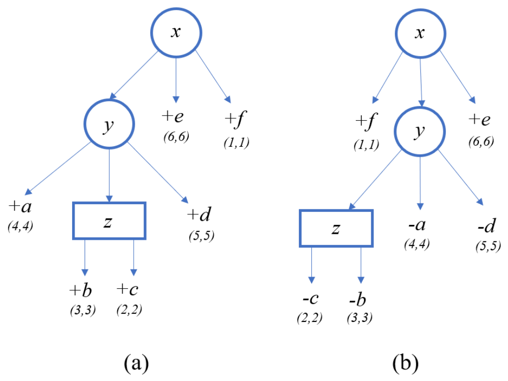

Figure 2 (a) PQ-tree T. **(b)** PQ-tree T', which is quasi-equivalent to T and ordered as S_2.

First, consider the iteration where $\mathcal{A}[z, 2]$ is calculated (the values for all other entries with node z is ∞). The intervals of the children of z, $[3, 3]$ and $[2, 2]$, complete the interval $(2, e)$ consecutively where $e = 2+\mathsf{span}(z)-1 = 3$. Thus, the substring $S_2[2 : 3]$ is derived from T_z, and in order to generate it we need to flip the order of $\mathsf{children}(z)$. For this the penalty of flipping, δ^Q_{flip}, is applied, and so $\Delta^{\mathsf{Q}}_{\mathsf{violation}}(z, z') = \delta^Q_{\mathsf{flip}} = 3$. $childrenDist = \mathcal{A}[b, 3]+\mathcal{A}[c, 2] = 6$ because $\mathcal{A}[b, 3] = \mathcal{A}[c, 2] = 3$. $\mathsf{FlipCorrection}(z, z') = 6$ because both b and c have been flipped. Therefore, $\mathcal{A}[z, 2] = childrenDist+violation-childrenFlipCorrection = 3$. Now, we update z's indices, $z.\ell = 2$ and $z.r = 3$. Figure 3 describes the PQ-tree T and its quasi-equivalent PQ-tree T' after calculating $\mathcal{A}[z, 2]$.

Next, consider the iteration where $\mathcal{A}[y, 2]$ is calculated (the values for all other entries with node y is ∞). Recall that Figure 3 describes T and T' after calculating the entries of the children of the P-node y. The intervals of the children complete the interval $(2, e)$,

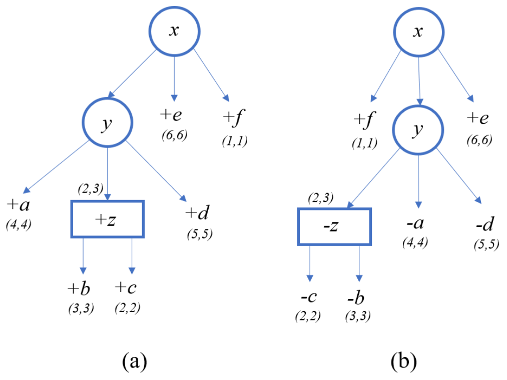

Figure 3 **(a)** PQ-tree T. **(b)** PQ-tree T', which is quasi-equivalent to T and ordered as S_2.

where $e = 2 + \mathsf{span}(y) - 1 = 5$, so we can continue with the iteration. $childrenDist = \mathcal{A}[a, 4] + \mathcal{A}[z, 2] + \mathcal{A}[d, 5] = 9$. $\mathsf{d}_{\mathsf{SBP}}(S(y), S(y')) = 1$ where the pair (z, d) is the only singed break-point (note that $S(y) = +a + z + d$, $S(y') = -z - a - d$). $\Delta^P_{\mathsf{jump}}(y, y') = 0$ because to reorder the children of y as $S_2[2 : 5]$ we can move z only (and its size is 1). Therefore, $\Delta^{\mathsf{P}}_{\mathsf{violation}}(y, y') = \mathsf{d}_{\mathsf{SBP}}(S(y), S(y')) + \Delta^P_{\mathsf{jump}}(y, y') = 1$. Then, $\mathcal{A}[y, 2] = 10$ and we can update y's indices, $y.\ell = 2$ and $y.r = 5$. Figure 4 describes the PQ-tree T and its quasi-equivalent PQ-tree T' after calculating $\mathcal{A}[y, 2]$.

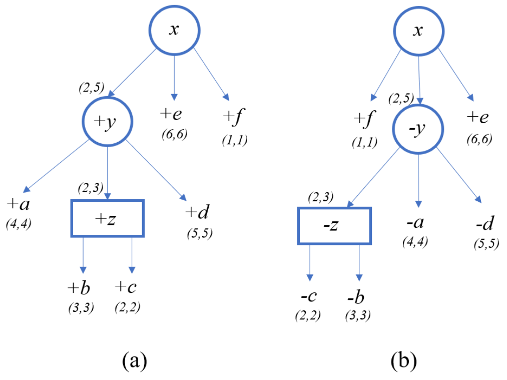

Figure 4 **(a)** PQ-tree T. **(b)** PQ-tree T', which is quasi-equivalent to T and ordered as S_2.

Finally, consider the iteration where $\mathcal{A}[x, 1]$ is calculated (the values for all other entries with node x is ∞). Figure 4 describes T and T' after calculating the entries of the children of the P-node x. The intervals of the children complete the interval $(1, e)$, where $e = 1 + \mathsf{span}(x) - 1 = 6$, so we can continue with the iteration. $childrenDist = \mathcal{A}[y, 2] + \mathcal{A}[e, 6] + \mathcal{A}[f, 1] = 10$. $\mathsf{d}_{\mathsf{SBP}}(S(x), S(x')) = 2$ where the pairs (y, e) and (e, f) are the signed break-points (note that $S(x) = +y + e + f$, $S(x') = +f - y + e$). $\Delta^P_{\mathsf{jump}}(x, x') = 0$ because to reorder the children of x as S_2 we can move f only (and its size is 1). Therefore, $\Delta^{\mathsf{P}}_{\mathsf{violation}}(x, x') = \mathsf{d}_{\mathsf{SBP}}(S(x), S(x')) + \Delta^P_{\mathsf{jump}}(x, x') = 2$. Then, $\mathcal{A}[x, 1] = 10 + 2 = 12$ and the algorithm returns 12. Figure 5 describes the PQ-tree T and its quasi-equivalent PQ-tree T' after calculating $\mathcal{A}[x, 1]$.

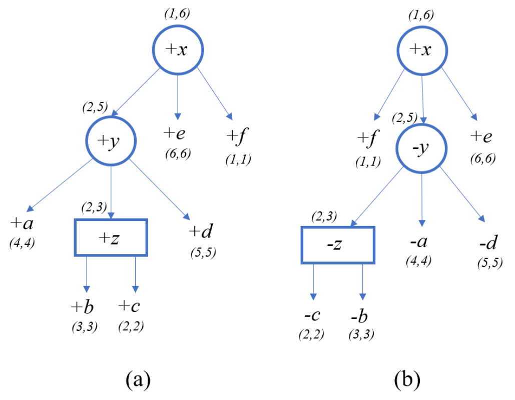

Figure 5 **(a)** PQ-tree T. **(b)** PQ-tree T', which is quasi-equivalent to T and ordered as S_2.

4 Methods and Datasets

Dataset and Gene Cluster Generation. $1,487$ fully sequenced prokaryotic strains with COG ID annotations were downloaded from GenBank (NCBI; ver 10/2012). The gene clusters were generated from this data using the tool CSBFinder-S [29]. CSBFinder-S was applied to all the genomes in the dataset after removing their plasmids, using parameters $q = 10$ (a colinear gene cluster is required to appear in at least ten genomes) and $k = 0$ (no insertions are allowed in a colinear gene cluster), resulting in 79,017 colinear gene orders. From these gene orders, only gene orders whose number of distinct COGs is between 4 and 9 were kept, leaving 28,537 gene orders. Next, ignoring strand and gene order information, colinear gene orders that contain the exact same COGs were united to form the generalized set of 91 gene clusters that abide by the requirements that each gene cluster contains at least 3 gene orders and each COG appears only once in each gene order. For each gene cluster, the most abundant gene order was designated as the "reference" (centroid) gene order. Based on this, the clusters were further filtered to keep only 63 gene clusters whose designated reference has instances in at least 30 genomes. Finally, clusters containing one or more gene orders that are identical to the designated reference gene order, in terms of the list of classes in which they have instances, were removed, leaving a benchmark set of 59 gene clusters.

PQ-tree construction. The input PQ-trees for our algorithm where constructed using the tool PQFinder (available on GitHub [37]). PQFinder was applied to each of the gene clusters in the dataset, to build the PQ-tree representing each cluster. In addition, each Q-node with exactly two children, whose height in the tree is greater than 1, was changed to a P-node (in this special case all children of the node were observed in all shuffling options, which in our opinion better fits the syntax of a P-node than that of a Q-node.)

Parameter Settings. In our experiment, we set the parameters of the algorithm as follows. $\delta^Q_{\text{ord}} = 1.5$, $\delta^Q_{\text{flip}} = 0.5$, $d_T = 0$, $d_S = 0$.

5 Results

5.1 Evaluation

In this section we evaluate the accuracy of our approach in measuring the evolutionary divergence between two gene orders that belong to the same gene cluster. To this end, we aim to generate a set of "control" distances, computed from real data, against which the divergence scores computed by our tool can be compared and evaluated.

Recall that in our application, each of the input sequences does not correspond to a specific genomic sequence but rather represents a gene order that occurs in multiple genomes. Furthermore, abundant gene clusters typically display several paralogous occurrences of distinct gene orders, and possibly several paralogous occurrences of a specific gene order, within the same genome. Furthermore, each distinct occurrence of a specific gene order could differ substantially from another occurrence of the same gene order in terms of its encoding genomic sequence, since each COG represents a cluster of genomic sequences that are not identical however are similar enough (possibly based on local sequence similarity) to be clustered to the same orthology group.

Thus, we chose to represent each gene order by the assemblage of its instances, i.e. the set of genomes in which it occurs. Several similarity (or overlap) indices based on presence/absence (incidence) data have been proposed in the literature [27, 12]. A classical and widely used index in comparative assemblage analysis is the Jaccard index [12]. In our comparative evaluation the instance assemblages are used to estimate divergence rather than similarity, and therefore we use the inverse Jaccard Index as an estimator of the instance assemblage based divergence between two gene orders.

Our divergene measure was evaluated, per each cluster, as follows: first, we applied our approach (Alg. 1) to measure the structure informed divergence from the cluster's designated reference (explained in Section 4) to each of the other gene orders. Then, we calculated the Inverse Jaccard based distance from the set of instances of the reference gene order to the sets of instances of each of the other gene orders. In order to tolerate the noise due to inter-specie and inter-genus horizontal transfer of gene orders, we first converted the assemblages of genomes to the assemblages of (taxonomic) classes to which these genomes belong. This resulted in two series of scores, which were then subjected to the computation of (weighted average) Spearman and Pearson correlations between them. The same evaluation procedure was repeated per each cluster, this time using the signed break-point distance instead of our structure-informed divergence measure.

The results are given in Table 1. The 59 gene clusters were distributed to three groups in increasing level of colinear component dependency, according to the number of Q-nodes in their representative PQ-trees. This yielded 8 gene clusters whose representative tree has no Q-nodes, 41 gene clusters whose representative tree has one Q-node, and 10 gene clusters whose representative tree has two or more Q-nodes. For each group, the average Spearman and Pearson correlation scores were computed for both divergence measures. The average score for each group, per each measure, was computed as a weighted arithmetic mean, normalized by the size of each cluster in the group.

Table 1 A comparison between our proposed rearrangement measure ($\mathsf{m}_{\mathsf{Diverg}}$) and signed break-point distance ($\mathsf{d}_{\mathsf{SBP}}$), based on their correlation to a taxonomical instance abundance measure.

num of Q-nodes	Correlation	$\mathsf{m}_{\mathsf{Diverg}}$	$\mathsf{d}_{\mathsf{SBP}}$
0	Pearson	0.759	0.761
	Spearman	0.666	0.665
1	Pearson	0.861	0.844
	Spearman	0.745	0.716
2	Pearson	0.924	0.844
	Spearman	0.898	0.823

Table 1 indicates that, in general, the rearrangement-based divergence between a reference gene order and its target gene order correlates well with the divergence in the taxonomic distribution of their corresponding instances, which is an interesting result on its own.

For our proposed measure $\mathsf{m}_{\mathsf{Diverg}}$, the correlation increases with the number of Q-nodes in the representative trees. The advantage obtained by applying our structure-informed rearrangement approach, in comparison to signed break-point distance, also increases with the number of Q-nodes in the PQ-trees representing the gene clusters.

5.2 Examples

ABC transporters are among the most intensively studied gene clusters, as they form the largest group of paralogous genes in bacterial and archaeal genomes [31]. ABC transporters tend to form operons, and several studies indicate that the ancestral ABC transporter operons may have arisen early in evolution, before the speciation of bacteria and archaea [33]. We exemplify the application of the proposed structure-informed rearrangement approach via two ABC transporter gene clusters from our benchmark: a gene cluster encoding for a Dpp dipeptide uptake system, and another gene cluster encoding for a Ugp sugar uptake system. In both examples, the correlation between the $\mathsf{m}_{\mathsf{Diverg}}$ series for the cluster and the corresponding series of taxonomic instance abundance scores, was higher than that obtained for the $\mathsf{d}_{\mathsf{SBP}}$ measure.

The examples are illustrated in Fig. 6. Due to space constraints, we analyze only one rearrangement scenario per example. Additional information can be found in our GitHub directory, where we publish the extensive log information for each of the 59 gene clusters analyzed in our experiment.

Example 1: A gene cluster encoding a dipeptide uptake system, Figure 6.A-E. The Dpp transporter encoded by this gene cluster is composed of two integral membrane proteins (heterodimer TMDs), two peripheral membrane proteins that bind and hydrolyze ATP (heterodimer NBPs), and a periplasmic (or lipoprotein) substrate-binding protein (SBP) that delivers the substrate to a core importer (Figure 6.H). TMDs and NBDs dimerize and assemble the minimal unit of an importer, with the SBP as the fifth component of the complex.

The signed break-point distance for the reference and target gene orders is 2, based on the break-points (1,2) and (3,4). However, there is no experimental evidence for any special, asymmetric interactions between the products of genes 1 (SBP) and 2 (TMD1). On the contrary, it is well-known that the two lobes of the SBP interact with both of the TMDs [15] after a substrate has been bound by the SBP. Similary, there is no biological basis for assuming an assymmetric interaction between the products of the two genes 3 (TMD2) and 4 (NBP1) that would require direct positional adjacency.

Previous studies indicate that the importer complex is composed of two heterodimer components: NBP1-NBP2, and TMD1-TMD2 [15]. However, this structural information is not taken into account by the signed break-point distance computation. Thus, the break-point distance seems to poorly model the gene order divergence from the reference, ancestral gene order of this gene cluster to the target, rearranged copy.

Interestingly, when considering the conserved structure of the gene orders in this family (represented by the PQ-tree in Figure 6.A), the pattern by which the second gene order diverges from the first gene order becomes evident: the first gene order spells an "outside-in" transcription (and possibly assembly) order of the components of the complex (Figure 6.D), while the second gene order spells an "inside-out" transcription (and possibly assembly) order. This observation is better reflected by the $\mathsf{m}_{\mathsf{Diverg}}$ score for this gene order pair, which is 2.5. This score is obtained by assigning a cost of 1 for a break-point between two Q-nodes

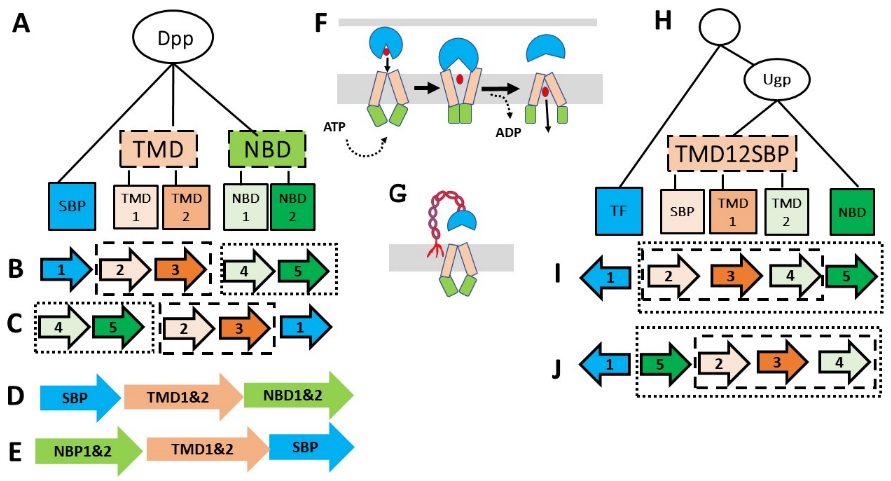

Figure 6 Examples from two ABC transporter gene clusters. **A,B,C.** Reference and target gene orders from the dipeptide uptake gene cluster DppBCD-GsiA-DdpA, and the representative PQ-tree for the corresponding gene cluster. **D,E.** Structure-informed encoding of the gene orders from B and C, respectively, reveals that the transcription order of their structural sub-components has been reversed by the re-arrangement. **F,G.** Substrate binding and import in ABC transporter in gram negative bacteria versus gram positive bacteria, respectively. **H,I,J.** Reference and target gene orders from the sugar uptake system UgpABE-MalK with the corresponding transcription factor PurR, and the representative PQ-tree for the corresponding gene cluster.

(representing the heterodimers TMD and NBD), increased by an additional penalty of 0.5 for a "jump" of a component of size 2, plus a cost of 1 for a break-point between the SBP gene and the Q-node representing TMD12. Thus, our structure-informed rearrangement approach is able to utilize the PQ-tree to capture the order reversal of the dimer-scale components of the encoded uptake machine.

When considering the taxonomical distribution of these two gene orders, we note that the reference order has instances in 263 genomes from our dataset: it is widely spread and spans both archea and bacteria, with many instances in both gram positive and gram negative bacteria. The second gene order, on the other hand, has 53 instances, 50 of which are confined to gram positive bacilli. Gram positive bacteria lack an outer membrane and consequently have no periplasm, and therefore their SPBs are lipoproteins that are tethered to the external surface of the cell membrane (Figure 6.G), anchoring the binding protein in the proximity of the external face of the TMDs of the transport system. To obtain such proximity, the TMDs need to be integrated to the cell membrane prior to the anchoring of the SBPS within their proximity. This could perhaps explain why a Dpp operon with "inside-out" transcription confers adaptation in some gram positive bacteria.

Example 2: A gene cluster encoding a sugar uptake system, and the transcription factor that regulates it. Figure 6.H-J. Here, the signed break-point distance between the reference gene order (Figure 6.I) and the target gene order (Figure 6.J) is 2, due to the break-points (1,2) and (4,5). In contrast, the $\mathsf{m}_{\mathsf{Diverg}}$ score is only 1. Using the structural information encoded by the PQ-tree, our engine recognizes that the break-point (1,2) may be functionally irrelevant, since the rearrangement events are confined to the P-node encapsulating the sub-cluster that encodes the uptake system machine, including gene 2 which codes for the

NBD MalK. Within the P-node encapsulating the Ugp operon, the NBD is separate from the rest of the genes in the transporter, which are confined to the Q-node, due to stochiometry (MalK is a homodimer [13]). Thus, it is realized by the PQ-tree, and conveyed via structure-informed re-arrangement, that the positional order between the uptake system operon and the transcription factor that regulates it (gene 1), remains intact during the rearrangement from the reference gene order to the target one.

References

1 Eric Alm, Katherine Huang, and Adam Arkin. The evolution of two-component systems in bacteria reveals different strategies for niche adaptation. *PLoS computational biology*, 2(11):e143, 2006.

2 Severine Bérard, Anne Bergeron, and Cedric Chauve. Conservation of combinatorial structures in evolution scenarios. In *RECOMB Workshop on Comparative Genomics*, pages 1–14. Springer, 2004.

3 Severine Bérard, Anne Bergeron, Cedric Chauve, and Christophe Paul. Perfect sorting by reversals is not always difficult. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4:4–16, 2007. `doi:10.1145/1229968.1229972`.

4 Sèverine Bérard, Annie Chateau, Cedric Chauve, Christophe Paul, and Eric Tannier. Computation of perfect dcj rearrangement scenarios with linear and circular chromosomes. *Journal of Computational Biology*, 16(10):1287–1309, 2009.

5 Severine Bérard, Cedric Chauve, and Christophe Paul. A more efficient algorithm for perfect sorting by reversals. *Information processing letters*, 106(3):90–95, 2008.

6 Anne Bergeron, Yannick Gingras, and Cedric Chauve. Formal models of gene clusters. *Bioinformatics Algorithms: Techniques and Applications*, 8:177–202, 2008. `doi:10.1002/9780470253441.ch8`.

7 Anne Bergeron, Julia Mixtacki, and Jens Stoye. Mathematics of evolution and phylogeny, chapter the inversion distance problem, 2005.

8 Matthias Bernt, Daniel Merkle, Kai Ramsch, Guido Fritzsch, Marleen Perseke, Detlef Bernhard, Martin Schlegel, Peter F Stadler, and Martin Middendorf. Crex: inferring genomic rearrangements based on common intervals. *Bioinformatics*, 23(21):2957–2958, 2007.

9 Guillaume Blin, Cedric Chauve, and Guillaume Fertin. The breakpoint distance for signed sequences. In *Proc. 1st Algorithms and Computational Methods for Biochemical and Evolutionary Networks (CompBioNets)*, pages 3–16, 2004.

10 Kellogg S Booth and George 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.

11 Marília DV Braga, Marie-France Sagot, Celine Scornavacca, and Eric Tannier. Exploring the solution space of sorting by reversals, with experiments and an application to evolution. *IEEE/ACM transactions on computational biology and bioinformatics*, 5(3):348–356, 2008.

12 Anne Chao, Robin L Chazdon, Robert K Colwell, and Tsung-Jen Shen. Abundance-based similarity indices and their estimation when there are unseen species in samples. *Biometrics*, 62(2):361–371, 2006.

13 Jue Chen, Gang Lu, Jeffrey Lin, Amy L Davidson, and Florante A Quiocho. A tweezers-like motion of the ATP-binding cassette dimer in an ABC transport cycle. *Molecular cell*, 12(3):651–661, 2003.

14 Aaron E Darling, István Miklós, and Mark A Ragan. Dynamics of genome rearrangement in bacterial populations. *PLoS genetics*, 4(7):e1000128, 2008.

15 Amy L Davidson, Elie Dassa, Cedric Orelle, and Jue Chen. Structure, function, and evolution of bacterial ATP-binding cassette systems. *Microbiology and molecular biology reviews*, 72(2):317–364, 2008.

16 Yoan Diekmann, Marie-France Sagot, and Eric Tannier. Evolution under reversals: Parsimony and conservation of common intervals. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(2):301–309, 2007.

17 Guillaume Fertin, Anthony Labarre, Irena Rusu, Stéphane Vialette, and Eric Tannier. *Combinatorics of genome rearrangements.* MIT press, 2009.

18 Martin Figeac and Jean-Stéphane Varré. Sorting by reversals with common intervals. In *International Workshop on Algorithms in Bioinformatics*, pages 26–37. Springer, 2004.

19 Marco Fondi, Giovanni Emiliani, and Renato Fani. Origin and evolution of operons and metabolic pathways. *Research in Microbiology*, 160(7):502–512, 2009. doi:10.1016/j.resmic.2009.05.001.

20 Yair E Gatt and Hanah Margalit. Common adaptive strategies underlie within-host evolution of bacterial pathogens. *Molecular biology and evolution*, 38(3):1101–1121, 2021.

21 Sridhar Hannenhalli and Pavel A Pevzner. Transforming men into mice (polynomial algorithm for genomic distance problem). In *Proceedings of IEEE 36th annual foundations of computer science*, pages 581–592. IEEE, 1995.

22 Sridhar Hannenhalli and Pavel A Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. *Journal of the ACM (JACM)*, 46(1):1–27, 1999.

23 Tom Hartmann, Matthias Bernt, and Martin Middendorf. An exact algorithm for sorting by weighted preserving genome rearrangements. *IEEE/ACM transactions on computational biology and bioinformatics*, 16(1):52–62, 2018.

24 Géraldine Jean and Macha Nikolski. Genome rearrangements: a correct algorithm for optimal capping. *Information Processing Letters*, 104(1):14–20, 2007.

25 Gad M Landau, Laxmi Parida, and Oren Weimann. Gene proximity analysis across whole genomes via PQ trees1. *Journal of Computational Biology*, 12(10):1289–1306, 2005.

26 Claire Lemaitre, Marilia DV Braga, Christian Gautier, Marie-France Sagot, Eric Tannier, and Gabriel AB Marais. Footprints of inversions at present and past pseudoautosomal boundaries in human sex chromosomes. *Genome biology and evolution*, 1:56–66, 2009.

27 Anne E Magurran. Measuring biological diversity. *Current Biology*, 31(19):R1174–R1177, 2021.

28 Marie-France Sagot and Eric Tannier. Perfect sorting by reversals. In *International Computing and Combinatorics Conference*, pages 42–51. Springer, 2005.

29 Dina Svetlitsky, Tal Dagan, and Michal Ziv-Ukelson. Discovery of multi-operon colinear syntenic blocks in microbial genomes. *Bioinformatics*, 2020. doi:10.1093/bioinformatics/btaa503.

30 Eric Tannier, Anne Bergeron, and Marie-France Sagot. Advances on sorting by reversals. *Discrete Applied Mathematics*, 155(6-7):881–888, 2007.

31 Roman L Tatusov, Arcady R Mushegian, Peer Bork, Nigel P Brown, William S Hayes, Mark Borodovsky, Kenneth E Rudd, and Eugene V Koonin. Metabolism and evolution of Haemophilus influenzae deduced from a whole-genome comparison with Escherichia coli. *Current biology*, 6(3):279–291, 1996.

32 Tatiana Tatusova, Stacy Ciufo, Boris Fedorov, Kathleen O'Neill, and Igor Tolstoy. Refseq microbial genomes database: new representation and annotation strategy. *Nucleic Acids Research*, 42(D1):D553–D559, 2014. doi:10.1093/nar/gkt1274.

33 Kentaro Tomii and Minoru Kanehisa. A comparative analysis of ABC transporters in complete microbial genomes. *Genome research*, 8(10):1048–1059, 1998.

34 Alice R Wattam, David Abraham, Oral Dalay, Terry L Disz, Timothy Driscoll, Joseph L Gabbard, Joseph J Gillespie, Roger Gough, Deborah Hix, Ronald Kenyon, et al. Patric, the bacterial bioinformatics database and analysis resource. *Nucleic Acids Research*, 42(D1):D581–D591, 2014. doi:10.1093/nar/gkt1099.

35 Jonathan N Wells, L Therese Bergendahl, and Joseph A Marsh. Operon gene order is optimized for ordered protein complex assembly. *Cell reports*, 14(4):679–685, 2016.

36 Sophia Yancopoulos, Oliver Attie, and Richard Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346, 2005.

37 G. R. Zimerman. The PQFinder tool. `www.github.com/GaliaZim/PQFinder`.

38 Galia R Zimerman, Dina Svetlitsky, Meirav Zehavi, and Michal Ziv-Ukelson. Approximate search for known gene clusters in new genomes using PQ-trees. *arXiv preprint arXiv:2007.03589*, 2020.