Approximate Search for Known Gene Clusters in New Genomes Using PQ-Trees

Galia R. Zimerman
Ben Gurion University of the Negev, Beer Sheva, Israel
zimgalia@gmail.com

Dina Svetlitsky
Ben Gurion University of the Negev, Beer Sheva, Israel
dina.svetlitsky@gmail.com

Meirav Zehavi
Ben Gurion University of the Negev, Beer Sheva, Israel
meiravze@bgu.ac.il

Michal Ziv-Ukelson[1]
Ben Gurion University of the Negev, Beer Sheva, Israel
michaluz@cs.bgu.ac.il

Abstract

We define a new problem in comparative genomics, denoted PQ-Tree Search, that takes as input a PQ-tree T representing the known gene orders of a gene cluster of interest, a gene-to-gene substitution scoring function h, integer parameters d_T and d_S, and a new genome S. The objective is to identify in S approximate new instances of the gene cluster that could vary from the known gene orders by genome rearrangements that are constrained by T, by gene substitutions that are governed by h, and by gene deletions and insertions that are bounded from above by d_T and d_S, respectively. We prove that the PQ-Tree Search problem is NP-hard and propose a parameterized algorithm that solves the optimization variant of PQ-Tree Search in $O^*(2^\gamma)$ time, where γ is the maximum degree of a node in T and O^* is used to hide factors polynomial in the input size.

The algorithm is implemented as a search tool, denoted PQFinder, and applied to search for instances of chromosomal gene clusters in plasmids, within a dataset of 1,487 prokaryotic genomes. We report on 29 chromosomal gene clusters that are rearranged in plasmids, where the rearrangements are guided by the corresponding PQ-tree. One of these results, coding for a heavy metal efflux pump, is further analysed to exemplify how PQFinder can be harnessed to reveal interesting new structural variants of known gene clusters.

2012 ACM Subject Classification Applied computing → Bioinformatics

Keywords and phrases PQ-Tree, Gene Cluster, Efflux Pump

Digital Object Identifier 10.4230/LIPIcs.WABI.2020.1

Related Version An extended version of the paper is archived in `https://arxiv.org/abs/2007.03589` [45].

Supplementary Material The code for the PQFinder tool as well as all the data needed to reconstruct the results are publicly available on GitHub (`https://github.com/GaliaZim/PQFinder`).

Funding The research of G.R.Z. and M.Z. was partially supported by the Israel Science Foundation (grant no. 1176/18). The research of G.R.Z., D.S. and M.Z.U. was partially supported by the Israel Science Foundation (grant no. 939/18).
Galia R. Zimerman: The research of G.R.Z. was partially supported by the Planning and Budgeting Committee of the Council for Higher Education in Israel and by the Frankel Center for Computer Science at Ben Gurion University.

Acknowledgements Many thanks to Lev Gourevitch for his excellent implementation of a PQ-tree builder. We also thank the anonymous WABI reviewers for their very helpful comments.

[1] M.Z and M.Z.U are joint corresponding authors.

1 Introduction

Recent advances in pyrosequencing techniques, combined with global efforts to study infectious diseases, yield huge and rapidly-growing databases of microbial genomes [38, 42]. These big new data statistically empower genomic-context based approaches to functional analysis: the biological principle underlying such analysis 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. Thus, if the functional association and annotation of the clustered genes is already known in one (or more) of the genomes, this information can be used to infer functional characterization of homologous genes that are clustered together in another genome.

Groups of genes that are co-locally conserved across many genomes are denoted *gene clusters*. The locations of the group of genes comprising a gene cluster in the distinct genomes are denoted *instances*. 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. However, the order of the genes in the distinct instances of a gene cluster may not be the same.

The discovery (i.e. data-mining) of conserved gene clusters in a given set of genomes is a well studied problem [8, 21, 44]. However, with the rapid sequencing of prokaryotic genomes a new problem is inspired: Namely, given an already known gene cluster that was discovered and studied in one genomic dataset, to identify all the instances of the gene cluster in a given new genomic sequence.

One exemplary application for this problem is the search for chromosomal gene clusters in plasmids. Plasmids are circular genetic elements that are harbored by prokaryotic cells where they replicate independently from the chromosome. They can be transferred horizontally and vertically, and are considered a major driving force in prokaryotic evolution, providing mutation supply and constructing new operons with novel functions [28], for example antibiotic resistance [20]. This motivates biologists to search for chromosomal gene clusters in plasmids, and to study structural variations between the instances of the found gene clusters across the two distinct replicons. However, in addition to the fact that plasmids evolve independently from chromosomes and in a more rapid pace [14], their sequencing, assembly and annotation involves a more noisy process [29].

To accommodate all this, the proposed search approach should be an approximate one, sensitive enough to tolerate some amount of genome rearrangements: transpositions and inversions, missing and intruding genes, and classification of genes with similar function to distinct orthology groups due to sequence divergence or convergent evolution. Yet, for the sake of specificity and search efficiency, we consider confining the allowed variations by two types of biological knowledge: (1) bounding the allowed rearrangement events considered by the search, based on some grammatical model trained specifically from the known gene orders of the gene cluster, and (2) governing the gene-to-gene substitutions considered by the search by combining sequence homology with functional-annotation based semantic similarity.

(1) Bounding the allowed rearrangement events. The PQ-tree [9] is a combinatorial data structure classically used to represent gene clusters [6]. A PQ-tree of a gene cluster describes its hierarchical inner structure and the relations between instances of the cluster succinctly, aids in filtering meaningful from apparently meaningless clusters, and also gives a natural and meaningful way of visualizing complex clusters. 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 or

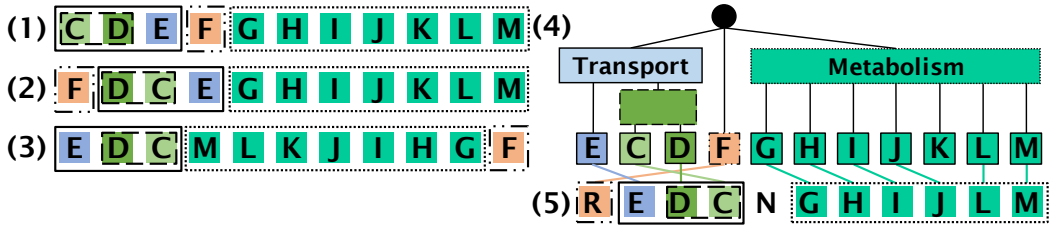


Figure 1 A gene cluster containing most of the genes of the *PhnCDEFGHIJKLMNOP* operon [25] and the corresponding PQ-tree. The *Phn* operon encodes proteins that utilize phosphonate as a nutritional source of phosphorus in prokaryotes. The genes *PhnCDE* encode a phosphonate transporter, the genes *PhnGHIJKLM* encode proteins responsible for the conversion of phosphonates to phosphate, and the gene *PhnF* encodes a regulator. **(1)-(3).** The three distinct gene orders found among 47 chromosomal instances of the *Phn* gene cluster. **(4).** A PQ-tree representing the *Phn* gene cluster, constructed from its three known gene orders shown in 1-3. **(5).** An example of a *Phn* gene cluster instance identified by the PQ-tree shown in (4), and the one-to-one mapping between the leaves of the PQ-tree and the genes comprising the instance. The instance genes are rearranged differently from the gene orders shown in 1-3 and yet can be derived from the PQ-tree. In this mapping, gene F is substituted by gene R, gene N is an intruding gene (i.e., deleted from the instance string), and gene K is a missing gene (i.e., deleted from the PQ-tree).

right-to-left order. (In the special case when a node has exactly two children, it does not matter whether it is labeled as a P-node or a Q-node.) Booth and Lueker [9], who introduced this data structure, were interested in representing a set of permutations over a set U, i.e. every member of U appears exactly once as a label of a leaf in the PQ-tree. We, on the other hand, allow each member of U to appear as a label of a leaf in the tree any non-negative number of times. Therefore, we will henceforth use the term *string* rather than *permutation* when describing the gene orders derived from a given PQ-tree.

An example of a PQ-tree is given in Figure 1. It represents a *Phn* gene cluster that encodes proteins that utilize phosphonate as a nutritional source of phosphorus in prokaryotes [25]. 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 [17]. 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 [43]. Thus, transposition events shuffling the order of the genes within this sub-operon could reduce its fitness. On the other hand, inversion events, in which the genes participating in this sub-operon remain colinearly ordered are accepted. This case is represented in the PQ-tree by a Q-node (marked with a rectangle). 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 [17]. This case is represented in the PQ-tree by a P-node (marked with a circle). Learning the internal topology properties of a gene cluster from its corresponding gene orders and constructing a query PQ-tree accordingly, could empower the search to confine the allowed rearrangement operations so that colinear dependencies among genes and between sub-operons are preserved.

(2) Governing the gene-to-gene substitutions. A prerequisite for gene cluster discovery is to determine how genes relate to each other across all the genomes in the dataset. In our experiment, genes are represented by their membership in Clusters of Orthologous

Groups (COGs) [37], where the sequence similarity of two genes belonging to the same COG serves as a proxy for homology. Despite low sequence similarity, genes belonging to two different COGs could have a similar function, which would be reflected in the functional description of the respective COGs. Using methods from natural language processing [31], we compute for each pair of functional descriptions a score reflecting their semantic similarity. Combining sequence and functional similarity could increase the sensitivity of the search and promote the discovery of systems with related functions.

Our Contribution and Roadmap. In this paper we define a new problem in comparative genomics, denoted PQ-Tree Search **(in Section 2)**, that takes as input a PQ-tree T (the query) representing the known gene orders of a gene cluster of interest, a gene-to-gene substitution scoring function h, integer parameters d_T and d_S, and a new genome S (the target). The objective is to identify in S a new approximate instance of the gene cluster that could vary from the known gene orders by genome rearrangements that are constrained by T, by gene substitutions that are governed by h, and by gene deletions and insertions that are bounded from above by d_T and d_S, respectively. We prove that PQ-Tree Search is NP-hard **(Theorem 9 in Appendix A)**.

We define an optimization variant of PQ-Tree Search and propose an algorithm **(in Section 3)** that solves it in $O(n\gamma {d_T}^2 {d_S}^2(m_p \cdot 2^\gamma + m_q))$ time, where n is the length of S, m_p and m_q denote the number of P-nodes and Q-nodes in T, respectively, and γ denotes the maximum degree of a node in T. In the same time and space complexities, we can also report all approximate instances of T in S and not only the optimal one.

The algorithm is implemented as a search tool, denoted PQFinder. The code for the tool as well as all the data needed to reconstruct the results are publicly available on GitHub (`https://github.com/GaliaZim/PQFinder`). The tool is applied to search for instances of chromosomal gene clusters in plasmids, within a dataset of 1,487 prokaryotic genomes. In our preliminary results **(given in Section 5)**, we report on 29 chromosomal gene clusters that are rearranged in plasmids, where the rearrangements are guided by the corresponding PQ-tree. One of these results, coding for a heavy metal efflux pump, is further analysed to exemplify how PQFinder can be harnessed to reveal interesting new structural variants of known gene clusters.

Previous Related Works. Permutations on strings representing gene clusters have been studied earlier by [5, 15, 22, 32, 39]. PQ-trees were previously applied in physical mapping [2, 10], as well as to other comparative genomics problems [3, 7, 24].

In Landau et al. [24] an algorithm was proposed for representation and detection of gene clusters in multiple genomes, using PQ-trees: the proposed algorithm computes a PQ-tree of k permutations of length n in $O(kn)$ time, and it is proven that the computed PQ-tree is the one with a minimum number of possible rearrangements of its nodes while still representing all k permutations. In the same paper, the authors also present a general scheme to handle gene multiplicity and missing genes in permutations. For every character that appears a times in each of the k strings, the time complexity for the construction of the PQ-tree, according to the scheme in that paper, is multiplied by an $O((a!)^k)$ factor.

Additional applications of PQ-trees to genomics were studied in [1, 4, 30], where PQ-trees were considered to represent and reconstruct ancestral genomes.

However, as far as we know, searching for approximate instances of a gene cluster that is represented as a PQ-tree, in a given new string, is a new computational problem.

2 Preliminaries

Let Π be an NP-hard problem. In the framework of Parameterized Complexity, each instance of Π is associated with a *parameter* k, and the goal is to confine the combinatorial explosion in the running time of an algorithm for Π to depend only on k. Formally, Π is *fixed-parameter tractable (FPT)* if any instance (I, k) of Π is solvable in time $f(k) \cdot |I|^{\mathcal{O}(1)}$, where f is an arbitrary computable function of k. Nowadays, Parameterized Complexity supplies a rich toolkit to design or refute the existence of FPT algorithms [11, 12, 16].

PQ-Tree: Representing the Pattern. The possible reordering of the children nodes in a PQ-tree may create many equivalent PQ-trees. Booth and Lueker [9] defined two PQ-trees T, T' as *equivalent* (denoted $T \equiv T'$) if one tree can be obtained by legally reordering the nodes of the other; namely, randomly permuting the children of a P-node, and reversing the children of a Q-node. To allow for deletions in the PQ-trees, a generalization of their definition is given in Definition 1 below. Here, *smoothing* is a recursive process in which if by deleting leaves from a tree T, some internal node x of T is left without children, then x is also deleted, but its deletion is not counted (i.e. only leaf deletions are counted).

▶ **Definition 1** (Quasi-Equivalence Between PQ-Trees). *For any two PQ-trees, T and T', the PQ-tree T is* quasi-equivalent to T' with a limit d, *denoted $T \succeq_d T'$, if T' can be obtained from T by (a) randomly permuting the children of some of the P-nodes of T, (b) reversing the children of some of the Q-nodes of T, and (c) deleting up to d leaves from T and applying the corresponding smoothing. (The order of the operations does not matter.)*

Figure S2 shows two equivalent PQ-trees (Figure S2a, Figure S2b) that are each quasi-equivalent with $d = 1$ to the third PQ-tree (Figure S2c). The *frontier* of a PQ-tree T, denoted $F(T)$, is the sequence of labels on the leaves of T read from left to right. For example, the frontier of the PQ-tree in Figure 1 is $ECDFGHIJKLM$. It is interesting to consider the set of frontiers of all the equivalent PQ-trees, defined in [9] as a *consistent set* and denoted by $C(T) = \{F(T') : T \equiv T'\}$. Intuitively, $C(T)$ is the set of all leaf label sequences defined by the PQ-tree structure and obtained by legally reordering its nodes. Here, we generalize the consistent set definition to allow a bounded number of deletions from T, using quasi-equivalence.

▶ **Definition 2** (d-Bounded Quasi-Consistent Set). $C_d(T) = \{F(T') : T \succeq_d T'\}$.

clearly $C_0(T) = C(T)$, and so in a setting where $d = 0$ the latter notation is used. For a node x of a PQ-tree T, the subtree of T rooted in x is denoted by $T(x)$, the set of leaves in $T(x)$ is denoted by $\mathsf{leaves}(x)$, and the *span* of x (denoted $\mathsf{span}(x)$) is defined as $|\mathsf{leaves}(x)|$.

PQ-Tree Search and Related Terminology. An instance of the PQ-Tree Search problem is a tuple (T, S, h, d_T, d_S), where T is a PQ-tree with m leaves, m_p P-nodes, m_q Q-nodes and every leaf x in T has a label $\mathsf{label}(x) \in \Sigma_T$; $S = \sigma_1 \dots \sigma_n \in \Sigma_S^n$ is a string of length n representing the input genome; $d_T \in \mathbb{N}$ specifies the number of allowed deletions from T; $d_S \in \mathbb{N}$ specifies the number of allowed deletions from S; and h is a *boolean substitution function*, describing the possible substitutions between the leaf labels of T and the characters of the given string, S. Formally, h is a function that receives a pair (σ_t, σ_s), where $\sigma_t \in \Sigma_T$ is one of the labels on the leaves of T, and $\sigma_s \in \Sigma_S$ is one of the characters of the given string S, and returns $True$ if σ_t can be replaced with σ_s, and $False$, otherwise. Considering the biological problem at hand, Σ_T and Σ_S are both sets of genes. For $1 \leq i \leq j \leq n$,

$S' = S[i:j] = \sigma_i ... \sigma_j$ is a substring of S beginning at index i and ending at index j. The substring S' is a *prefix* of S if $S' = S[1:j]$ and it is a *suffix* of S if $S' = S[i:n]$. In addition, we denote σ_i, the i^{th} character of S, by $S[i]$.

The objective of PQ-Tree Search is to find a one-to-one mapping $\mathcal{M}$ between the leaves of T and the characters of a substring S' of S, that comprises a set of pairs each having one of three forms: the substitution form, $(x, \sigma_s(\ell))$, where x is a leaf in T, $\sigma_s \in \Sigma_S$, $h(\mathsf{label}(x), \sigma_s) = True$ and $\ell \in \{1, \ldots, n\}$ is the index of the occurrence of σ_s in S that is mapped to the leaf x; the character deletion form, $(\varepsilon, \sigma_s(\ell))$, which marks the deletion of the character $\sigma_s \in \Sigma_S$ at index ℓ of S; the leaf deletion form, (x, ε), which marks the deletion of x, a leaf node of T.

To account for the number of deletions of characters of S' and leaves of T in $\mathcal{M}$, the number of pairs in $\mathcal{M}$ of the form (ε, σ) are marked by $\mathsf{del}_S(\mathcal{M})$ and the number of pairs in $\mathcal{M}$ of the form (x, ε) are marked by $\mathsf{del}_T(\mathcal{M})$. Applying the substitutions defined in $\mathcal{M}$ to S' resulting in the string $S_{\mathcal{M}}$ is the process in which for every $(x, \sigma_s(\ell)) \in \mathcal{M}$, the character σ_s at index ℓ of S is deleted if $x = \varepsilon$, and otherwise substituted by x. This process is demonstrated in Figure S3b. We say that S' is *derived* from T *under* $\mathcal{M}$ with d_T deletions from the tree and d_S deletions from the string, if $d_T = \mathsf{del}_T(\mathcal{M})$, $d_S = \mathsf{del}_S(\mathcal{M})$ and $S_{\mathcal{M}} \in C_{d_T}(T)$. Thus, by definition, there is a PQ-tree T' such that $F(T') = S_{\mathcal{M}}$ and $T \succeq_{d_T} T'$. Note that the deletions of the nodes in T to obtain the nodes in T' are determined by $\mathcal{M}$. The conversion of T to T' as defined by the derivation is illustrated in Figure S3a. The set of permutations and node deletions performed to obtain T' from T together with the substitutions and deletions from S' specified by $\mathcal{M}$ is named the *derivation* μ of T to S'. We also say that $\mathcal{M}$ *yields* the derivation μ.

For a derivation μ of T to $S' = S[s:e]$, we give the following terms and notations (illustrated in Figure S3). The root of T (denoted $root_T$[2]) is *the node that* μ *derives* or *the root of the derivation* and it is denoted by $\mu.v$. For abbreviation, we say that μ *is a derivation of* $\mu.v$. The substring S' is *the string that* μ *derives*. We name s and e the start and end points of the derivation and denote them by $\mu.s$ and $\mu.e$, respectively. The one-to-one mapping that yields μ is denoted by $\mu.o$. The number of deletions from the tree is denoted by $\mu.del_T$. The number of deletions from the string is denoted by $\mu.del_S$. In addition, if x is a leaf node in T and $(x, \sigma_s(\ell)) \in \mu.o$, then x is *mapped to* $S[\ell]$ *under* μ. The character $S[\ell]$ is said to be *deleted under* μ if $(\varepsilon, \sigma_s(\ell)) \in \mu.o$. If $x \in T(\mu.v)$ is a leaf for which $(x, \varepsilon) \in \mu.o$, then x is *deleted under* μ. For an internal node of T, x, if every leaf in $T(x)$ is deleted under μ, then x is *deleted under* μ, and otherwise x is *kept under* μ.

We define two versions of the PQ-Tree Search problem: a decision version (Definition 3) and an optimisation version (Definition 4).

▶ **Definition 3** (Decision PQ-Tree Search)**.** *Given a string S of length n, a PQ-tree T with m leaves, deletion limits $d_T, d_S \in \mathbb{N}$, and a boolean substitution function h between Σ_S and Σ_T, decide if there is a one-to-one mapping $\mathcal{M}$ that yields a derivation of T to a substring S' of S with up to d_T and up to d_S deletions from T and S', respectively.*

To define an optimization version of the PQ-Tree Search problem it is necessary to have a score for every possible substitution between the characters in Σ_T and the characters in Σ_S. Hence, for this problem variant assume that h is a *substitution scoring function*, that is, $h(\sigma_t, \sigma_s)$ for $\sigma_t \in \Sigma_T, \sigma_s \in \Sigma_S$ is the score for substituting σ_s by σ_t in the derivation, and if σ_t cannot be substituted by σ_s, then $h(\sigma_t, \sigma_s) = -\infty$. In addition, we need a cost function, denoted by δ, for the deletion of a character of S and for the deletion of a leaf of T according

[2] We abuse notation and use the term $root_T$ also to refer to the index of the root in T.

to the label of the leaf. The score of a derivation μ, denoted by $\mu.score$, is the sum of scores of all operations (deletions from the tree, deletions from the string and substitutions) in μ. Now, instead of deciding whether there is a one-to-one mapping that yields a derivation of T to a substring of S, we can search for the one-to-one mapping that yields the best derivation (if there exists such a derivation), i.e. a one-to-one mapping for which $\mu.score$ is the highest.

▶ **Definition 4** (Optimization PQ-Tree Search). *Given a string of length n, S, a PQ-tree with m leaves, T, deletion limits $d_T, d_S \in \mathbb{N}$, a substitution scoring function between Σ_S and Σ_T, h, and a deletion cost function, δ, return the one-to-one mapping, $\mathcal{M}$, that yields the highest scoring derivation of T to a substring S' of S with up to d_T deletions from T and up to d_S deletions from S' (if such a mapping exists).*

3 A Parameterized Algorithm

In this section we develop a dynamic programming (DP) algorithm to solve the optimization variant of PQ-Tree Search (Definition 4). Our algorithm receives as input an instance of PQ-Tree Search (T, S, h, d_T, d_S), where h is a substitution scoring function as defined in Section 2. Our default assumption is that deletions are not penalized, and therefore δ is not given as input. The case where deletions are penalized, as well as additional technical details, are omitted due to lack of space, and can be found in [45]. The output of the algorithm is a one-to-one mapping, $\mathcal{M}$, that yields the best (highest scoring) derivation of T to a substring of S with up to d_T deletions from T and up to d_S deletions from the substring, and the score of that derivation. With a minor modification, the output can be extended to include a one-to-one mapping for every substring of S and the derivations that they yield.

Brief Overview. On a high level, our algorithm consists of three components: the main algorithm, and two other algorithms that are used as procedures by the main algorithm. Apart from an initialization phase, the crux of the main algorithm is a loop that traverses the given PQ-tree, T. For each internal node x, it calls one of the two other algorithms: P-mapping (given in Section 3.3) and Q-mapping (deferred to [45], due to space constraints). These algorithms find and return the best derivations from the subtree of T rooted in x, $T(x)$, to substrings of S, based on the type of x (P-node or Q-node). Then, the scores of the derivations are stored in the DP table.

We now give a brief informal description of the main ideas behind our P-mapping and Q-mapping algorithms. Our P-mapping algorithm is inspired by an algorithm described by Bevern et al. [40] to solve the Job Interval Selection problem. Our problem differs from theirs mainly in its control of deletions. Intuitively, in the P-mapping algorithm we consider the task at hand as a packing problem, where every child of x is a set of intervals, each corresponding to a different substring. The objective is to pack non-overlapping intervals such that for every child of x at most one interval is packed. Then, the algorithm greedily selects a child x' of x and decides either to pack one of its intervals (and which one) or to pack none (in which case x' is deleted). Our Q-mapping algorithm is similar to the P-mapping algorithm, but simpler. It can be considered as an interval packing algorithm as well, however, this algorithm packs the children of x in a specific order.

In the following sections, we describe the main algorithm, the P-mapping algorithm, and afterwards analyse the time complexity.

3.1 The Main Algorithm

We now delve into more technical details. The algorithm constructs a 4-dimensional DP table $\mathcal{A}$ of size $m' \times n \times d_T + 1 \times d_S + 1$, where $m' = m + m_p + m_q$ is the number of nodes in T. The purpose of an entry of the DP table, $\mathcal{A}[j, i, k_T, k_S]$, is to hold the highest score of a derivation of the subtree $T(x_j)$ to a substring S' of S starting at index i with k_T deletions from $T(x_j)$ and k_S deletions from S'. If no such derivation exists, $\mathcal{A}[j, i, k_T, k_S] = -\infty$. Addressing $\mathcal{A}$ with some of its indices given as dots, e.g. $\mathcal{A}[j, i, \cdot, \cdot]$, refers to the subtable of $\mathcal{A}$ that is comprised of all entries of $\mathcal{A}$ whose first two indices are j and i. Some entries of the DP table define illegal derivations, namely, derivations for which the number of deletions are inconsistent with the start index, i, the derived node and S. These entries are called *invalid entries* and their value is defined as $-\infty$ throughout the algorithm.

The algorithm first initializes the entries of $\mathcal{A}$ that are meant to hold scores of derivations of the leaves of T to every possible substring of S. Afterwards, 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 ascending order over every index, i, that can be a start index for the substring of S derived from $T(x)$ (the possible values of i are explained in the next paragraph). For every x and i, use the algorithm for Q-mapping or P-mapping according to the type of x. Both algorithms receive the same input: a substring S' of S, the node x, its children $x_1, \ldots, x_\gamma$, the collection of possible derivations of the children (denoted by $\mathcal{D}$), which have already been computed and stored in $\mathcal{A}$ (as will be explained ahead) and the deletion arguments d_T, d_S. Intuitively, the substring S' is the longest substring of S starting at index i that can be derived from $T(x)$ given d_T and d_S. After being called, both algorithms return a set of derivations of $T(x)$ to a prefix of $S' = S[i : e]$ and their scores. The set holds the highest scoring derivation for every $E(x_j, i, d_T, 0) \leq e \leq E(x_j, i, 0, d_S)$ and for every legal deletion combination $0 \leq k_T \leq d_T$, $0 \leq k_S \leq d_S$.

We now explain the possible values of i and the definition of S' more formally. To this end, note that given the node x and some numbers of deletions k_T and k_S, the length of the derived substring is $L(x, k_T, k_S) \doteq \mathsf{span}(x) - k_T + k_S$. Thus, on the one hand, a substring of maximum length is obtained when there are no deletions from the tree and d_S deletions from the string. Hence, $S' = S[i : E(x, i, 0, d_S)]$ where $E(x, i, k_T, k_S)$ is the function for the calculation of the end point of a derivation, defined as $E(x, i, k_T, k_S) \doteq i - 1 + L(x, k_T, k_S)$. On the other hand, a shortest substring is obtained when there are d_T deletions from the tree and none from the string. Then, the length of the substring is $L(x, d_T, 0) = \mathsf{span}(x) - d_T$. Hence, the index i runs between 1 and $n - (\mathsf{span}(x) - d_T) + 1$.

We now turn to address the aforementioned input collection $\mathcal{D}$ in more detail. Formally, it contains the best scoring derivations of every child x_j of x to every substring of S' with up to d_T and d_S deletions from the tree and string, respectively. It is produced from the entries $\mathcal{A}[j, i', k_T, k_S]$ (where each entry gives one derivation) for all k_T and k_S, and all i' between i and the end index of S', i.e. $i \leq i' \leq E(x_j, i, 0, d_S)$. For the efficiency of the Q-mapping and P-mapping algorithms, the derivations in $\mathcal{D}$ are arranged in descending order with respect to their end point ($\mu.e$). This does not increase the time complexity of the algorithm, as this ordering is received by previous calls to the Q-mapping and P-mapping algorithms.

In the final stage of the main algorithm, when the DP table is full, the score of a best derivation is the maximum of $\{\mathcal{A}[root_T, i, k_T, k_S] : k_T \leq d_T,\ k_S \leq d_S,\ 1 \leq i \leq n - (\mathsf{span}(root_T) - k_T) + 1\}$. We remark that by tracing back through $\mathcal{A}$ the one-to-one mapping that yielded this derivation can be found.

3.2 P-Node and Q-Node Mapping: Terminology

Before describing the P-mapping algorithm, we set up some terminology, which is useful both for the P-mapping algorithm and the Q-mapping algorithm.

We first define the notion of a partial derivation. In the Q-mapping and P-mapping algorithms, the derivation of the input node, x, is built by considering subsets U of its children. With respect to such a subset U, a derivation μ of x is built as if x had only the children in U, and is called a *partial derivation*. Formally, μ is a partial derivation of a node x if $\mu.v = x$ and there is a subset of children $U' \subseteq \mathsf{children}(x)$ such that the two following conditions are true. First, for every $u \in U'$ all the leaves in $T(u)$ are neither mapped nor deleted under μ - that is, there is no mapping pair $(\ell, y) \in \mu.o$ such that $\ell \in \mathsf{leaves}(u)$. Second, for every $v \in \mathsf{children}(x) \setminus U'$ the leaves in $T(v)$ are either mapped or deleted under μ. For every $u \in U'$, we say that u is *ignored under* μ. Notice that any derivation is a partial derivation, where the set of ignored nodes (U' above) is empty. Since all derivations that are computed in a single call to the P-mapping or Q-mapping algorithms have the same start point i, it can be omitted (for brevity) from the end point function: thus, we denote $E_I(x, k_T, k_S) \doteq L(x, k_T, k_S)$. Then, for a set U of nodes, we define $L(U, k_T, k_S) \doteq \sum_{x \in U} \mathsf{span}(x) + k_S - k_T$ and accordingly $E_I(U, k_T, k_S) \doteq L(U, k_T, k_S)$.

We now define certain collections of derivations with common properties (such as having the same numbers of deletions and end point).

▶ **Definition 5.** *The collection of all the derivations of every node $u \in U$ to suffixes of $S'[1 : E_I(U, k_T, k_S)]$ with exactly k_T deletions from the tree and exactly k_S deletions from the string is denoted by $\mathcal{D}(U, k_T, k_S)$.*

▶ **Definition 6.** *The collection of all the best derivations from the nodes in U to suffixes of $S'[1 : E_I(U, k_T, k_S)]$ with up to k_T deletions from the tree and up to k_S deletions from the string is denoted by $\mathcal{D}_{\leq}(U, k_T, k_S)$. Specifically, for every node $u \in U$, $k'_T \leq k_T$ and $k'_S \leq k_S$, the set $\mathcal{D}_{\leq}(U, k_T, k_S)$ holds only one highest scoring derivation of u to a suffix of $S'[1 : E_I(U, k_T, k_S)]$ with k'_T and k'_S deletions from the tree and string, respectively.*[3]

It is important to distinguish between these two definitions. First, the derivations in $\mathcal{D}(U, k_T, k_S)$ have *exactly* k_T and k_S deletions, while the derivations in $\mathcal{D}_{\leq}(U, k_T, k_S)$ have *up to* k_T and k_S deletions. Second, in $\mathcal{D}(U, k_T, k_S)$ there can be several derivations that differ only in their score and in the one-to-one mapping that yields them, while in $\mathcal{D}_{\leq}(U, k_T, k_S)$, there is only one derivation for every node $u \in U$ and deletion combination pair (k'_T, k'_S). Note that the end points of all of the derivations are equal.

Definition 5 is used for describing the content of an entry of the DP table, where the focus is on the collection of all the derivations of x to S' with exactly k_T and k_S deletions, $\mathcal{D}(\{x\}, k_T, k_S)$. For simplicity, the abbreviation $\mathcal{D}(u, k_T, k_S) = \mathcal{D}(\{u\}, k_T, k_S)$ is used. In every step of the P-mapping and Q-mapping algorithms, a different set of derivations of the children of x is examined, thus, Definition 6 is used for $U \subseteq \mathsf{children}(x)$. In addition, the set of derivations $\mathcal{D}$ that is received as input to the algorithms can be described using Definition 6 as can be seen in Equation (1) below. In this equation, the union is over all $U \subseteq \mathsf{children}(x)$ because in this way the derivations of all the children of x with *every possible*

[3] $\mathcal{D}_{\leq}(U, k_T, k_S)$ can be defined using Definition 5: $\mathcal{D}_{\leq}(U, k_T, k_S) = \bigcup_{u \in U} \bigcup_{k'_T \leq k_T} \bigcup_{k'_S \leq k_S} \max_{\substack{\mu \in \mathcal{D}(U, k_T, k_S) \\ \text{s.t.} \\ \mu.del_T = k'_T \\ \mu.del_S = k'_S \\ \mu.v = u}} \mu.score.$

end point are obtained (in contrast to having only $U = \mathsf{children}(x)$, which results in the derivations of all the children of x with the end point $E_I(\mathsf{children}(x), k_T, k_S)$).

$$\mathcal{D} = \bigcup_{U \subseteq \mathsf{children}(x)} \bigcup_{k_T \leq d_T} \bigcup_{k_S \leq d_S} \mathcal{D}_{\leq}(U, k_T, k_S) \tag{1}$$

In the P-mapping algorithm for $C \subseteq \mathsf{children}(x)$, the notation $x^{(C)}$ is used to indicate that the node x is considered as if its only children are the nodes in C. Consequentially, the span of $x^{(C)}$ is defined as $\mathsf{span}(x^{(C)}) \doteq \sum_{c \in C} \mathsf{span}(c)$, and the set $\mathcal{D}(x^{(C)}, k_T, k_S)$ (in Definition 5 where $U = \{x^{(C)}\}$) now refers to a set of *partial* derivations.

3.3 P-Node Mapping: The Algorithm

Recall that the input consists of an internal P-node x, a string S', limits on the number of deletions from the tree T and the string S', d_T and d_S, respectively, and a set of derivations $\mathcal{D}$ (see Equation (1)). The output is $\bigcup_{k_T \leq d_T} \bigcup_{k_S \leq d_S} \arg\max_{\mu \in \mathcal{D}(x, k_T, k_S)} \mu.score$, which is the collection of the best scoring derivations of x to every possible prefix of S' having up to d_T and d_S deletions from the tree and string, respectively. Thus, there are $O(d_T d_S)$ derivations in the output.

The algorithm constructs a 3-dimensional DP table $\mathcal{P}$, which has an entry for every $0 \leq k_T \leq d_T$, $0 \leq k_S \leq d_S$ and subset $C \subseteq \mathsf{children}(x)$. The purpose of an entry $\mathcal{P}[C, k_T, k_S]$ is to hold the best score of a partial derivation in $\mathcal{D}(x^{(C)}, k_T, k_S)$, i.e. a partial derivation rooted in $x^{(C)}$ to a prefix of S' with exactly k_T deletions from the tree and k_S deletions from the string. The children of x that are not in C are *ignored* (as defined in Section 3.2) under the partial derivation stored by the DP table entry $\mathcal{P}[C, k_T, k_S]$, thus they are neither deleted nor counted in the number of deletions from the tree, k_T. (They will be accounted for in the computation of other entries of $\mathcal{P}$.) Similarly to the main algorithm, some of the entries of $\mathcal{P}$ are invalid, and their value is defined as $-\infty$. Every entry $\mathcal{P}[C, k_T, k_S]$ for which $L(C, k_T, k_S) = 0$ and $k_S = 0$ or for which $C = \emptyset$ and $k_T = 0$ is initialized with 0.

After the initialization, the remaining entries of $\mathcal{P}$ are calculated using the recursion rule in Equation (2) below. The order of computation is ascending with respect to the size of the subsets C of the children of x, and for a given $C \subseteq \mathsf{children}(x)$, the order is ascending with respect to the number of deletions from both tree and string.

$$\mathcal{P}[C, k_T, k_S] = \max \begin{cases} \mathcal{P}[C, k_T, k_S - 1] \\ \max\limits_{\mu \in \mathcal{D}_{\leq}(C, k_T, k_S)} \mathcal{P}[C \setminus \{\mu.v\}, k_T - \mu.del_T, k_S - \mu.del_S] + \mu.score \end{cases} \tag{2}$$

Intuitively, every entry $\mathcal{P}[C, k_T, k_S]$ defines some index e' of S' that is the end point of every partial derivation in $\mathcal{D}(x^{(C)}, k_T, k_S)$. Thus, $S'[e']$ must be a part of any partial derivation $\mu \in \mathcal{D}(x^{(C)}, k_T, k_S)$, so, either $S'[e']$ is deleted under μ or it is mapped under μ. The former option is captured by the first case of the recursion rule. If $S'[e']$ is mapped under μ, then due to the hierarchical structure of $T(x)$, it must be mapped under some derivation μ' of one of the children of x that are in C. Thus we receive the second case of the recursion rule. We remark that the case of a node deletion is captured by the initialization.

Once the entire DP table is filled, a derivation of maximum score for every end point and deletion numbers combination can be found in $\mathcal{P}[\mathsf{children}(x), \cdot, \cdot]$. Traversing the said subtable in a specific order guarantees the output derivations are ordered with respect to their end point without further calculations.

3.4 Complexity Analysis of the Main Algorithm

In this section we compare the time complexity of the main algorithm (in Section 3.1) to the naïve solution for PQ-Tree Search. Lemma 8 ahead (proven in Appendix B) gives the time complexity of the main algorithm, and thus it is proven that PQ-Tree Search has an FPT solution with the parameter γ (Theorem 7).

▶ **Theorem 7.** *PQ-Tree Search with parameter γ is FPT. Particularly, it has an FPT algorithm that runs in $O^*(2^\gamma)$ time*[4].

▶ **Lemma 8.** *The algorithm in Section 3.1 runs in $O(n\gamma {d_T}^2 {d_S}^2(m_p 2^\gamma + m_q))$ time and $O(d_T d_S(mn + 2^\gamma))$ space, where γ is the maximum degree of a node in T.*

The naïve solution for PQ-Tree Search solves it in $O(2^{m_q}(\gamma!)^{m_p} nm(d_T + d_S)d_T d_S)$ time. Therefore, we conclude that the time complexity of our algorithm is substantially better, as exemplified by considering two complementary cases. One, when there are only P-nodes in T (i.e.$m_p = O(m)$), the naïve algorithm is super-exponential in γ, and even worse, exponential in m, while ours is exponential only in γ, and hence polynomial for any γ that is constant (or even logarithmic in the input size). Second, when there are only Q-nodes in T (i.e. $m_q = O(m)$), the naïve algorithm is exponential while ours is polynomial.

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). Among these strains, 471 genomes included a total of 933 plasmids.

The gene clusters were generated using the tool CSBFinder-S [36]. CSBFinder-S was applied to all the genomes in the dataset after removing their plasmids, using parameters $q = 1$ (a colinear gene cluster is required to appear in at least one genome) and $k = 0$ (no insertions are allowed in a colinear gene cluster), resulting in 595,708 colinear gene clusters. Next, ignoring strand and gene order information, colinear gene clusters that contain the exact same COGs were united to form the generalized set of gene clusters. The resulting gene clusters were then filtered to 26,270 gene clusters that appear in more than 30 genomes.

Generation of PQ-Trees. The generation of PQ-trees was performed using a program [19] that implements the algorithm described in [24] for the construction of a PQ-tree from a list of strings comprised from the same set of characters. In the case where a character appeared more than once in a training string, the PQ-tree with a minimum sized consistent set was chosen. The generated PQ-trees varied in size and complexity. The length of their frontier ranged between 4 and 31, and the size of their consistent set ranged between 4 and 362,880.

Implementation and Performance. PQFinder is implemented in Java 1.8. The runs were performed on an Intel Xeon X5680 machine with 192 GB RAM. The time it took to run all plasmid genomes against one PQ-tree ranged between 5.85 seconds (for a PQ-tree with a consistent set of size 4) and 181.5 seconds (for a PQ-tree with a consistent set of size 362,880). In total it took an hour and 47 minutes to run every one of the 779 PQ-trees against every one of the 933 plasmids.

[4] The notation O* is used to hide factors polynomial in the input size.

Substitution Scoring Function. The substitution scoring function reflects the distance between each pair of COGs, that is computed based on sentences describing the functional annotation of the COGs (e.g., "ABC-type sugar transport system, ATPase component"). The "Bag of Words model" was employed, where the functional description of each COG is represented by a sparse vector that is normalized to have a unit Euclidean norm. First, each COG description was tokenized and the occurrences of tokens in each description was counted and normalized using tf–idf term weighting. Then, the cosine similarity between each two vectors was computed, resulting in similarity scores ranging between 0 and 1. The sentences describing COGs are short, therefore each word largely influences the score, even after the tf–idf term weighting. Therefore, words that do not describe protein functions that were found in the top 30 most common words in the description of all COGs were used as stop-words. Two COGs with the same COG IDs were set to have a score of 1.1, and the substitution score between a gene with no COG annotation to any other COG was set to be -0.1. Two COGs with a zero score were penalized to have a score of -0.2 and the deletion of a COG from the query or the target string was set to have a score of zero.

Enrichment Analysis. For each of the four variants in Figure 2.C, a hypergeometric test was performed to measure the enrichment of the corresponding variant in one of the classes in which it appears. A total of 10 p-values were computed and adjusted using the Bonferroni correction; two p-values were found significant (<0.05), reported in Section 5.

Specificity Score. We define a specificity score for a PQ-tree T of a gene cluster named S-score. Let $\tilde{T}$ be the least specific PQ-tree that could have been generated for the genes of the gene cluster based on which T was constructed. Namely, a PQ-tree that allows all permutations of said genes, has height 1 and is rooted in a P-node whose children (being the leaves of the tree) are the leaves of T. Thus, the S-score of T is $\frac{|C(\tilde{T})|}{|C(T)|}$. For a gene cluster of permutations (i.e. there are no duplications), the computation of $|C(T)|$ is as described in Equation (3), where the set of P-nodes in T is denoted by $T.p$.

$$|C(T)| = 2^{m_q} \cdot \prod_{x \in T.p} |\mathsf{children}(x)|! \tag{3}$$

For a gene cluster that has duplications, the set $C(T)$ is generated to learn its size. Let $\mathsf{a}(\ell, T)$ denote the number of appearances of the label ℓ in the leaves of T and let $\mathsf{labels}(T)$ denote the set of all labels of the leaves of T. So, the formula for $|C(\tilde{T})|$ is as in Equation (4). Clearly, for T with no duplications $|C(\tilde{T})| = |F(T)|!$.

$$|C(\tilde{T})| = \frac{|F(T)|!}{\prod_{\ell \in \mathsf{labels}(T)} \mathsf{a}(\ell, T)!} \tag{4}$$

5 Results

5.1 Chromosomal Gene Orders Rearranged in Plasmids

The labeling of each internal node of a PQ-tree as P or Q, is learned during the construction of the tree, based on some interrogation of the gene orders from which the PQ-tree is trained [24]. As a result, the set of strings that can be derived from a PQ-tree T, consists of two parts: (1) all the strings representing the known gene orders from which T was constructed, and (2) additional strings, denoted *tree-guided rearrangements*, that do not appear in the set of gene orders constructing T, but can be obtained via rearrangement operations that

are constrained by T. Thus, the tree-guided rearrangements conserve the internal topology properties of the gene cluster, as learned from the corresponding gene orders during the construction of T, such that colinear dependencies among genes and between sub-operons are preserved in the inferred gene orders.

In this section, we used the PQ-trees constructed from chromosomal gene clusters, to examine whether tree-guided rearrangements can be found in plasmids. The objective was to discover gene orders in plasmids that abide by a PQ-tree representing a chromosomal gene cluster, and differ from all the gene orders participating in the PQ-tree's construction. PQ-trees that are constructed from gene clusters that have only one gene order or gene clusters with less than four COGs cannot generate gene orders that differ from the ones participating in their construction. Therefore, only 779 out of 26,270 chromosomal gene clusters were used for the construction of query PQ-trees (the generation of the chromosomal gene clusters is detailed in Section 4). Using our tool PQFinder that implements the algorithm proposed for solving the PQ-Tree Search problem, the query PQ-trees were run against all plasmid genomes. This benchmark was run conservatively without allowing substitutions or deletions from the PQ-tree or from the target string. 380 of the query gene clusters were found in at least one plasmid. The instances of these gene clusters in plasmids are provided in the Supplementary Materials as a session file that can be viewed using the tool CSBFinder-S [36].

Tree-guided rearrangements were found among instances of 29 gene clusters. The PQ-trees corresponding to these gene clusters were sorted by a decreasing S-score, where higher scores are given to a more specific tree (details in Section 4). In this setting, the higher the S-score, the smaller the number of possible gene orders that can be derived from the respective PQ-tree. Interestingly, 21 out of these 29 gene clusters code for transporters, namely 20 importers (ABC-type transport systems) and one exporter (efflux pump). The 10 top ranking results are presented in Table 1.

We selected the third top-ranking PQ-tree in Table 1 for further analysis. This PQ-tree was constructed from seven gene orders of a gene cluster that encodes a heavy metal efflux pump. This gene cluster was found in the chromosomes of 79 genomes (represented by the seven distinct gene orders mentioned above) and in the plasmids of seven genomes. The tree-guided rearrangement instance was found in the strain *Cupriavidus metallidurans CH34*, isolated from an environment polluted with high concentrations of several heavy metals. This strain contains two large plasmids that confer resistance to a large number of heavy metals such as zinc, cadmium, copper, cobalt, lead, mercury, nickel and chromium. We hypothesize that the rearrangement event could have been caused by a heavy metal stress [41]. In the following section we will focus on this PQ-tree to further study its different variants in plasmids.

5.2 RND Efflux Pumps in Plasmids

The heavy metal efflux pump examined in the previous section (corresponding to the third top-ranking PQ-tree in Table 1), was used as a PQFinder query and re-run against all the plasmids in our dataset in order to discover approximate instances of this gene cluster, possibly encoding remotely related variations of the efflux pump it encodes. This time, in order to increase sensitivity, a semantic substitution scoring function (described in Section 4) was used, and the parameters were set to $d_T = 1$ (up to one deletion from the tree, representing missing genes) and $d_S = 3$ (up to three deletions from the plasmid, representing intruding genes). An instance of a gene cluster is accepted if it was derived from the corresponding PQ-tree with a score that is higher than 0.75 of the highest possible score attainable by the query. The plasmid instances detected by PQFinder are displayed in Figure S4.

Table 1 Ten top ranked PQ-trees for which tree-guided rearrangements were found in plasmids. [1]Square brackets represent a Q-node; round brackets represent a P-node. Numbers indicate the respective COG IDs. [2]This column indicates the number of genomes harboring plasmid instances of the respective PQ-tree. The number in brackets indicates the number of genomes harboring a tree-guided gene rearrangement of the corresponding gene cluster. The full table can be found in [45].

	PQ-Tree[1]	S-score	# Genomes[2]	Functional Category
1	[[0683 [[0411 0410] [0559 4177]]] 0583]	22.5	5 (2)	Amino acid transport
2	(1609 [1653 1175 0395] 3839)	10.0	10 (2)	Carbohydrate transport
3	[[1538 [3696 0845]] [0642 0745]]	7.5	7 (1)	Heavy metal efflux
4	[[2115 1070] [4213 [1129 4214]]]	7.5	1 (1)	Carbohydrate transport
5	[1960 [[2011 1135] [2141 1464]]]	7.5	3 (1)	Amino acid transport
6	[[0596 0599] [[3485 3485] 0015]]	7.5	9 (1)	Metabolism
7	[[[1129 1172 1172] 1879] 3254]	7.5	6 (1)	Carbohydrate transport
8	(1609 1869 [[1129 1172] 1879] 0524)	7.5	1 (1)	Carbohydrate transport
9	(0683 [0559 4177] [0411 0410] 0318)	7.5	1 (1)	Amino acid transport
10	(3839 0673 [[0395 1175] 1653])	5.0	10 (1)	Carbohydrate transport

Heavy metal efflux pumps are involved in the resistance of bacteria to a wide range of toxic metal ions [27] and they belong to the resistance-nodulation-cell division (RND) family. In Gram-negative bacteria, RND pumps exist in a tripartite form, comprised from an outer-membrane protein (OMP), an inner membrane protein (IMP), and a periplasmic membrane fusion protein (MFP) that connects the other two proteins. In some cases, the genes of the RND pump are flanked with two regulatory genes that encode the factors of a two-component regulatory system comprising a sensor/histidine kinase (HK) and response regulator (RR) (Figure 2.B). This regulatory system responds to the presence of a substrate, and consequently enhances the expression of the efflux pump genes.

The PQ-tree of this gene cluster (Figure 2.A) shows that the COGs encoding the IMP and MFP proteins always appear as an adjacent pair, the OMP COG is always adjacent to this IMP-MFP pair, and the HK and RR COGs appear as a pair downstream or upstream to the other COGs. COG3696, which encodes the IMP protein, is annotated as a heavy metal efflux pump protein, while the other COGs are common to all RND efflux pumps. Therefore, it is very likely that the respective gene cluster corresponds to a heavy metal RND pump. The absence of an additional periplasmic protein likely indicates that this gene cluster encodes a Czc-like efflux pump that exports divalent metals such as the cobalt, zinc and cadmium exporter in *Cupriavidus metallidurans* [27] (Figure 2.C(1)).

PQFinder discovered instances of this gene cluster in the plasmids of 12 genomes (Figures 2.C(1) and 2.D), and it is significantly enriched in the β-proteobacteria class (hypergeometric p-value= 1.09×10^{-5}, Bonferroni corrected p-value = 1.09×10^{-4}). In addition, three other variants of RND pumps were found as instances of the query gene cluster (Figure 2.C(2-4)). The plasmids of three genomes contained instances that were missing the COG corresponding to the OMP gene CzcC (Figure 2.C(2)). This could be caused by a low quality sequencing or assembly of these plasmids. An alternative possible explanation is that a Czc-like efflux pump can still be functional without CzcC; a previous study showed that the deletion of CzcC resulted in the loss of cadmium and cobalt resistance, but most of the zinc resistance was retained [27].

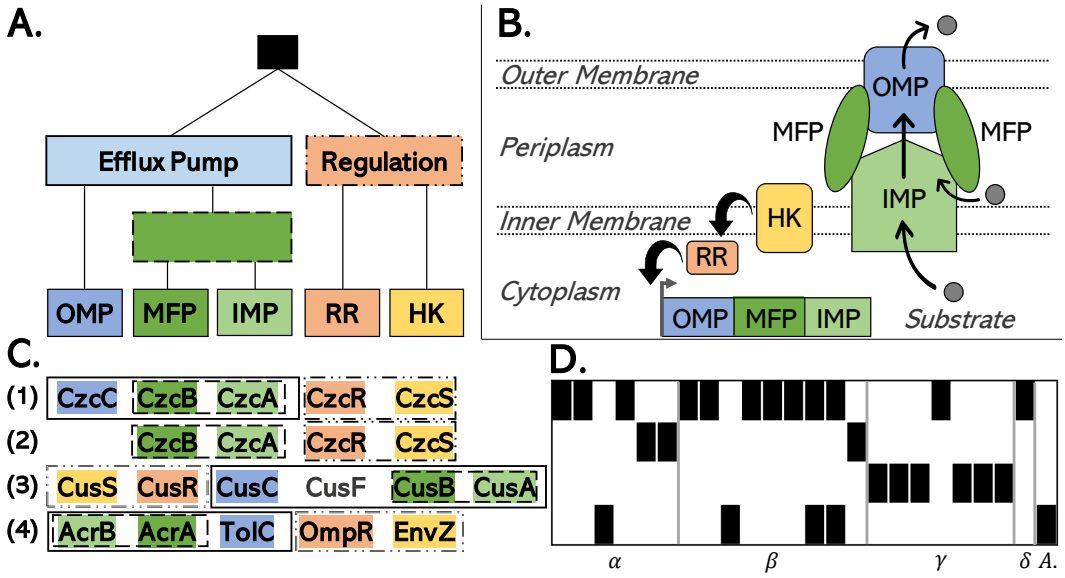


Figure 2 A. A PQ-tree of a heavy metal RND efflux pump, corresponding to the third top scoring result in Table 1. **B.** An illustration of an RND efflux pump consisting of an outer-membrane protein (OMP), an inner membrane protein (IMP), and a periplasmic membrane fusion protein (MFP) that connects the other two proteins. In addition, a two-component regulatory system consisting of a sensor/histidine kinase (HK) and response regulator (RR) enhances the transcription of the efflux pump genes. **C.** Representatives of the three different RND efflux pumps found in plasmids. **(1)** A Czc-like heavy metal efflux pump, **(2)** A Czc-like heavy metal efflux pump with a missing OMP gene, **(3)** A Cus-like heavy metal efflux pump, **(4)** An Acr-like multidrug efflux pump. Additional details can be found in the text. **D.** The presence-absence map of the three types of efflux pumps found in the plasmids of different genomes. The rows correspond to the rows in (C), the columns correspond to the genomes in which instances were found, organized according to their taxonomic classes. A black cell indicates that the corresponding efflux pump is present in the plasmids of the genome. The labels below the map indicate the classes $\alpha, \beta, \gamma, \delta$-Proteobacteria and Acidobacteriia.

Some instances identified by the query, found in the plasmids of six genomes, seem to encode a different heavy metal efflux pump (Figure 2.C(3)). This variant includes all COGs from the query, in addition to an intruding COG that encodes a periplasmic protein (CusF). This protein is a predicted copper usher that facilitates access of periplasmic copper towards the heavy metal efflux pump. Indeed, the genomic region of Cus-like efflux pumps that export monovalent metals, such as the silver and copper exporter in *Escherichia coli*, include this periplasmic protein, in contrast to the Czc-like efflux pump [27]. This variant was found in the plasmids of six bacterial genomes belonging to the class γ-proteobacteria (Figure 2.D). This gene cluster is significantly enriched in the γ-proteobacteria class (hypergeometric p-value= 2.13×10^{-4}, Bonferroni corrected p-value = 2.13×10^{-3}). Surprisingly, all of these strains, except for one, are annotated as human or animal pathogens. Interestingly, previous studies suggest that the host immune system exploits excess copper to poison invading pathogens [18], which can explain why these pathogens evolved copper efflux pumps.

Another variant of the pump, appearing in five genomes (Figures 2.C(4) and 2.D), resulted from a substitution of the query IMP gene (COG3696) by a different IMP gene (COG0841) belonging to the multidrug efflux pump AcrAB-TolC. The AcrAB-TolC system, mainly studied in *Escherichia coli*, transports a diverse array of compounds with little chemical similarity [13]. AcrAB-TolC is an example of an intrinsic non-specific efflux pump, which is

widespread in the chromosomes of Gram-negative bacteria, and likely evolved as a general response to environmental toxins [35]. In this case, the query gene cluster and the identified variant share all COGs, except for the COGs encoding the IMP genes. The differing COGs are responsible for substrate recognition, which naturally differs between the two pumps, as one pump exports heavy metal while the other exports multiple drugs. When considering the functional annotation of these two COGs, we see that the query metal efflux pump COG encoding the IMP gene is annotated as "Cu/Ag efflux pump CusA", while in the multidrug efflux pump the COG encoding the IMP gene is annotated as "Multidrug efflux pump subunit AcrB". Thus, in spite of the difference in substrate specificity, the semantic similarity measure employed by PQFinder was able to reflect their functional similarity and allowed the substitution between them, while conferring to the structure of the PQ-tree.

6 Conclusions

In this paper, we defined a new problem in comparative genomics, denoted PQ-Tree Search. The objective of PQ-Tree Search is to identify approximate new instances of a gene cluster in a new genome S. In our model, the gene cluster is represented by a PQ-tree T, and the approximate instances can vary from the known gene orders by genome rearrangements that are constrained by T, by gene substitutions that are governed by a gene-to-gene substitution scoring function h, and by gene deletions and insertions that are bounded from above by integer parameters d_T and d_S, respectively.

We proved that the PQ-Tree Search problem is NP-hard and proposed a parameterized algorithm that solves it in $O^*(2^\gamma)$ time, where γ is the maximum degree of a node in T and O^* is used to hide factors polynomial in the input size.

The proposed algorithm was implemented as a publicly available tool and harnessed to search for tree-guided rearrangements of chromosomal gene clusters in plasmids. We identified 29 chromosomal gene clusters that are rearranged in plasmids, where the rearrangements are guided by the corresponding PQ-tree. A tree-guided rearrangement event of one of these gene clusters, coding for a heavy metal efflux pump, was detected in a bacterial strain that was isolated from an environment polluted with several heavy metals. Thus, a future extension of this study could explore whether similar gene cluster rearrangement events are correlated with environmental stress or other bacterial adaptations.

The said gene cluster was further analysed to characterize its approximate instances in plasmids. An interesting variant of the analysed gene cluster, found among its approximate instances, corresponds to a copper efflux pump. It was found mainly in pathogenic bacteria, and likely constitutes a bacterial defense mechanism against the host immune response. These results exemplify how our proposed tool PQFinder can be harnessed to find meaningful variations of known biological systems that are conserved as gene clusters, and to explore their function and evolution.

One of the downsides to using PQ-trees to represent gene clusters is that very rare gene orders taken into account in the tree construction could greatly increase the number of allowed rearrangements and thus substantially lower the specificity of the PQ-tree. Thus, a natural continuation of our research would be to increase the specificity of the model by considering a stochastic variation of PQ-Tree Search. Namely, defining a PQ-tree in which the internal nodes hold the probability of each rearrangement, and adjusting the algorithm for PQ-Tree Search accordingly. In addition, future extensions of this work could also aim to increase the sensitivity of the model by taking into account gene duplications, gene-merge and gene-split events, which are typical events in gene cluster evolution.

References

1 Zaky Adam, Monique Turmel, Claude Lemieux, and David Sankoff. Common intervals and symmetric difference in a model-free phylogenomics, with an application to streptophyte evolution. *Journal of Computational Biology*, 14(4):436–445, 2007. doi:10.1089/cmb.2007.A005.

2 Farid Alizadeh, Richard M Karp, Deborah K Weisser, and Geoffrey Zweig. Physical mapping of chromosomes using unique probes. *Journal of Computational Biology*, 2(2):159–184, 1995. doi:10.1089/cmb.1995.2.159.

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(1):4–16, 2007. doi:10.1145/1229968.1229972.

4 Anne Bergeron, Mathieu Blanchette, Annie Chateau, and Cedric Chauve. Reconstructing ancestral gene orders using conserved intervals. In *International Workshop on Algorithms in Bioinformatics*, pages 14–25. Springer, 2004. doi:10.1007/978-3-540-30219-3_2.

5 Anne Bergeron, Sylvie Corteel, and Mathieu Raffinot. The algorithmic of gene teams. In *International Workshop on Algorithms in Bioinformatics*, pages 464–476. Springer, 2002. doi:10.1007/3-540-45784-4_36.

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. Reversal distance without hurdles and fortresses. In *Annual Symposium on Combinatorial Pattern Matching*, pages 388–399. Springer, 2004. doi:10.1007/978-3-540-27801-6_29.

8 Sebastian Böcker, Katharina Jahn, Julia Mixtacki, and Jens Stoye. Computation of median gene clusters. *Journal of Computational Biology*, 16(8):1085–1099, 2009. doi:10.1089/cmb.2009.0098.

9 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. doi:10.1016/S0022-0000(76)80045-1.

10 Thomas Christof, Michael Jünger, John Kececioglu, Petra Mutzel, and Gerhard Reinelt. A branch-and-cut approach to physical mapping of chromosomes by unique end-probes. *Journal of Computational Biology*, 4(4):433–447, 1997. doi:10.1089/cmb.1997.4.433.

11 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

12 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

13 Dijun Du, Zhao Wang, Nathan R James, Jarrod E Voss, Ewa Klimont, Thelma Ohene-Agyei, Henrietta Venter, Wah Chiu, and Ben F Luisi. Structure of the AcrAB–TolC multidrug efflux pump. *Nature*, 509(7501):512–515, 2014. doi:10.1038/nature13205.

14 William G Eberhard. Evolution in bacterial plasmids and levels of selection. *The Quarterly Review of Biology*, 65(1):3–22, 1990. doi:10.1086/416582.

15 Revital Eres, Gad M Landau, and Laxmi Parida. A combinatorial approach to automatic discovery of cluster-patterns. In *International Workshop on Algorithms in Bioinformatics*, pages 139–150. Springer, 2003. doi:10.1007/978-3-540-39763-2_11.

16 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019.

17 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.

18 Yue Fu, Feng-Ming James Chang, and David P Giedroc. Copper transport and trafficking at the host–bacterial pathogen interface. *Accounts of Chemical Research*, 47(12):3605–3613, 2014. doi:10.1021/ar500300n.

19 Lev Gourevitch. A program for pq-tree construction. https://github.com/levgou/pqtrees.

20 Susu He, Michael Chandler, Alessandro M Varani, Alison B Hickman, John P Dekker, and Fred Dyda. Mechanisms of evolution in high-consequence drug resistance plasmids. *mBio*, 7(6):e01987–16, 2016. doi:10.1128/mBio.01987-16.

21 Xin He and Michael H Goldwasser. Identifying conserved gene clusters in the presence of homology families. *Journal of Computational Biology*, 12(6):638–656, 2005. doi:10.1089/cmb.2005.12.638.

22 Steffen Heber and Jens Stoye. Algorithms for finding gene clusters. In *International Workshop on Algorithms in Bioinformatics*, pages 252–263. Springer, 2001. doi:10.1007/3-540-44696-6_20.

23 J Mark Keil. On the complexity of scheduling tasks with discrete starting times. *Operations Research Letters*, 12(5):293–295, 1992. doi:10.1016/0167-6377(92)90087-J.

24 Gad M Landau, Laxmi Parida, and Oren Weimann. Gene proximity analysis across whole genomes via pq trees. *Journal of Computational Biology*, 12(10):1289–1306, 2005. doi:10.1089/cmb.2005.12.1289.

25 William W Metcalf and Barry L Wanner. Evidence for a fourteen-gene, phnC to phnP locus for phosphonate metabolism in escherichia coli. *Gene*, 129(1):27–32, 1993. doi:10.1016/0378-1119(93)90692-V.

26 Kazuo Nakajima and S Louis Hakimi. Complexity results for scheduling tasks with discrete starting times. *Journal of Algorithms*, 3(4):344–361, 1982. doi:10.1016/0196-6774(82)90030-X.

27 Dietrich H Nies. Efflux-mediated heavy metal resistance in prokaryotes. *FEMS Microbiology Reviews*, 27(2-3):313–339, 2003. doi:10.1016/S0168-6445(03)00048-2.

28 Vic Norris and Annabelle Merieau. Plasmids as scribbling pads for operon formation and propagation. *Research in Microbiology*, 164(7):779–787, 2013. doi:10.1016/j.resmic.2013.04.003.

29 Alex Orlek, Nicole Stoesser, Muna F Anjum, Michel Doumith, Matthew J Ellington, Tim Peto, Derrick Crook, Neil Woodford, A Sarah Walker, Hang Phan, et al. Plasmid classification in an era of whole-genome sequencing: application in studies of antibiotic resistance epidemiology. *Frontiers in Microbiology*, 8:182, 2017. doi:10.3389/fmicb.2017.00182.

30 Laxmi Parida. Using pq structures for genomic rearrangement phylogeny. *Journal of Computational Biology*, 13(10):1685–1700, 2006. doi:10.1089/cmb.2006.13.1685.

31 Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975. doi:10.1145/361219.361220.

32 Thomas Schmidt and Jens Stoye. Quadratic time algorithms for finding common intervals in two and more sequences. In *Combinatorial Pattern Matching*, pages 347–358. Springer, 2004. doi:10.1007/978-3-540-27801-6_26.

33 Frits CR Spieksma. On the approximability of an interval scheduling problem. *Journal of Scheduling*, 2(5):215–227, 1999. doi:10.1002/(SICI)1099-1425(199909/10)2:5<215::AID-JOS27>3.0.CO;2-Y.

34 Frits CR Spieksma and Yves Crama. *The complexity of scheduling short tasks with few starting times*. Rijksuniversiteit Limburg. Vakgroep Wiskunde, 1992.

35 Mark C Sulavik, Chad Houseweart, Christina Cramer, Nilofer Jiwani, Nicholas Murgolo, Jonathan Greene, Beth DiDomenico, Karen Joy Shaw, George H Miller, Roberta Hare, et al. Antibiotic susceptibility profiles of escherichia coli strains lacking multidrug efflux pump genes. *Antimicrobial Agents and Chemotherapy*, 45(4):1126–1136, 2001. doi:10.1128/AAC.45.4.1126-1136.2001.

36 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.

37 Roman L Tatusov, Michael Y Galperin, Darren A Natale, and Eugene V Koonin. The cog database: a tool for genome-scale analysis of protein functions and evolution. *Nucleic Acids Research*, 28(1):33–36, 2000. doi:10.1093/nar/28.1.33.

38 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.

39 Takeaki Uno and Mutsunori Yagiura. Fast algorithms to enumerate all common intervals of two permutations. *Algorithmica*, 26(2):290–309, 2000. doi:10.1007/s004539910014.

40 René van Bevern, Matthias Mnich, Rolf Niedermeier, and Mathias Weller. Interval scheduling and colorful independent sets. *Journal of Scheduling*, 18(5):449–469, October 2015. doi:10.1007/s10951-014-0398-5.

41 Joachim Vandecraen, Michael Chandler, Abram Aertsen, and Rob Van Houdt. The impact of insertion sequences on bacterial genome plasticity and adaptability. *Critical Reviews in Microbiology*, 43(6):709–730, 2017. PMID: 28407717. doi:10.1080/1040841X.2017.1303661.

42 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.

43 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. doi:10.1016/j.celrep.2015.12.085.

44 Sascha Winter, Katharina Jahn, Stefanie Wehner, Leon Kuchenbecker, Manja Marz, Jens Stoye, and Sebastian Böcker. Finding approximate gene clusters with gecko 3. *Nucleic Acids Research*, 44(20):9600–9610, 2016. doi:10.1093/nar/gkw843.

45 G. R. Zimerman, D. Svetlitsky, M. Zehavi, and M. Ziv-Ukelson. Approximate search for known gene clusters in new genomes using pq-trees, 2020. arXiv:2007.03589.

A PQ-Tree Search is NP-Hard

In this section we prove Theorem 9 by describing a reduction from the Job Interval Selection problem (JISP) to PQ-Tree Search.

▶ **Theorem 9.** *PQ-Tree Search is NP-hard.*

Since its initial definition by Nakajima and Hakimi [26], JISP has seen several equivalent definitions [23, 33, 34, 40]. We use the following formulation for JISPk based on colors. Given γ k-tuples of intervals on the real line, where the intervals of every k-tuple have a different color i ($1 \leq i \leq \gamma$), select exactly one interval of each color (k-tuple) such that no two intervals intersect. The notation I_j^i is used to denote the interval that starts at s_{ij}, ends at f_{ij} (i.e. the interval $[s_{ij}, f_{ij}]$) and has the color i (i.e. it is a part of the i^{th} k-tuple).

JISP3 was shown to be NP-complete by Keil [23]. Crama et al. [34] showed that JISP3 is NP-complete even if all intervals are of length 2. We use these results to show that PQ-Tree Search is NP-hard.

The Reduction. Given an instance, J, of JISP3 where all intervals have length 2, an instance of PQ-Tree Search is created. It is easy to see that shifting all intervals by some constant does not change the problem. Hence, assume that the leftmost starting interval starts at 1. Let L be the rightmost ending point of an interval, so the focus can be only on the segment $[1, L]$ of the real line. Now, an instance of PQ-Tree Search (T, S, h, d_T, d_S) is constructed (an illustrated example is given in Figure S1 below):

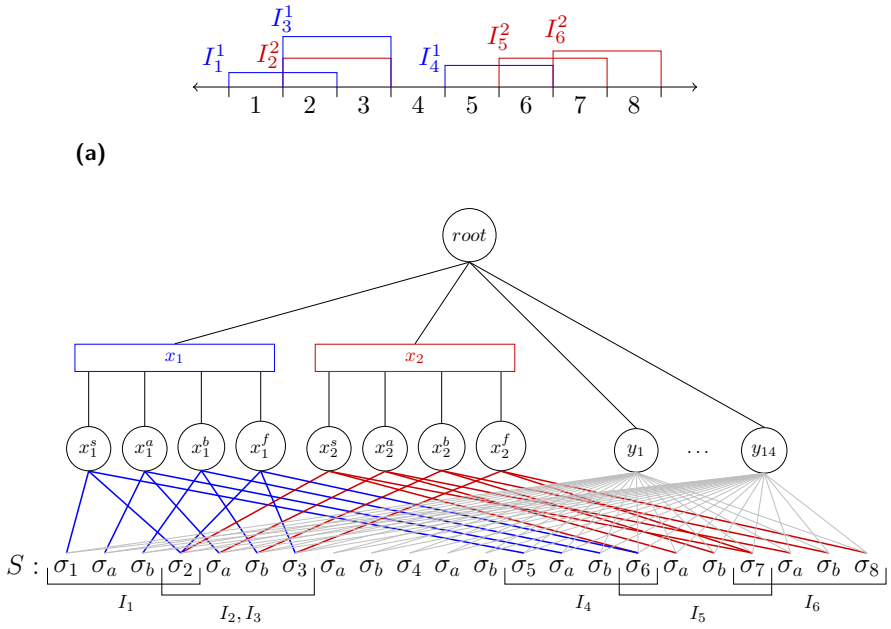


Figure S1 **(a)** The input of the reduction - a JISP3 instance J with intervals of length 2. **(b)** The output of the reduction - a PQ-Tree Search instance (T, S, h, d_T, d_S).

- **The PQ-tree T:** The root node, $root_T$, is a P-node with $3L-2-3\gamma$ children: $x_1, \ldots, x_\gamma$, $y_1, \ldots, y_{3L-2-4\gamma}$. The children of $root_T$ are defined as follows: for every color $1 \leq i \leq \gamma$, create a Q-node x_i with four children x_i^s, x_i^a, x_i^b, x_i^f; for every index $1 \leq i \leq 3L-2-3\gamma$, create a leaf y_i.
- **The string S:** Define $S = \sigma_1\sigma_a\sigma_b\sigma_2\sigma_a\sigma_b \ldots \sigma_a\sigma_b\sigma_L$.
- **The substitution function h:** For every interval of the color i, $I_j^i = [s_{ij}, f_{ij}]$, the function h returns $True$ for the following pairs: $(x_i^s, \sigma_{s_{ij}})$, $(x_i^f, \sigma_{f_{ij}})$, (x_i^a, σ_a) and (x_i^b, σ_b). In addition, every leaf y_r can be substituted by every letter of S, namely for every index $1 \leq r \leq 3L-2-3\gamma$ and for every $s \in \{a, b, 1, \ldots, L\}$ the function h returns $True$ for the pair (y_r, σ_s). For every other pair h returns $False$. For the optimization version of the problem, define a scored substitution function h', such that $h'(u, v) = 1$ if $h(u, v) = True$ and $h'(u, v) = -\infty$ if $h(u, v) = False$.
- **Number of deletions:** Define $d_T = 0$ and $d_S = 0$, i.e. deletions are forbidden from both tree and string.

An example of the reduction is shown in Figure S1. A collection of two 3-tuples (one blue and one red) where each interval is of length 2, i.e a JISP3 instance, is in Figure S1a. Running the reduction algorithm yields the PQ-Tree Search instance in Figure S1b. The pairs that can be substituted (i.e. the pairs for which h returns $True$) are given by the lines connecting the leafs of the PQ-tree and the letters of the string S. The nodes and substitutable pairs created due to the blue and red intervals in the JISP3 instance are marked in blue and red, respectively. The substitutable pairs containing a y node are marked in gray. Note that the colors given in Figure S1b are not a part of the PQ-Tree Search instance, and are given for convenience.

B Time and Space Complexity of the PQ-Tree Search Algorithm

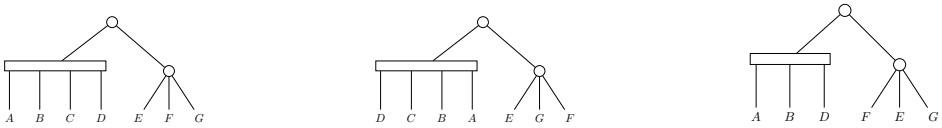
Here we prove Lemma 8.

Proof. The number of leaves in the PQ-tree T is m, hence there are $O(m)$ nodes in the tree, i.e the size of the first dimension of the DP table, $\mathcal{A}$, is $O(m)$. In the algorithm description (Section 3.1) a bound for the possible start indices of substrings derived from nodes in T is given. The node with the largest span in T is the root which has a span of m. The root is mapped to the longest substring when there are d_S deletions from the string. Hence, the size of the second dimension of $\mathcal{A}$ is $\Omega(n-(m+d_S)+1)=\Omega(n)$ (given that $d<m<<n$). The nodes with the smallest spans are the leaves, which have a span of 1, hence the size of the second dimension of $\mathcal{A}$ is $O(n)$. The third and fourth dimensions of $\mathcal{A}$ are of size d_T+1 and d_S+1, respectively. In total, the DP table $\mathcal{A}$ is of size $O(d_T d_S mn)$.

In the initialization step $O(d_T d_S mn)$ entries of $\mathcal{A}$ are computed in $O(1)$ time each. This holds because there are m leaves and n possible start indices for strings of length 1. The d_T and d_S factors come from the initialization of entries with $-\infty$. The P-mapping algorithm is called for every P-node in T and every possible start index i, i.e. the P-mapping algorithm is called $O(nm_p)$ times. Similarly, the Q-mapping algorithm is called $O(nm_q)$ times. Thus, it takes $O(n\ (m_p\cdot\text{Time(P-mapping)}+m_q\cdot\text{Time(Q-mapping)}))$ time to fill the DP table. In the final stage of the algorithm the maximum over the entries corresponding to every combination of deletion number and start index $(0\le k_T\le d_T, 0\le k_S\le d_S, 1\le i\le n-(\mathsf{span}(x)-d_T)+1\})$ is computed. So, it takes $O(d_T d_S n)$ time to find the maximum score of a derivation. Tracing back through the DP table to find the actual mapping does not increase the time complexity.

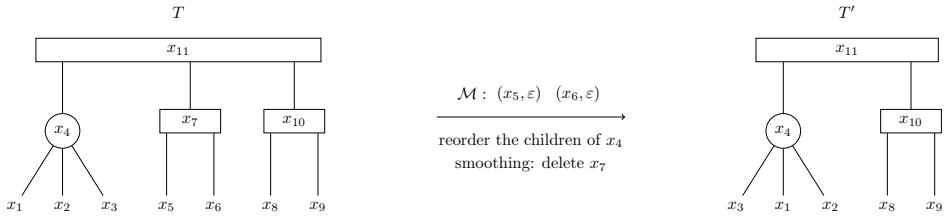
The P-mapping algorithm takes $O(\gamma 2^\gamma {d_T}^2 {d_S}^2)$ time and $O(d_T d_S 2^\gamma)$ space, and the Q-mapping algorithm takes $O(\gamma {d_T}^2 {d_S}^2)$ time and $O(d_T d_S \gamma)$ space. Thus, in total, our algorithm runs in $O(n(m_p\cdot\gamma 2^\gamma {d_T}^2{d_S}^2+m_q\cdot\gamma {d_T}^2{d_S}^2))=O(n\gamma {d_T}^2{d_S}^2(m_p\cdot 2^\gamma+m_q))$ time. Adding to the space required for the main DP table the space required for the P-mapping algorithm (the space needed for the Q-mapping algorithm is insignificant with respect to the P-mapping algorithm) results in a total space complexity of $O(d_T d_S mn)+O(d_T d_S 2^\gamma)=O(d_T d_S(mn+2^\gamma))$. This completes the proof. ◀

C Figures

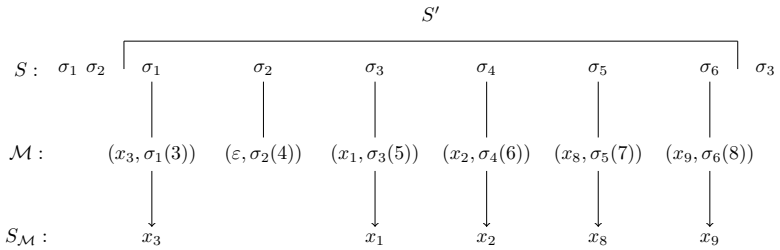


(a) T_1. **(b)** T_2. **(c)** T_3.

Figure S2 Three different PQ-trees. By the definition of frontier, $F(T_1)=ABCDEFG$; $F(T_2)=DCBAEGF$; $F(T_3)=ABDFEG$. T_2 can be obtained from T_1 by reversing the children of a Q-node (the left child of the root) and by reordering the children of a P-node (the right child of the root), so $T_2\equiv T_1$. T_3 can be obtained from T_1 by deleting one leaf and permuting the children of the right child of the root, so $T_1\succeq_1 T_3$. Now, $T_2\succeq_1 T_3$ can be inferred, because the $\equiv$ is an equivalence relation.



(a) The derivation μ applied on T resulting in T': reorder the children of x_4, delete leaves according to $\mathcal{M}$ (delete x_5 and x_6) and perform smoothing (delete x_7, the parent node of x_5 and x_6). The root of T, x_{11}, is the node that μ derives, denoted $\mu.v$. Also, μ is a derivation of x_{11}. The nodes x_5, x_6 and x_7 are deleted under μ. The leaves x_1, x_2, x_3, x_8, x_9 are mapped under μ. The nodes x_4, x_{10}, x_{11} are kept under μ.



(b) The derivation μ on S' resulting in $S_{\mathcal{M}}$: apply substitutions and deletions according to $\mathcal{M}$. The substring $S' = S[3:8]$ is the string that μ derives. The character $S[4]$ is deleted under μ. The characters $S[3], S[5], S[6], S[7], S[8]$ are mapped under μ.

Figure S3 An illustration of the derivation μ from the PQ-tree T to the substring S' under the one-to-one mapping $\mathcal{M}$ ($\mu.o$) with $\mu.del_T = \mathsf{del}_T(\mathcal{M}) = 2$ deletions from the tree and $\mu.del_S = \mathsf{del}_S(\mathcal{M}) = 1$ deletions from the string. The start point of the derivation ($\mu.s$) is 3. The end point of the derivation ($\mu.e$) is 8. Notice that $S_{\mathcal{M}} = F(T')$ and $T \succeq_2 T'$ which means that $S_{\mathcal{M}} \in C_2(T)$.

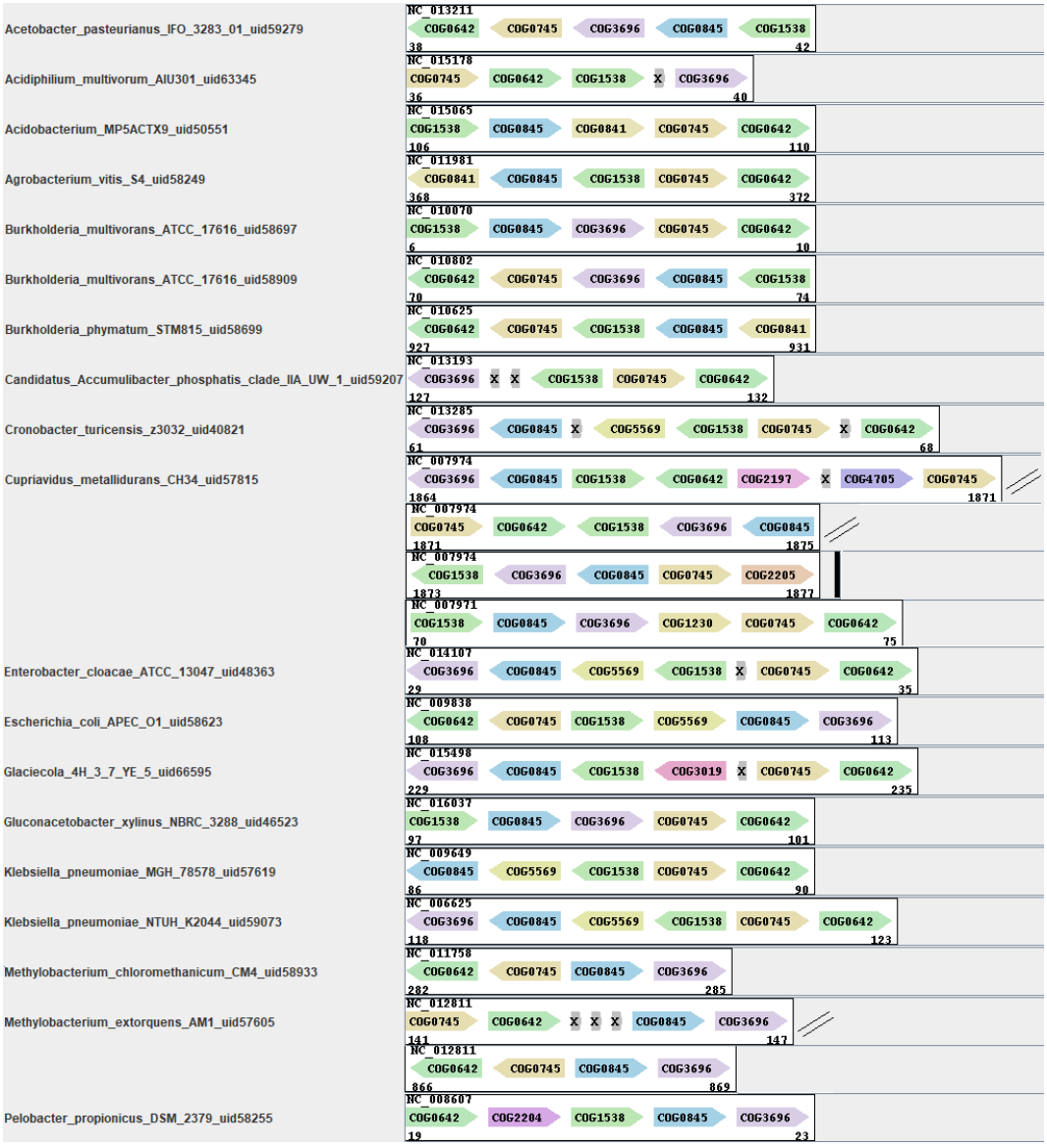
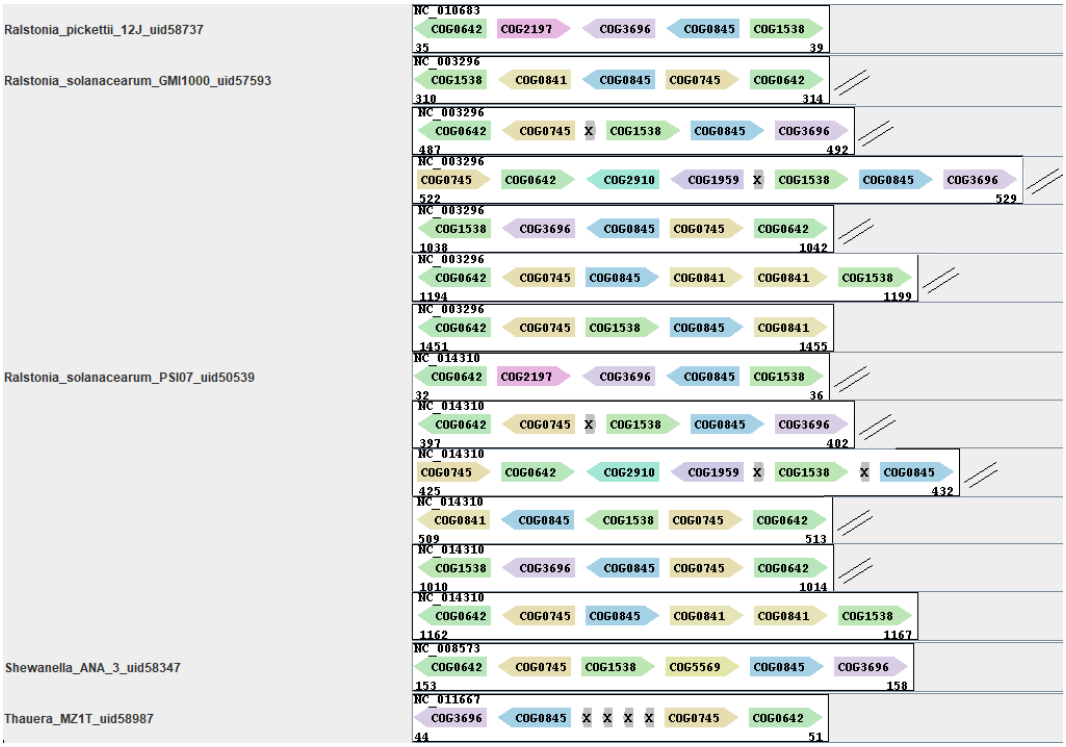


Figure S4 This figure is continued in the next page.



(a)

COG0642 Signal transduction mechanisms | Signal transduction histidine kinase | BaeS
COG0745 Signal transduction mechanisms/Transcription | DNA-binding response regulator, OmpR family, contains REC and winged-helix (wHTH) domain | OmpR
COG3696 Inorganic ion transport and metabolism | Cu/Ag efflux pump CusA | CusA
COG0845 Cell wall/membrane/envelope biogenesis/Defense mechanisms | Multidrug efflux pump subunit AcrA (membrane-fusion protein) | AcrA
COG1538 Cell wall/membrane/envelope biogenesis | Outer membrane protein TolC | TolC
Inserted Genes:
COG3019 Function unknown | Uncharacterized conserved protein |
COG2197 Signal transduction mechanisms/Transcription | DNA-binding response regulator, NarL/FixJ family, contains REC and HTH domains | CitB
COG1230 Inorganic ion transport and metabolism | Co/Zn/Cd efflux system component | CzcD
COG0841 Defense mechanisms | Multidrug efflux pump subunit AcrB | AcrB
COG2205 Signal transduction mechanisms | K+-sensing histidine kinase KdpD | KdpD
COG2910 General function prediction only | Putative NADH-flavin reductase | YwnB
COG1959 Transcription | DNA-binding transcriptional regulator, IscR family | IscR
COG2204 Signal transduction mechanisms | DNA-binding transcriptional response regulator, NtrC family, contains REC, AAA-type ATPase, and a Fis-type DNA-binding domains | AtoC
COG5569 Inorganic ion transport and metabolism | Periplasmic Cu and Ag efflux protein CusF | CusF
COG4705 Function unknown | Uncharacterized membrane-anchored protein |

(b)

Figure S4 (Cont.) **(a)** The plasmid instances of the heavy metal efflux pump gene cluster discussed in Section 5.2. The COGs of the query gene cluster are: COG0642, COG0745, COG3639, COG0845, COG1538. The instances were identified using PQFinder and displayed using the graphical interface of the tool CSBFinder-S [36]. X indicates a gene with no COG annotation. The image was edited to display instances of the same genome in separate lines. **(b)** The functional description of the COGs shown in (a).