Bioinformatics, 36, 2020, i21–i29
doi: 10.1093/bioinformatics/btaa503
ISMB 2020

Discovery of multi-operon colinear syntenic blocks in microbial genomes

Dina Svetlitsky[1], Tal Dagan[2] and Michal Ziv-Ukelson[1,*]

[1]Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel and [2]Institute of Microbiology, Kiel University, Kiel 24118, Germany

*To whom correspondence should be addressed.

Abstract

Motivation: An important task in comparative genomics is to detect functional units by analyzing gene-context patterns. Colinear syntenic blocks (CSBs) are groups of genes that are consistently encoded in the same neighborhood and in the same order across a wide range of taxa. Such CSBs are likely essential for the regulation of gene expression in prokaryotes. Recent results indicate that colinearity can be conserved across multiple operons, thus motivating the discovery of multi-operon CSBs. This computational task raises scalability challenges in large datasets.

Results: We propose an efficient algorithm for the discovery of cross-strand multi-operon CSBs in large genomic datasets. The proposed algorithm uses match-point arithmetic, which is scalable for large datasets of microbial genomes in terms of running time and space requirements. The algorithm is implemented and incorporated into a tool with a graphical user interface, called CSBFinder-S. We applied CSBFinder-S to data mine 1485 prokaryotic genomes and analyzed the identified cross-strand CSBs. Our results indicate that most of the syntenic blocks are exclusively colinear. Additional results indicate that transcriptional regulation by overlapping transcriptional genes is abundant in bacteria. We demonstrate the utility of CSBFinder-S to identify common function of the gene-pair PulEF in multiple contexts, including Type 2 Secretion System, Type 4 Pilus System and DNA uptake machinery.

Availability and implementation: CSBFinder-S software and code are publicly available at https://github.com/dinasv/CSBFinder.

Contact: michaluz@cs.bgu.ac.il

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

An important problem in genomics is the organization of genes in the genomes, and the interpretation of genomic information that is encoded in genomic contexts (Rogozin *et al.*, 2004). In the absence of selective pressure on gene order, successive rearrangement will lead to randomization of gene order (Huynen *et al.*, 2000; Mushegian and Koonin, 1996). Nevertheless, functionally related genes may be constrained to remain close to each other due to natural selection, forming a *syntenic block (SB)*: (Overbeek *et al.*, 1999)—a group of genes that are located close to each other in several different genomes. Here, we study *colinear syntenic blocks (CSBs)*, where a CSB is an SB with a conserved gene order across all its instances.

To compare genes from different microbial genomes, the respective genes need to be annotated using orthology group IDs. Public microbial databases often provide such annotations. For example the Integrated Microbial Genomes database (Chen *et al.*, 2019) provides gene assignments to COG (Tatusov *et al.*, 2000), Pfam (Bateman *et al.*, 2002) and TIGRfam (Selengut *et al.*, 2007) protein families. Large datasets of newly sequenced genomes that are not included in public databases, can be analyzed using tools such as SwiftOrtho (Hu and Friedberg, 2019), SonicParanoid (Cosentino and Iwasaki, 2019), OMA (Altenhoff *et al.*, 2018) or ProteinOrtho (Lechner *et al.*, 2011) for the inference of gene orthology groups.

Selection pressure for the conservation of CSBs can be due to several reasons, including the interaction of the gene products in pathways or complexes (Dandekar *et al.*, 1998; Marsh *et al.*, 2013), common lateral gene transfer events (Danchin *et al.*, 2000), co-localization of the gene transcripts in the cell (Danchin *et al.*, 2000), gene proximity to the Origin of Replication Signal (ORI) and gene co-expression under specific environmental conditions (Rocha, 2008).

The identification of CSBs across different genomes is important for the discovery of new biological systems. Examples are bacterial defense mechanisms (Doron *et al.*, 2018) and functions relevant for host-associated lifestyle (Levy *et al.*, 2018).

CSBs in prokaryotic genomes often correspond to operons; those are neighboring genes that constitute a single unit of transcription and translation. CSBs in prokaryotes can span multiple operons. A recent study showed that the contiguity of the tufB-secE operons in *Salmonella* is essential for the organism's fitness (Brandis *et al.*, 2019). The authors further concluded that the concatenation of the operons, by an inter-operon terminator–promoter overlap, plays a significant role in regulation of gene expression. Notably, the orientation of genes in conserved multi-operons may have implications to gene expression due to overlapping gene regulatory elements [e.g. as in Excludons (Sesto *et al.*, 2013)]. Thus, an approach for the detection of multi-operon CSBs could benefit from integration of gene orientation information. This motivated us to develop a novel bioinformatic approach that facilitates the identification and study of *multi-operon cross-strand CSBs*.

The general SB discovery problem (also denoted 'gene cluster discovery') has been well-formalized and extensively studied on the theoretical front (reviewed in Supplementary Section S1.2). In a nutshell, previous works for SB discovery can be classified into two categories: reference-based approaches versus non-reference-based approaches. Non-reference-based approaches (e.g. Böcker *et al.*, 2009; He and Goldwasser, 2005) seek some ancestral consensus pattern that does not necessarily appear in its exact form as a substring in one of the input sequences. Such approaches suffer from a search space that grows exponentially with increasing input size. Thus, these approaches are not feasible for data mining of thousands of genomes.

To solve this problem, the reference-based gene cluster model was proposed in Jahn (2011). In this model, a gene cluster is not represented by an optimal consensus gene set but rather by a set of genes that appears as a substring of one of the input genomes. This constraint results in a polynomially bounded search space. In Jahn (2011), it was shown that the results obtained by using a reference-based model are highly comparable to the results obtained by non-reference-based approaches. As scalability is one of our main objectives in this article, the model presented here is also reference-based.

There are currently several available tools for the analysis of multiple genomes for the discovery of SBs [GECKO 3 (Winter *et al.*, 2016) and Evolclust (Marcet-Houben and Gabaldón, 2019)], and for the discovery of CSBs [CYNTENATOR (Rödelsperger and Dieterich, 2010), MCScanX (Wang *et al.*, 2012) and i-ADHoRe 3 (Proost *et al.*, 2012)]. All of the aforementioned tools cannot scale up to thousands of microbial genomes in all-versus-all search mode, and would require an infeasible amount of running time or memory consumption. In addition, except for GECKO 3, these tools are lacking a well-defined model for SB discovery; instead, each of these tools indirectly defines an SB through its algorithm, and employs a heuristic search. A comprehensive review of these tools is given in Supplementary Section S1.1.

In a previous work (Svetlitsky *et al.*, 2019), we formalized the CSB discovery optimization problem, as follows. Given an input of m genomes, modelled as strings of gene identifiers and parameters k and q: a CSB is formally defined as a pattern that appears as a substring of at least one of the input genomes, and has instances in at least q of the input genomes, where each instance may vary from the CSB pattern by at most k gene insertions (see Definition 1 in Section 2.1).

Following the CSB discovery problem formalization, we also gave an exact, polynomial time and space algorithm to solve it. In that work, we targeted the data mining of CSBs that encode operons, and enforced this by segmenting the input genomes to directons (consecutive genes encoded on the same strand). Testing the algorithm performance showed that there is a large overlap between the inferred CSBs and experimentally verified operons in *E.coli K-12 str. MG1655*.

However, the time complexity of that algorithm was sensitive to the parameter k by a multiplicative factor, while the space complexity of the algorithm was sensitive to the parameter ℓ, denoting an upper bound on the length of the CSBs. For the purpose of operon detection, small values of k and ℓ make sense. In contrast, in this work our goal is to discover colinear *multi-operon CSBs* that can *span both strands*. This goal requires larger values of k and ℓ that pose a challenge to the scalability of CSB discovery.

Our contribution and roadmap. In this article, we generalize CSB discovery to extract cross-strand multi-operon CSBs. To scale up to this generalization, a novel exact algorithm that uses Match-Point (MP) arithmetic is proposed (Section 2). The time and space complexities of the algorithm are insensitive to the parameters k and ℓ. We show that in practice, the new algorithm is indeed faster than the algorithm given in Svetlitsky *et al.* (2019) for larger values of k (Section 4.1). Additional advantages of the algorithm are its simplicity of implementation, and the fact that it is easily parallelizable, yielding further scalability.

The implementation of the proposed algorithm is incorporated in a publicly available tool, including a graphical user interface, denoted CSBFinder-S. The workflow of the tool is given in Figure 1. CSBFinder-S takes as an input a set of genomes, where each genome is modeled as a sequence of gene identifiers; a gene identifier indicates the corresponding gene orthology group as well as the strand (+/−) in which the gene is encoded (Fig. 1A). The genomes are mined to identify all patterns that qualify as CSBs according to user-specified parameters (Fig. 1B). Next, the discovered CSBs are ranked according to a probabilistic score that is adjusted by the gene content similarity between all the genomes in which the corresponding CSB appears (Fig. 1C). Finally, the CSBs are clustered to families according to their gene content similarity, and the rank of a family is determined by the score of its highest scoring CSB (Fig. 1D).

CSBFinder-S provides several novel mechanisms to help the user sort, filter and interpret the discovered CSBs. One such mechanism is a ranking score that takes into account the gene content similarity between the genomes in which the corresponding CSBs appear (Section 3.7.3). Additional options enable the user to constrain the structural features of the inferred CSBs (length, abundance, etc.), as well as to extract CSBs confined to specific functional semantic categories. The new functions are available via a graphical user interface that also includes a taxonomic viewer of the genomes that contain instances of each CSB (Section 3.7.1).

We applied CSBFinder-S to data mine 1485 prokaryotic genomes and analyzed the identified cross-strand CSBs (Section 4). Our results indicate that most of the SBs are exclusively colinear. Additional results indicate that transcriptional regulation by overlapping transcriptional genes is abundant in bacteria. We demonstrate the utility of CSBFinder-S to identify common function of the gene-pair PulEF in multiple contexts, including Type 2 Secretion System, Type 4 Pilus System and DNA uptake machinery.

2 Algorithm

2.1 Preliminaries and definitions

Let Σ denote a finite set of characters representing gene identifiers, where a gene identifier indicates the corresponding gene orthology group as well as the strand (+/−) in which the gene is encoded. A genome is represented by a string $T = \sigma_1 \ldots \sigma_n$ of concatenated characters, where $\sigma_1, \ldots, \sigma_n \in \Sigma$. For any set ψ, let $|\psi|$ denote the number of members in ψ. For a string P, we use $|P|$ to denote the length of P and $P[i] = \sigma_i$ to denote the ith character of P. For $1 \leq i \leq j \leq |P|$: $P' = P[i \ldots j] = \sigma_i \ldots \sigma_j$ is a substring of P, starting from index i and ending at index j. P' is a proper substring of P if P'

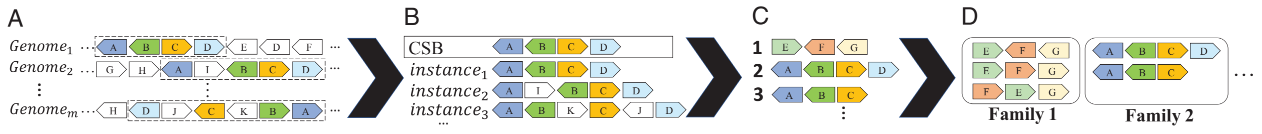

Fig. 1. The proposed workflow for cross-strand CSB discovery and analysis. (**A**) A dataset of input genomes, where each genome is modeled as a sequence of gene identifiers; an identifier is assigned to a gene based on the gene orthology group to which the gene belongs and the strand (+/−) in which the gene is encoded. Genes marked by the same letter belong to the same gene orthology group. (**B**) CSB discovery, where each CSB consists of a pattern (outlined by a rectangle) and its instances. A CSB must have an instance in at least q genomes, and each instance can vary from the pattern by up to k gene insertions, where q and k are user specified parameters. (**C**) Ranking CSBs using a probabilistic similarity-adjusted score. Note that the CSB exemplified in (B) is ranked second in this example. (**D**) Clustering CSBs to families based on gene content and ranking families by their highest scoring CSB

is a substring of P and if $|P'| < |P|$. P' is a suffix of P if $P' = P[i \ldots |P|]$ and P' is a prefix of P if $P' = P[1 \ldots j]$. A string P' is a subsequence of a string P, if $|P'| \geq 1$ and if P can be obtained from P' by inserting zero or more characters to P'.

Given a pattern string P and a string T: An *instance of* P *in* T is a substring T' of T, such that P is a subsequence of T'. Given an integer k: A *k-instance of* P *in* T is an instance T' of P in T, such that $|T'| - |P| \leq k$, i.e. T' can be obtained from P by inserting at most k characters to P. We say that T' is a *minimal k-instance* of P in T, if there is no proper substring of T' that is also a k-instance of P in T.

Example 1 We refer the reader to Figure 2a for an illustration of the concept of a minimal k-instance. For the pattern $P = $ BAC and $k = 3$, $S_2[4 \ldots 8] = $ BAACC is a k-instance of P in S_2, as P is a subsequence of BAACC, and two character insertions are needed to obtain BAACC from P. It is not minimal, as $S_2[4 \ldots 7] = $ BAAC is a k-instance of P in S_2 and a proper substring of BAACC. $S_2[2 \ldots 7] = $ BABAAC is also a k-instance of P in S_2, but it is not minimal as well. $S_2[4 \ldots 7] = $ BAAC (marked in gray) is a minimal k-instance of P in S_2. $S_2[9 \ldots 15] = $ BDAADDC is not a k-instance of P, since $|\text{BDAADDC}| - |\text{BAC}| = 4 > k$.

The pattern discovery problem is defined as follows:

Definition 1 (CSB) *Given a set of strings $S = (S_1, \ldots, S_m)$ over alphabet Σ, and integers k and q. A string P is a CSB if the following conditions hold:*

1. *P is a substring of at least one string from S.*
2. *P has a minimal k-instance in at least q different strings from S.*

The ***Colinear Syntenic Block (CSB) Discovery Problem*** *is to find all strings P s.t. P is a CSB, and to report minimal k-instances for each of the discovered CSBs.*

In this article, we use some notions from MP arithmetic, previously applied to the related problem of computing the pairwise longest common subsequence (Bergroth *et al.*, 2000; Hunt and Szymanski, 1977). Given two strings, P and T, a **match-point** of P versus T is an ordered pair of positions (i, j), such that $P[i] = T[j]$. For a set of strings S and an integer q, define the **match-points of S abiding by q** to be the collection of MPs of S_x versus S_y, accumulated across all pairs of strings $S_x, S_y \in S$ s.t. $x \neq y$ and each MP corresponds to a character in Σ that appears in at least q strings from S (see Supplementary Fig. S1 for an example).

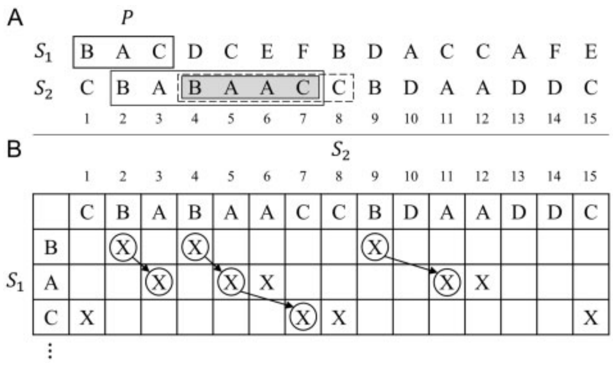
A
P
S_1 B A C D C E F B D A C C A F E
S_2 C B A B A A C C B D A A D D C
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
B
S_2
S_1
X

Fig. 2. (**A**) Pattern $P = $BAC, which is a substring of S_1, and its k-instances in string S_2 for $k = 3$. The k-instances of BAC in S_2 are outlined by rectangles, and the rectangle marking a minimal k-instance is grey-filled. (**B**) Execution of the algorithm on the dataset $S = \{S_1, S_2\}$ with the parameters $k = 3$ and $q = 2$. An 'X' represents a MP of S_1 versus S_2. The MPs are assembled to chains that represent minimal k-instances of a specific pattern. Each chain of the pattern BA is extended, if possible, to form a chain of the pattern BAC. The pattern BAC is then reported as a CSB, as it has instances in both S_1 and S_2. Additional elaboration on this figure is found in the body of the text

An ordered set of MPs forms a chain; given two strings, P and T, a **chain** of P versus T is an ordered set of MPs of P versus T: $((1, s_1), (2, s_2), \ldots, (|P|, s_{|P|}))$ such that $s_1 < s_2 < \ldots < s_{|P|}$.

Since our proposed algorithm uses MP arithmetic, we represent each minimal k-instance by a corresponding chain. However, note that a specific minimal k-instance could be represented by more than one chain.

Example 2 In Figure 2b for pattern $P = BAC$ and $k = 3$, the chains $((1, 4), (2, 5), (3, 7))$ and $((1, 4), (2, 6), (3, 7))$ both represent the minimal k-instance BAAC of P in S_2.

Therefore, we define a canonical representation of the chains corresponding to a minimal k-instance: Given two strings P and T, a chain of P versus T is **canonical** if each prefix of length i of this chain, for $i = 1 \ldots |P|$, corresponds to some minimal k-instance of $P[1 \ldots i]$ in T.

Example 3 In the previously mentioned example, the chain $((1, 4), (2, 5), (3, 7))$ (the second chain shown in Fig. 2b) is canonical, as the chain $((1, 4), (2, 5))$ corresponds to a minimal k-instance of $P[1 \ldots 2]$ (BA) and the chain $((1, 4))$ corresponds to a minimal k-instance of $P[1]$ (B). However, the chain $((1, 4), (2, 6), (3, 7))$ is not canonical, as the chain $((1, 4), (2, 6))$ corresponds to a non-minimal k-instance (BAA) of $P[1 \ldots 2]$ (BA).

The data structures used by the algorithm are *MatchLists* and *NextMatch*, defined as follows. *MatchLists*[1:|Σ|][1: m] is a two-dimensional array, that contains for each $\sigma \in \Sigma$ and for each $S_y \in S$, an ordered list of indices in S_y with the character σ i.e. $S_y[p] = \sigma$ for each $p \in$ MatchLists[σ][y].

For any input string $S_y \in S$, and any index $1 \leq i \leq |S_y|$; let σ denote a character in Σ such that the string $S_y[i]\sigma$ is a substring of some input string in S. Then, NextMatch$_y(i, \sigma)$ is defined to be the smallest index j, $i < j \leq |S_y|$, s.t. $S_y[j] = \sigma$. The requirement for the appearance of $S_y[i]\sigma$ as a substring of one of the input strings is used to reduce the time and space complexity of the algorithm (Supplementary Section S2.2). This requirement is justified by the fact that according to Definition 1, two consecutive MPs can be a part of a chain only if the corresponding characters appear as substring of length two in one of the input strings.

Example 4 In Figure 2, NextMatch$_2(9, A) = 11$ as this is the closest MP with the character A in S_2 to the right of index 9. There is no point in computing NextMatch$_2(9, C)$ during pre-processing, as there is no substring BC in the dataset, and consequently (by Definition 1), there will be no pattern with BC as its substring thus there will be no corresponding chain with these characters. Only the values of NextMatch$_2(9, A)$ and NextMatch$_2(9, D)$ need to be computed.

2.2 A match-point arithmetic-based algorithm

The proposed algorithm consists of a pre-processing stage and a main stage. During the pre-processing stage, the input genomes are annotated with the alphabet Σ to generate the input strings. Based on these input strings, data structures are constructed to support efficient MP arithmetic operations. Then, during the main stage, the MPs of S abiding by q are processed to form chains representing CSB instances. A partial pseudo-code describing the main algorithm is given below. The pseudo-code for the procedure *findInstances* and additional procedures is given in Supplementary Section S2.2. The algorithm is exemplified throughout the text and in Figure 2.

Pre-processing stage. To detect CSBs spanning both strands, the input genomes are pre-processed to generate the set of input strings, as detailed in Section 3.2. Then, the two data structures *Matchlists* and *NextMatch* are constructed from the MPs of S abiding by q (construction details are given in Supplementary Section S2.2).

Algorithm 1 Pseudo-code of the MP arithmetic-based algorithm

```
1: Input: A set of strings $S = (S_1, \dots, S_m)$, integers $k$, $q$
2: Output: All CSBs in $S$ and their minimal k-instances
3: Construct the data structures $MatchLists$ and $NextMatch$
4: for $S_x \in S$ do
5:   for $i \leftarrow 1$ to $|S_x|$ do
6:     for $j \leftarrow i$ to $|S_x|$ do
7:       $P \leftarrow S_x[i \dots j]$
8:       $InstanceList_P \leftarrow$ new list
9:       $count \leftarrow findInstances(P, S_x, InstanceList_P,$
10:                          $MatchLists, NextMatch)$
11:      if $count \geq q$ then
12:        Report $P$ as a CSB and $InstanceList_P$ as its instances
13:      else
14:        Exit internal loop
```

Main stage. During the main stage, the proposed algorithm iterates over all input strings in S, and extracts substrings that are candidate CSBs. Consider, in turn, input string $S_x \in S$. Each index in S_x serves as a starting point for substrings that are potential CSBs. The consideration of candidate CSBs is done by gradually increasing the length of the considered substrings, and checking if they abide by the definition of a CSB and the corresponding user specified parameters (k and q).

In what follows, we describe an algorithm that computes the minimal k-instances of a specific substring $P \in S_x$ in S_y. This algorithm is easily extended to the general CSB finding algorithm by applying it in 'all versus all' mode to the input strings. In the algorithm, a minimal k-instance $S_y[s \dots e]$ of P is represented by a corresponding canonical chain that starts with the MP $(1, s)$ and ends with the MP $(|P|, e)$. In each iteration, we try to extend each chain that represents a minimal k-instance of $P[1 \dots |P| - 1]$ in S_y to a chain that represents a minimal k-instance of P in S_y by augmenting it with an additional MP. A MP can extend a chain, only if it is the closest MP to the right of the chain's end. This MP can be obtained in constant time using the *NextMatch* data structure.

Example 5 We illustrate the extension of a chain using the example in Figure 2. The current pattern is $P = S_1[1 \dots 3] = \text{BAC}$. The chains of the prefix $P[1 \dots 2] = \text{BA}$ are considered for extension by MPs to obtain the chains of P. The first chain consisting of the MPs $((1, 2), (2, 3))$ is not extended; the MP (3, 1) cannot extend the first chain as it is positioned to the left of the end of the chain. Although the MP (3, 7) can possibly extend this chain without exceeding the number of allowed insertions, this extension will create a k-instance that is not minimal, as this MP can also extend the second chain $((1, 4)(2, 5))$. The second chain is extended by the MP (3, 7), as it is the closest MP to the right of the chain's end. The third chain $((1, 9), (2, 11))$ is not extended by the MP (3, 15) because this extension would create a chain of length seven, resulting in $4 > k$ insertions. We refer the reader to Supplementary Section S2.2 for a more formal description of this step.

Time and space complexity analysis. The time complexity of the algorithm is $O(nm + \text{OCC})$, where m denotes the number of input strings, n denotes the average length of an input string, and OCC denotes the sum of lengths of all the surviving chains, corresponding to all CSB instances reported in the output. The space complexity of the algorithm is $O(nm + r)$, where r is the number of MPs of S abiding by q. A comprehensive analysis of time and space complexities can be found in Supplementary Section S2.2. The time complexity analysis is briefly summarized below.

In the pre-processing stage, computing for each character in Σ all the indexes in which it appears and storing them in *MatchLists* takes $O(nm)$ time. Constructing *NextMatch* takes $O(r)$ time.

In the main stage, chains of MPs are computed and reported as CSB instances. Note that a MP is an instance of length one, and if a pattern is not a CSB (i.e. there are less than q input strings encoding instances of this pattern), then there is no point in extending the pattern by an additional character. Hence, all active chains that are maintained during the runtime of the algorithm, represent instances and are part of the output. $\text{OCC} \geq r$, as MPs abiding by the quorum could participate in one or more instances. As a result, the time complexity of the algorithm proposed in this article is $O(nm + \text{OCC})$.

Note that the value of the parameter OCC is output sensitive; in a given experiment, OCC depends on the size and composition of S, as well as on the values assigned to the parameters q and k. Hence, we provide an empirical measurement of OCC on the benchmark dataset used in this study (described in Section 3.1). According to our measurements, the values of OCC ranges between 1.5 nm (for $q = 100$ and $k = 0$) and 11.7 nm (for $q = 10$ and $k = 20$). Additional details are given in Supplementary Section S2.3.

3 Materials and methods

3.1 Dataset

1485 fully sequenced prokaryotic strains with COG ID annotations [Clusters of Orthologous Groups (Tatusov *et al.*, 2000)] were downloaded from GenBank (NCBI; ver 10/2012). A list of all the genomes included in this dataset is provided in the Supplementary Materials.

3.2 Generation of input strings from the dataset

In our string representation, each character in the alphabet is coded as a gene-orthology group followed by a strand specifier, e.g. A(+) denotes a gene belonging to gene orthology group A, and residing on the positive strand. The reverse complement sequence of each input genome is also considered as an extension of the string representing the genome to detect reverse complement instances of a pattern. For example C(+)B(−)A(−) is considered to be an instance of the pattern A(+)B(+)C(−).

3.3 Running time measurements

The Suffix-Tree (ST)-based algorithm (Svetlitsky *et al.*, 2019) and the MP-based algorithm presented in this article are both implemented as part of the CSBFinder-S tool and can be selected by the user. In this experiment, both algorithms were run in cross-strand CSB discovery mode. The average running time of both algorithms on the benchmark dataset (Section 3.1) was measured by executing CSBFinder-S on an Intel Xeon X5680 machine with 192 GB RAM, using the 'time' command in Linux. For each tested value of the parameter k, CSBFinder-S was executed five times in a single threaded mode with the Java option -Xmx190g, and the quorum parameter set to 50. The results are given in Section 4.1.

3.4 Calculation of enrichment in different functional categories

CSBFinder-S was executed on the dataset described in Section 3.1 with the parameters $q = 30$, $k = 0$, and with the length of a CSB constrained to three genes, resulting in 5234 CSBs of length three. Each CSB of length three (or its reverse complement) has one of the following strand order combination patterns: $\rightarrow\rightarrow\rightarrow$, $\leftarrow\rightarrow\rightarrow$, $\rightarrow\leftarrow\leftarrow$ and $\rightarrow\leftarrow\rightarrow$. These patterns are equivalent to their reverse complement strand order combination patterns: $\leftarrow\leftarrow\leftarrow$, $\leftarrow\leftarrow\rightarrow$, $\rightarrow\rightarrow\leftarrow$ and $\leftarrow\rightarrow\leftarrow$, respectively. The strand order combination pattern $\rightarrow\leftarrow\rightarrow$ was excluded due to a limited number of CSBs with this pattern.

The detection of significant enrichment within a specific COG functional category, was performed for the first gene of CSBs having a specific strand combination pattern, using a one-sided hypergeometric test [Python library SciPy (Virtanen *et al.*, 2020)]. In our dataset, the population size is the total number of distinct functional categories collected from all genes participating in any of the extracted CSBs. For a functional category X and a strand order combination pattern Y, the number of successes in the population is the number of genes belonging to the category X, the sample size is the number of distinct categories belonging to any of the genes in the first position of the strand order combination pattern Y, and the number of successes in the sample is the number of genes in the first position of the strand order combination pattern Y belonging to category X.

The hypergeometric test was performed for each COG functional category out of 25 different categories, and for each strand order combination pattern, resulting in a total of 75 P-values. Finally, these P-values were corrected using a two-stage FDR correction (Benjamini and Hochberg, 2000) [Python library Statsmodels (Seabold and Perktold, 2010)]. The results of this analysis are detailed in Section 4.2.

3.5 Detecting SBs using CSBFinder-S

CSBFinder-S was executed on the dataset described in Section 3.1 using parameters $q=1$ and $k=0$, resulting in 595 708 CSBs. This yielded the set of all possible substrings of length two or more that appear in at least one of the genomes and contain only genes that are annotated into orthology groups. Next, ignoring strand and gene order information, CSBs that contain the exact same genes were united to form the generalized set of SBs. The resulting SBs were then filtered to 26 270 SBs that have more than 30 instances. These SBs are analyzed in Section 4.3.

3.6 Exemplifying different functional contexts

CSBFinder-S was executed on the dataset described in Section 3.1 with the parameters $q=30$ and $k=10$. The CSBs were ranked using a ranking score that is adjusted using a taxonomic similarity to overcome a sampling bias (Section 3.7.3). A pre-specified parameter δ bounds the allowed similarity between any two genomes in the dataset in order for them to be considered distinct (see Section 3.7.3 for further details). In this example, the parameter δ was set to 0.7. The entire run in multi-threaded mode took 17 min and 16 s and yielded 133 823 CSBs spanning 13 686 families. The results of this benchmark are given in Section 4.4.

3.7 Additional features of CSBFinder-S

3.7.1 Semantic filters and a taxonomic view

The advantage of using COGs to represent gene orthology groups is that each COG usually has a specific functional description in addition to an association to one of 26 general functional categories. CSBFinder-S allows the user to filter the resulting CSBs by specifying keywords that are required to appear in the functional description of any of their COGs. The functional description of COGs is extracted from an input file that contains information from the COG database (Tatusov *et al.*, 2000). This file also contains common gene IDs of COGs, extracted from the Conserved Domaines Databse (CDD) (Marchler-Bauer *et al.*, 2015). The users can also provide their own file with a functional description of gene orthology groups in their dataset. In addition, if the user provides a file containing a taxonomic annotation (phylum, class, genus and species), a taxonomic view is then displayed for each CSB, showing the genomes containing instances of this CSB (see for example Supplementary Fig. S5).

3.7.2 Multi-threading

As each potential pattern can be processed independently by a different thread, a multi-threading option was incorporated into the implementation of the algorithm. This further accelerates the practical running time of the tool.

3.7.3 Taxonomic similarity-adjusted ranking score

One of the main difficulties in inferring synteny from multiple genomes is the bias incurred by the biased sampling of genomes, i.e. some strains and species are more frequently represented in the dataset than others. This could be handled by normalizing the input sequences to keep only one representative from each distinct taxonomic group, albeit at the cost of reducing the sampling density. Furthermore, different users may prefer to apply different levels of normalization; some users emphasize CSBs that are conserved across a wide taxonomic range, while others focus on CSBs that characterize a narrow taxonomic scope. In the latter case, sample size reduction comes at great expense in sensitivity.

The single-strand CSBs described in Svetlitsky *et al.* (2019), were scored using the assumption that any two genomes from the dataset have a randomized gene order, unless there exists a selection pressure against this randomization. This can result in over-estimation of a ranking score for CSBs that are present mostly in closely related genomes. To overcome sampling bias and allow the user to control the level of redundancy normalization, CSBFinder-S incorporates a measure of divergence between genomes into the ranking score computation. A detailed review of the original score in Svetlitsky *et al.* (2019) can be found in Supplementary Section S3.

In a related work, Junier and Rivoire (2016) inferred pairs of colocalized genes from multiple genomes to study synteny in bacterial genomes. In their computation of the significance score, they reduce the weight of each genome that participates in the score computation proportionally to the number of other genomes to which it is similar.

In this work, we apply a similar approach. In our implementation an inter-genomic similarity measure, denoted D_{ij}, is computed between any two genomes $S_i, S_j \in S$. The similarity measure used here is the Jaccard index of the sets of gene orthology groups of the respective genomes. Here, a similarity of one implies that the two genomes have identical gene orthology groups, whereas if the similarity is 0, the two genomes do not share any gene orthology groups.

The proposed (and implemented) modifications affect parameters m and q_W (described in Supplementary Section S3) in the computation of the ranking score. These two parameters now depend on a pre-specified threshold δ bounding from below the allowed similarity between two genomes in order for them to be considered distinct. First, consider the parameter m which denotes the total number of genomes in the dataset. This number is reduced to an effective number $m' \leq m$ of non-redundant genomes participating in the dataset. m' will be computed as follows:

$$m' = \Sigma_{S_i \in S} \frac{1}{|\{S_j \in S : D_{ij} \geq \delta\}|}. \quad (1)$$

Second, consider the parameter q_W, which denotes, per each CSB W, the number of genomes in which it has an instance. The corrected parameter q'_W will be computed as follows:

$$q'_W = \Sigma_{\{S_i : W \in S_i\}} \frac{1}{|\{S_j : W \in S_j \text{ and } D_{ij} \geq \delta\}|}, \quad (2)$$

where $W \in S_i$ indicates that W has an instance in S_i (respectively, for $W \in S_j$).

Note that δ is a user specified parameter between 0 and 1. If δ is set to 1, the input genomes are considered to be independent, whereas if δ is set to 0 all genomes are considered to be similar.

4 Results

4.1 Scaling up to support large values of k

In Svetlitsky *et al.* (2019), we described and implemented a ST-based algorithm for the discovery of CSBs, which is sensitive to the parameter k (the number of allowed insertions) by a multiplicative factor, in terms of time complexity. Here, we proposed and implemented a MP arithmetic based algorithm that is insensitive to the parameter k. Both algorithms are exact and address the same formally defined computational problem (see Definition 1). Thus, if both algorithms are run with the same genome pre-processing setup and in the same CSB discovery mode (i.e. allowing the consideration of CSBs spanning either one or both strands), the output returned by both algorithms is identical. In this experiment, both algorithms were run in cross-strand CSB discovery mode.

In this section, we give a comparison of the practical running times of the ST based algorithm described in the previous paper versus the MP based algorithm described in the current manuscript (technical details are given in Section 3.3). Our results show that on our benchmark dataset (described in Section 3.1), the MP algorithm indeed outperforms the ST algorithm in terms of practical running time for larger values of the parameter k (Supplementary Table S1). In particular, for $k = 8$, the MP algorithm is ~5% faster than the ST algorithm on average, while for $k = 20$ the MP algorithm is ~38% faster.

Thus, a major advantage of the MP algorithm is its reduced sensitivity to the number of allowed insertions. This feature pays off when seeking multi-operon CSBs, in which large values of k need to be accommodated. For example the CSBs described in Section 4.4 were obtained by running CSBFinder-S with the parameter $k = 10$, allowing the discovery of CSB instances of higher divergence (Supplementary Fig. S2). Notably, the running time of the MP algorithm increases only slightly with k. This slight increase could be due to the constants involved in processing a larger number of CSBs and CSB instances that are discovered when more insertions are allowed.

4.2 Functional analysis of cross-strand CSBs

The enhanced scalability of CSBFinder-S and its ability to identify CSBs spanning both strands makes it possible to run an analysis characterizing CSBs with different strand spanning patterns. CSBs of length three were split to three different strand combination groups, depending on the strand combination pattern of their genes: →→→, →←← or ←→→. Using the hypergeometric test, the enrichment for each functional category in the first position of CSBs, belonging to each strand combination group, was tested (details in Section 3.4). Significant results (P-value ≤ 0.05) were obtained for the ‘Transcription’ functional category for the strand combination pattern ←→→ (FDR corrected P-value $= 1.71 \times 10^{-34}$), and for the ‘Translation, ribosomal structure and biogenesis’ functional category for the strand combination pattern →→→ (FDR corrected P-value $= 0.01$).

The enrichment of transcription related genes in the first position of the strand combination pattern ←→→, corresponds well with the results of a previous analysis of conserved gene pairs in 100 prokaryotic genomes (Korbel *et al.*, 2004). In that study, an analysis of a set of conserved divergently transcribed gene pairs (i.e. gene pairs with the strand pattern ←→) showed that there is strong enrichment of pairs in which one gene encodes a transcriptional regulator, while the other encodes any other class of protein. Here, we generalized this result for longer blocks of consecutive genes.

Another previous related study investigated the interrelationship between transcription regulation and chromosomal organization of transcription units (TUs) in *Escherichia coli*, *Bacillus subtilis* and *Saccharomyces cerevisiae* (Hershberg *et al.*, 2005). Transcription factors that regulate an adjacent TU were found to be very common in *E.coli* and in *B.subtilis* (44 and 42% of all transcription factors in these genomes follow this pattern, respectively), but practically non-existent in the yeast *S.cerevisiae*.

These two aforementioned studies serve as a positive validation of our approach and motivate the utilization of CSBFinder-S to include strand information in CSB detection. Furthermore, our results indicate that transcriptional regulation by overlapping transcriptional genes is abundant in bacteria, as previously suggested for genes arranged in excludons (Sesto *et al.*, 2013).

4.3 Colinearity among syntenic blocks

Having a scalable tool that can data-mine CSBs spanning multiple operons, we can now harness it to re-examine the colinearity of SBs. The definition of an SB is similar to the definition of a CSB (see Definition 1); the only difference is in how an instance is defined: Given a pattern string P and a string T, an instance of P in T is a substring T' of T such that the characters of P constitute a subset of the characters of T'. In others words, unlike CSBs, SBs are not required to be colinearly conserved.

Previous approaches for finding general SBs, with no restrictions on gene order, showed that colinearity is largely conserved across instances of SBs. In one study, SBs were extracted from 133 bacterial genomes and then the percentage of gene pairs that remain adjacent, in each genome where an SB occurs, was computed (Ling *et al.*, 2009). This measure showed that a large proportion of the gene order is preserved across all SBs. In another study (Winter *et al.*, 2016), half of the 65 SBs obtained by searching the genome of *Synechocystis* sp. PCC 6803 against 677 bacterial genomes were shown to be exclusively colinearly conserved. Furthermore, most of the found SBs had a dominant (i.e. most frequent) colinear variant.

In this section, we execute CSBFinder-S on a dataset of 1485 archaeal and eubacterial genomes spanning 31 phyla (Section 3.1). CSBFinder-S was executed with the parameters $k = 0$ and $q = 1$, i.e. the quorum restriction was removed and insertions were prohibited to uncover all possible orders of any set of genes that occurs consecutively in any of these genomes. The resulting CSBs were transformed to SBs by merging all CSBs that share the same gene set into a single SB (details in Section 3.5). We focus on SBs that appear in at least 30 different genomes in our dataset, and distinguish between ‘exclusively ordered SBs’, which are SBs that have only one colinear order, versus the rest of the SBs, denoted hereafter ‘shuffled SBs’. Our goal is to estimate how often SBs are colinearly conserved, and whether exclusively ordered SBs are different from shuffled SBs in terms of their size and their taxonomic distribution.

First, we compute the proportion of exclusively ordered versus shuffled SBs in the computed set of SBs. This analysis reveals that our set of SBs include 20 088 exclusively ordered SBs and only 6182 shuffled SBs. Thus, the frequency of shuffled SBs is substantially smaller than the frequency of exclusively ordered SBs—a staggering 76.5% of the SBs have only one gene order. Our results show that exclusively ordered SBs are much more common than previously thought.

Analyzing the length distributions of exclusively ordered SBs versus shuffled SBs, we find a negative correlation between the proportion of shuffled SBs and their length; almost half of the SBs of length two have at least two gene order variants among their instances (out of four possible order and strand combinations). In comparison, only 3% of the SBs of length seven are shuffled (see Figure 3). Furthermore, the average length of exclusively ordered SBs is 4.91 that is considerably larger than the average length of shuffled SBs (2.61). We note, however, that SBs in our analysis may be dependent due to shared COGs. To test the effect of such dependency on our results, we examined all SBs of length two (9855 SBs), including

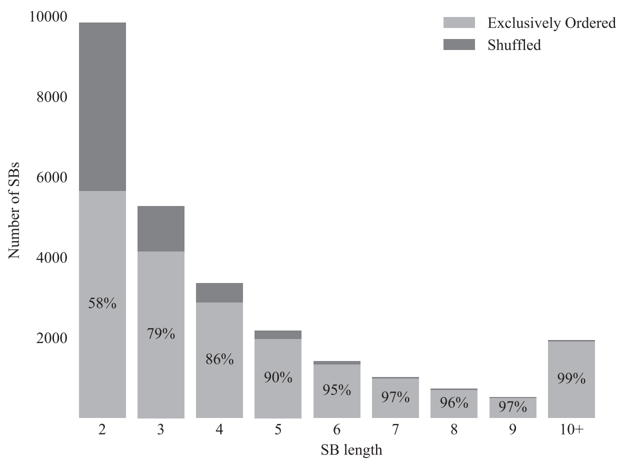

Fig. 3. Frequency of shuffled and exclusively ordered SBs according to SB length

Table 1. The average Shannon index of exclusively ordered versus shuffled SBs

	Phylum	Class	Genus
Exclusively ordered	0.458	0.312	2.278
Shuffled	0.780	1.307	2.841

5675 exclusively ordered SBs and 4180 shuffled SBs. We found that 4008 (70%) of the exclusively ordered SBs are substrings of longer exclusively ordered SBs. Of the shuffled SBs, 1198 (25%) have a colinear variant that is a substring of a longer exclusively ordered SB; in most of these SBs (1137; 95%), this colinear variant is the dominant one among all other order and strand combinations. Thus, our results demonstrate that SBs of length two that are commonly colinear, are likely to be conserved as part of longer colinear SBs.

To quantify the taxonomic diversity of SBs, we analyzed the taxonomic diversity of the genomes containing instances of shuffled SBs versus genomes containing instances of exclusively ordered SBs. We used the Shannon (1948) index to quantify the biodiversity of SBs, denoted by $H = \Sigma p_i \ln(p_i)$, where p_i is the proportion of genomes belonging to the ith taxon in the dataset. The Shannon index was computed for each SB and for three different taxa groups: Phylum, Class and Genus, by considering the genomes containing at least one instance of the corresponding SB. The comparison of average values for all shuffled and exclusively ordered SBs shows that the instances of shuffled SBs are found in more diverse taxa in comparison to exclusively ordered SBs, in all tested taxonomic levels (Table 1). Thus, the taxonomic resolution of the analyzed data is expected to have an effect on the proportion of exclusively ordered versus shuffled SBs; for example analysis of specific taxonomic groups would yield a higher proportion of exclusively ordered SBs.

4.4 Contextual analysis of CSBs

In this section, we exemplify how the functional semantic filter implemented in CSBFinder-S, coupled with the data-mining algorithm, can be utilized for contextual and evolutionary analysis of CSB families. CSBFinder-S was used to analyze the dataset of 1485 chromosomal genomes (Section 3.1) to obtain CSBs spanning both strands (details in Section 3.6). We focus our example on secretion systems, as they play a critical role in the evolution of bacterial virulence (Green and Mecsas, 2016).

In Section 4.3 we observed that many SBs of length two have more than one colinear variant. However, in most of these SBs, the dominant variant is a substring of a longer exclusively ordered SB. The most common shuffled SB of length two that has genes related to secretion, consists of the genes PulE (COG2804) and PulF (COG1459). In CSBFinder-S execution without insertions, there are two CSBs consisting of this gene pair: PulE(+)PulF(+) has 745 instances, and PulF(+)PulE(+) has 41 instances (see Supplementary Fig. S3). The dominant CSB is a substring of longer exclusively colinear CSBs, while the second CSB is not a substring of any other CSBs. This motivated us to explore longer CSBs that contain the PulE(+)PulF(+) CSB.

To examine different functional contexts of PulEF, we executed CSBFinder-S with up to 10 insertions. Allowing a large number of insertions revealed CSB instances with higher divergence (see Supplementary Fig. S2 for an example). The results were filtered for CSBs containing the PulEF COG IDs and then re-clustered into families using the CSBFinder-S user interface (Supplementary Table S2). Figure 4 illustrates seven of the resulting CSBs selected to exemplify three distinct contexts: Type II Secretion System (T2SS), Type IV Pilus system (T4PS) and DNA uptake machinery. Genes that function in these contexts share several structural and functional features, and were previously shown to be evolutionary related (Giltner et al., 2012; Peabody et al., 2003).

Homologous proteins shared across these three contexts include proteins forming filamentous structures (e.g. pillins and pseudopillins), the cytoplasmic ATPases (e.g. pulE and its homologs) and the membrane proteins (e.g. pulF and its homologs). A common ancestry has been previously suggested for the T2SS, T4PS and the Archaeal Flagella, based on their homologous ATPases and their homologous transmembrane proteins (Peabody et al., 2003). Note that the user interface of CSBFinder-S enables to display a detailed taxonomic distribution of the instances of each CSB (Supplementary Figs S5–S8). Furthermore, the different orientation of genes in CSBs 5, 6 and 7 indicates that these CSBs likely include multiple operons. In what follows, we elaborate on the different functional contexts of PulE(+)PulF(+) CSB.

T2SS context: CSBs 1 and 2 in Figure 4 contain genes from the Type II Secretion System (T2SS), and in particular, the second CSB contains most of the known T2SS genes. The T2SS is a membrane spanning secretion system composed of 12–15 different secretory pathway proteins that secrete a wide variety of folded exoproteins (Korotkov et al., 2012). The order of T2SS genes is typically well conserved, where variations in gene content in different species are usually found at the 5′ and 3′ ends of the gene cluster (Sandkvist, 2001). The first CSB we present in Figure 4, was detected in alpha-, beta-, gamma- and delta-proteobacteria, while the second CSB was found only in gamma-proteobacteria. T2SSs contain five pilins (PulGHIJK) needed for secretion; PulG is the major pseudo-pilin, while the other four are minor pseudo-pilins (Possot et al., 2000). The three copies of pulG (COG2165) in the second CSB correspond to homologs of pulG, pulH and PulI, which are classified into the same COG ID due to sequence similarity.

DNA uptake context: CSBs 3–5 in Figure 4 exemplify the occurrence of pulEF in the context of natural competence (DNA uptake). These CSBs include COG4537, a gene homologous to ComGC, that functions in DNA uptake in Gram-positive bacteria (Muschiol et al., 2015). Indeed, these three CSBs are found exclusively in Gram-positive strains; CSB 4 is found in the genera *Bacillus*, *Geobacillus* and *Staphylococcus*, whereas the other two CSBs are *Streptococcus*-specific. Note that PulE and PulF in these CSBs are termed in these organisms ComGA and ComGB, respectively (classified into the same COG IDs).

Examining the instances of CSB 4 reveals that it has 10 instances in *Bacillus* strains with 7–10 gene insertions, 2 instances in *Geobacillus* strains with 5 gene insertions and 19 instances in *Staphylococcus* strains without any gene insertions (see Supplementary Fig. S2). Thus, setting the parameter k to 10 (i.e.

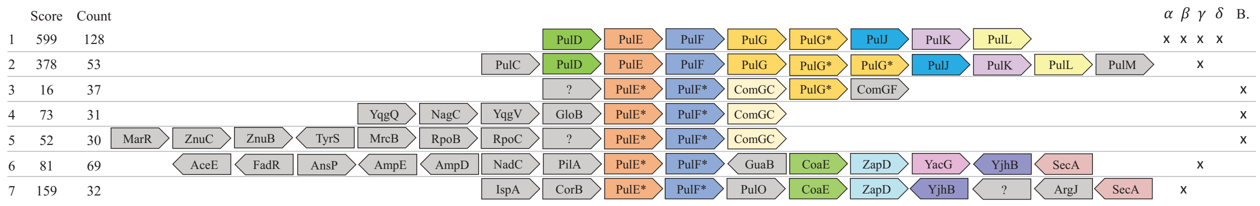

Fig. 4. Seven of the CSBs obtained after filtration using the keywords PulE and PulF. The COGs in the figure were annotated with gene IDs using the CDD database (Marchler-Bauer et al., 2015). The taxonomic classes in which each CSB has an instance are indicated. $\alpha - \delta$ are the classes of proteobacteria, B. stands for the class Bacilli. COGs are color coded, except for COGs appearing in only one CSB that are colored in gray. A question mark denotes an uncharacterized gene. Note that the instances of the CSBs shown in this figure can have up to 10 gene insertions. To exemplify this, some of the instances of CSB 4 are shown in Supplementary Figure S2. *Gene names may be ambiguous, see text

enabling up to 10 insertions) revealed further instances of this CSB in strains where the gene synteny is less conserved (typically distantly related strains). In addition, as the instance quorum was set to 30 in this benchmark, if less than 10 insertions were allowed this CSB would not have passed the quorum and thus would not have been detected.

T4P context: CSBs 6 and 7 have many genes in common and were clustered by CSBFinder-S to the same CSB family. The surrounding context of CSB 6 seems to indicate that it functions in the type IV pilus biogenesis (T4P) system, while the context of CSB 7 is quite puzzling as it does not include pilus genes. The T2SS and the T4P pathways share many homologous proteins, suggesting a common evolutionary origin (Nunn, 1999). Well characterized roles of T4P systems include adherence to natural surfaces, twitching motility, modulation of biofilm architecture, DNA uptake (competence) and transfer (conjugation), secretion of exoproteins and bacteriophage susceptibility (Giltner *et al.*, 2012). Note that PulE and PulF in these CSBs are termed in these organisms PilF and PilG, respectively (classified into the same COG IDs).

The T4P gene PilA, as well as ComGC in the DNA uptake system and PulG in the T2SS, are type IV pilin proteins with a distinct N-terminal signal sequence (Giltner *et al.*, 2012). Proteins with this unique signal sequence are called prepilins and are not competent for assembly until the signal is cleaved, using prepilin peptidases (PilD/PulO), that is included in CSB 7. The protein SecA, an ATPase that is present in both CSBs 6 and 7, is a primary component of the Sec pathway that inserts prepillins into the cytoplasmic membrane. It is conceivable that the co-expression of PilA and SecA in CSB 6 may be important for Sec-mediated translocation of the prepilin PilA.

Accessory virulence genes across different contexts: CSBs 4, 5 and 6 in Figure 4 illustrate multi-operon context including genes that play a role in resistance to antimicrobial substances. These genes span CSBs from two distinct functions (DNA uptake versus T4P) and a wide taxonomic range (both Gram-positive and Gram-negative bacteria, correspondingly). CSB 4 contains a GloB-homolog—a predicted glyoxylase that is part of a methylglyoxal degradation pathway; it has been previously shown that methylglyoxal has an antibacterial activity against *Staphylococcus aureus* (Jervis-Bardy *et al.*, 2011). The resistance factor found in CSB 5 is an MrcB-homolog—a penicillin binding protein. The presence of antimicrobial genes in the neighborhood of DNA uptake genes may be related to the functional link between antibiotic stress response and the induction of natural competence (Prudhomme *et al.*, 2006).

Genes associated with antimicrobial resistance are also found in CSB 6 (in the predicted context of Pilus Assembly in gamma-proteobacteria): this CSB contains the operon AmpDE, which plays a role in the regulation of the AmpC gene encoding a beta-lactamase. AmpE is a signal transduction inner-membrane protein, predicted to be activated in the presence of beta-lactam (Honoré *et al.*, 1989), while AmpD is the predicted repressor of AmpC. The longest CSB that includes PulEF comprises 36 genes, among which are AmpE and AmpD (Supplementary Fig. S4, see details in Section S4.5).

5 Discussion

We propose an efficient algorithm for the discovery of cross-strand, multi-operon CSBs in large genomic datasets. The proposed algorithm uses match-point arithmetic and is thus both time and space scalable. The new algorithm is insensitive to the number of allowed insertions and to the length of the sought CSBs, thus it enables the discovery of CSBs spanning multiple operons. The algorithm was implemented and incorporated into a tool with a graphical user interface, denoted CSBFinder-S.

CSBFinder-S differs from our previously released tool for CSB discovery (Svetlitsky *et al.*, 2019) by several major contributions. Mainly, it is more general; while in the previous version of the tool CSBs were confined to directons (consecutive genes encoded on the same strand), the new tool can also detect cross-strand multi-operon CSBs. This generalization is scalable due to the newly proposed algorithm (Section 2), and due to its multi-threaded implementation (Section 3.7.2). In addition, CSBFinder-S incorporates a measure of divergence between the input genomes to overcome sampling bias in the ranking score computation (Section 3.7.3).

Furthermore, CSBFinder-S offers new functions to help the user navigate through the database of discovered CSBs. These include filters that enable the user to constrain the structural features of the inferred CSBs (length, abundance, etc.), as well as to extract CSBs confined to specific functional semantic categories (exemplified in Section 4.4). The user interface also includes a taxonomic viewer of the genomes in which CSB instances appear (see for example Supplementary Fig. S5), and the option to re-cluster the found CSBs into families according to user-specified parameters.

Notably, despite extensive shuffling of prokaryotic genomes, most SBs in our dataset were exclusively ordered, also when considering cross-strand CSBs that can span multiple operons. The CSBs we uncovered with CSBFinder-S likely represent functional units whose expression is tightly coordinated and their gene order evolves under a strong purifying selection (Koonin, 2009). The analysis of conserved colinear gene orders is instrumental for functional studies of microbial organisms.

Funding

This study was partially supported by the European Research Council [281357 to T.D.]; and the Israel Science Foundation [179/14 and 939/18 to D.S. and M.Z.-U.].

Conflict of Interest: none declared.

References

Altenhoff,A.M. *et al.* (2018) The OMA orthology database in 2018: retrieving evolutionary relationships among all domains of life through richer web and programmatic interfaces. *Nucleic Acids Res.*, **46**, D477–D485.

Bateman,A. *et al.* (2002) The Pfam protein families database. *Nucleic Acids Res.*, **30**, 276–280.

Benjamini,Y. and Hochberg,Y. (2000) On the adaptive control of the false discovery rate in multiple testing with independent statistics. *J. Educ. Behav. Stat.*, **25**, 60–83.

Bergroth,L. *et al.* (2000) A survey of longest common subsequence algorithms. In *Proceedings of Seventh International Symposium on String Processing and Information Retrieval, 2000. SPIRE 2000*. IEEE, pp. 39–48. Piscataway, New Jersey, USA.

Böcker,S. *et al.* (2009) Computation of median gene clusters. *J. Comput. Biol.*, **16**, 1085–1099.

Brandis,G. *et al.* (2019) Operon concatenation is an ancient feature that restricts the potential to rearrange bacterial chromosomes. *Mol. Biol. Evol.*, **36**, 1990–2000.

Chen,I.-M.A. *et al.* (2019) IMG/M v. 5.0: an integrated data management and comparative analysis system for microbial genomes and microbiomes. *Nucleic Acids Res.*, **47**, D666–D677.

Cosentino,S. and Iwasaki,W. (2019) SonicParanoid: fast, accurate and easy orthology inference. *Bioinformatics*, **35**, 149–151.

Danchin,A. *et al.* (2000) Mapping the bacterial cell architecture into the chromosome. *Philos. Trans. R. Soc. Lond. B Biol. Sci.*, **355**, 179–190.

Dandekar,T. *et al.* (1998) Conservation of gene order: a fingerprint of proteins that physically interact. *Trends Biochem. Sci.*, **23**, 324–328.

Doron,S. *et al.* (2018) Systematic discovery of antiphage defense systems in the microbial pangenome. *Science*, **359**, eaar4120.

Giltner,C.L. *et al.* (2012) Type IV pilin proteins: versatile molecular modules. *Microbiol. Mol. Biol. Rev.*, **76**, 740–772.

Green,E.R. and Mecsas,J. (2016) Bacterial secretion systems—an overview. *Microbiol. Spectr.*, **4**, 213–239.

He,X. and Goldwasser,M.H. (2005) Identifying conserved gene clusters in the presence of homology families. *J. Comput. Biol.*, **12**, 638–656.

Hershberg,R. *et al.* (2005) Chromosomal organization is shaped by the transcription regulatory network. *Trends Genet.*, **21**, 138–142.

Honoré,N. *et al.* (1989) Regulation of enterobacterial cephalosporinase production: the role of a membrane-bound sensory transducer. *Mol. Microbiol.*, **3**, 1121–1130.

Hu,X. and Friedberg,I. (2019) SwiftOrtho: a fast, memory-efficient, multiple genome orthology classifier. *GigaScience*, **8**, giz118.

Hunt,J.W. and Szymanski,T.G. (1977) A fast algorithm for computing longest common subsequences. *Commun. ACM*, **20**, 350–353.

Huynen,M. *et al.* (2000) Exploitation of gene context. *Curr. Opin. Struct. Biol.*, **10**, 366–370.

Jahn,K. (2011) Efficient computation of approximate gene clusters based on reference occurrences. *J. Comput. Biol.*, **18**, 1255–1274.

Jervis-Bardy,J. *et al.* (2011) Methylglyoxal-infused honey mimics the anti-*Staphylococcus aureus* biofilm activity of manuka honey: potential implication in chronic rhinosinusitis. *Laryngoscope*, **121**, 1104–1107.

Junier,I. and Rivoire,O. (2016) Conserved units of co-expression in bacterial genomes: an evolutionary insight into transcriptional regulation. *PLoS One*, **11**, e0155740.

Koonin,E.V. (2009) Evolution of genome architecture. *Int. J. Biochemist. Cell Biol.*, **41**, 298–306.

Korbel,J.O. *et al.* (2004) Analysis of genomic context: prediction of functional associations from conserved bidirectionally transcribed gene pairs. *Nat. Biotechnol.*, **22**, 911–917.

Korotkov,K.V. *et al.* (2012) The type ii secretion system: biogenesis, molecular architecture and mechanism. *Nature Reviews Microbiology*, **10**, 336–351.

Lechner,M. *et al.* (2011) Proteinortho: detection of (co-)orthologs in large-scale analysis. *BMC Bioinformatics*, **12**, 124.

Levy,A. *et al.* (2018) Genomic features of bacterial adaptation to plants. *Nat. Genet.*, **50**, 138–150.

Ling,X. *et al.* (2009) Detecting gene clusters under evolutionary constraint in a large number of genomes. *Bioinformatics*, **25**, 571–577.

Marcet-Houben,M. and Gabaldón,T. (2019) EvolClust: automated inference of evolutionary conserved gene clusters in eukaryotes. *Bioinformatics*, **36**, 1265–1266.

Marchler-Bauer,A. *et al.* (2015) CDD: NCBI's conserved domain database. *Nucleic Acids Res.*, **43**, D222–D226.

Marsh,J.A. *et al.* (2013) Protein complexes are under evolutionary selection to assemble via ordered pathways. *Cell*, **153**, 461–470.

Muschiol,S. *et al.* (2015) Uptake of extracellular DNA: competence induced pili in natural transformation of streptococcus pneumoniae. *Bioessays*, **37**, 426–435.

Mushegian,A.R. and Koonin,E.V. (1996) A minimal gene set for cellular life derived by comparison of complete bacterial genomes. *Proc. Natl. Acad. Sci. USA*, **93**, 10268–10273.

Nunn,D. (1999) Bacterial type II protein export and pilus biogenesis: more than just homologies? *Trends Cell Biol.*, **9**, 402–408.

Overbeek,R. *et al.* (1999) The use of gene clusters to infer functional coupling. *Proc. Natl. Acad. Sci. USA*, **96**, 2896–2901.

Peabody,C.R.,Jr. *et al.* (2003) Type II protein secretion and its relationship to bacterial type IV pili and archaeal flagella. *Microbiology*, **149**, 3051–3072.

Possot,O.M. *et al.* (2000) Multiple interactions between pullulanase secreton components involved in stabilization and cytoplasmic membrane association of pule. *J. Bacteriol.*, **182**, 2142–2152.

Proost,S. *et al.* (2012) i-ADHoRe 3.0—fast and sensitive detection of genomic homology in extremely large data sets. *Nucleic Acids Res.*, **40**, e11–e11.

Prudhomme,M. *et al.* (2006) Antibiotic stress induces genetic transformability in the human pathogen *Streptococcus pneumoniae*. *Science*, **313**, 89–92.

Rocha,E.P. (2008) The organization of the bacterial genome. *Annu. Rev. Genet.*, **42**, 211–233.

Rödelsperger,C. and Dieterich,C. (2010) CYNTENATOR: progressive gene order alignment of 17 vertebrate genomes. *PLoS One*, **5**, e8861.

Rogozin,I.B. *et al.* (2004) Computational approaches for the analysis of gene neighbourhoods in prokaryotic genomes. *Brief. Bioinf.*, **5**, 131–149.

Sandkvist,M. (2001) Type II secretion and pathogenesis. *Infect. Immun.*, **69**, 3523–3535.

Seabold,S. and Perktold,J. (2010) Statsmodels: econometric and statistical modeling with Python. In: *Proceedings of the 9th Python in Science Conference*, p. 61. Austin, Texas.

Selengut,J.D. *et al.* (2007) TIGRFAMs and genome properties: tools for the assignment of molecular function and biological process in prokaryotic genomes. *Nucleic Acids Res.*, **35**, D260–D264.

Sesto,N. *et al.* (2013) The excludon: a new concept in bacterial antisense RNA-mediated gene regulation. *Nat. Rev. Microbiol.*, **11**, 75–82.

Shannon,C.E. (1948) A mathematical theory of communication. *Bell Syst. Tech. J.*, **27**, 379–423.

Svetlitsky,D. *et al.* (2019) CSBFinder: discovery of colinear syntenic blocks across thousands of prokaryotic genomes. *Bioinformatics*, **35**, 1634–1643.

Tatusov,R.L. *et al.* (2000) The COG database: a tool for genome-scale analysis of protein functions and evolution. *Nucleic Acids Res.*, **28**, 33–36.

Virtanen,P. *et al.* (2020) SciPy 1.0—fundamental algorithms for scientific computing in Python. *Nat. Methods,* **17**, 261–272.

Wang,Y. *et al.* (2012) MCScanX: a toolkit for detection and evolutionary analysis of gene synteny and collinearity. *Nucleic Acids Res.*, **40**, e49–e49.

Winter,S. *et al.* (2016) Finding approximate gene clusters with GECKO 3. *Nucleic Acids Res.*, **44**, 9600–9610.