Constrained Gene Block Discovery and Its Application to Prokaryotic Genomes

JONATHAN ENGEL, ISANA VEKSLER-LUBLINSKY, and MICHAL ZIV-UKELOSON

ABSTRACT

Recent advances in Next Generation Sequencing techniques, combined with global efforts to study infectious diseases, yield huge and rapidly-growing databases of microbial genomes. These big new data statistically empower genomic-context based approaches to functional analysis: the idea is that groups of genes that are clustered locally together across many genomes usually express protein products that interact in the same biological pathway (e.g., operons). The problem of finding such conserved “gene blocks” in a given genomic data has been studied extensively. In this work, we propose a new gene block discovery problem variant: find conserved gene blocks abiding by a user specification of biological functional constraints. We take advantage of the biological constraints to efficiently prune the search space. This is achieved by modeling the new problem as a special constrained variant of the well-studied “Closed Frequent Itemset Mining” problem, generalized here to handle item duplications. We exemplify the application of the tool we developed for this problem with two different case studies related to microbial ATP (adenosine triphosphate)-binding cassette (ABC) transporters.

Keywords: ABC transporters, conserved gene blocks, gene block discovery, gene teams, itemset mining.

1. INTRODUCTION

Recent advances in Next Generation Sequencing (NGS) technology have made microbial sequencing highly accessible: sequences are now regularly collected during routine monitoring and in clinical studies. Many new NGS databases are forming, thus increasing the amount of publicly available bacterial sequences (Kulikova et al., 2006; Benson et al., 2012; Wattam et al., 2013). To date, complete genome sequences of almost all major bacterial pathogens have been determined, providing significant insights into microbial pathogenesis and the mechanism of drug resistance. Sequences of bacteria that are not pathogenic to human, such as microbiome data, are also collected and deposited in sequence databases (Chen et al., 2010, 2016; Gevers et al., 2012). Altogether, the available number of sequenced bacterial genomes is in the range of hundreds of thousands and is growing rapidly (Mashima et al., 2015; Wattam et al., 2016).

The increasing availability of completely sequenced genomes opens up new possibilities for large-scale data mining approaches to identify conserved gene organizations in prokaryotic genomes and empowers
the use of gene neighborhood context as a source for predicting associations among gene products
(Moreno-Hagelsieb, 2015). In bacteria and archaea and, incidentally, in eukaryotes (Blumenthal, 1998),
genomes are transcribed into polycistronic messenger RNAs (mRNAs) encoding co-regulated and functionally
related genes. In the absence of selective pressure on gene organization, successive rearrangement will lead
to randomization of gene order and proximity. Therefore, the conservation of groups of genes within the
same local genomic proximity (neighborhood) across the genomes of many remote species is a source of
evidence for inferring evolutionary relationships and/or for predicting the functional roles of the genes.

A gene block is a group of genes that are consistently clustered within the same genomic vicinity across a set
of genomes, regardless of the internal order and orientation of the genes from the block in each genome. The
clustered occurrences of these genes within the corresponding genomes are called instances. The number
of genomes in which a gene block has an instance is denoted by the instance support of the gene block.

Gene blocks are biologically related to operons as follows: Operons are defined as sets of genes that are
co-transcribed into a single mRNA sequence (Jacob and Monod, 1961). In contrast, our computationally
predicted gene blocks could contain genes spanning both strands of the DNA and could consist of one or
more operons or alternatively just a part of an operon.

Identification of gene blocks that are conserved across different genomes (i.e., abiding by a prespecified
instance support requirement) is important for several problems in comparative genomics (Rogozin et al.,
2004): first, to predict operons or functional modules (Westover et al., 2004; Svetlitsky et al., 2018). In
computational genomics, operon structure is inferred from the conserved physical proximity of genes in
phylogenetically distant genomes. Second, to annotate uncharacterized genes, whose functions could be
inferred from other genes belonging to the same block (Kim et al., 2005). This concept has been used in
numerous cases in which gaps in metabolic pathways could be filled by hypothetical genes encoded in the
operon for that pathway (e.g., Osbourn, 2010). Third, in microbial genomics, comparisons of gene organ-
izations and gene block discovery have been used to study the importance of spatial organization in
genome function, including functional selection, operon formation, and horizontal transfer. In short,
conservation of spatial organization of genes and the computational identification of gene blocks provides
an important source of information that is orthogonal to primary sequences of genes and thus could be
exploited to supplement existing genomic analysis tools.

The gene block discovery problem (also denoted as "gene cluster discovery") has been well-formalized
and extensively studied on the theoretical front, and several models were proposed for the problem
(reviewed in Section 2). Current tools for general gene block discovery are computationally expensive and
do not scale up, in terms of CPU time and memory requirements, to the fast-growing databases of
prokaryotic genomes, which already contain tens of thousands of samples (Svetlitsky et al., 2018).
However, biologists are not always interested in the full output computed by exhaustive discovery of all
gene blocks in a given data, but rather in discovering gene blocks participating in a specific pathway under
study, or gene blocks that are supersets of a specific subset of genes of interest.

This inspires a new problem variant, denoted "Constrained Gene Block Discovery," which we formalize
and solve in this article. In our constrained variant of the gene block discovery problem, the sought gene
blocks must comply by a user-specified list of constraints. The constraints describe either a set of genes, or
a list of functional categories (keywords), that are required to appear in the semantic annotation of the
sought genes. In both cases the genes in a sought gene block are a superset of the genes described by the
query.

This variant of the gene block discovery problem has several applicative motivations: First, biologists
interested in particular groups of genes or functional modules could find other related features, by in-
specting the genomic context or neighborhood of the genes of their interest. Such experts might be able to
interpret these neighborhoods and find examples of nonorthologous gene displacements, or horizontal gene
transfers, that might have an effect on the functional module in particular organisms.

Second, biologists often ask general questions regarding specific functional context conservation (e.g.,
ATP-binding cassette (ABC)-transporters and their regulons), without wanting to go into the details of
which specific gene ortholog groups are associated with each function of interest. As far as we know, none
of the existing gene-block discovery tools implements a functional semantics based search for gene blocks
based on a prespecified set of functional-annotation keywords.

The third motivation involves scalability, as mentioned above. Our proposed approach could be har-
nessed to make the data-mining of the rapidly growing databases feasible: if we know in advance that the
biologist making the query is interested in a specific set of genes or functional categories, the discovery of
gene blocks abiding by the query could be sped up by focusing the search according to the specified constraint. Of course, constrained gene block (CGB) discovery could naively be solved through the following two-stage approach: first interrogate the input data with a general gene block discovery tool (e.g., Jahn, 2011), to identify all gene blocks complying with the specifications that are typical of the general gene block discovery problem (i.e., a requirement on the minimal number of genomes in which an instance of the gene block is to be found, and a parameter bounding the length of a genomic window in which the genes of an instance are to be colocalized). Then, in a postprocessing stage, filter the results identified in the first stage by enforcing the constraints that are specific to the constrained gene block discovery problem variant (i.e., identify gene blocks resulting from the first stage that also comply with the prespecified functional annotation requirements). However, such an approach does not make sense in terms of scalability, and thus, we propose hereby to use the constraints to speed up the search.

1.1. Our contribution and roadmap

In this article, we propose a new approach to comprehensively identify and analyze constrained gene blocks in large datasets of prokaryotic genomes. Our new approach helps to reveal conserved gene neighborhood context that is associated with the query genes. The discovered context, in turn, helps to decipher the function of uncharacterized gene clusters and to discover new functional associations between operons, as exemplified in Section 6.

To address the new problem, we formalize the Constrained Gene Block discovery problem as a clear-cut optimization problem (Section 3) and propose a solution to the problem based on harnessing a well-known data mining approach, previously developed for market-basket analysis (Pei and Han, 2002), to solve it (Section 4). The proposed solution is exact, that is, it finds and reports all Constrained Gene Blocks complying with the user-defined parameters. In Section 6, we exemplify the application of the tool we developed for this problem with two different case studies related to microbial ABC transporters. Related work is reviewed in Section 2, and a discussion is given in Section 7.

1.2. Workflow overview

The workflow of the proposed approach is given in Figure 1. The input to the workflow is a dataset of input genomes $D = \{S_1 \ldots S_m\}$, where each genome is a sequence of gene orthology group identifiers (Fig. 1A). Additional input consists of user-specified parameter $\ell$, bounding from below the number of genes in a gene block; $q$ a quorum parameter, bounding from below the number of genomes in which a gene block is required to have an instance; and a window size $d$ bounding from above the distance between any two genes in any instance of the gene block. The input also includes a user-defined set of specifications $C = \{c_1 \ldots c_k\}$, allowing repetitions of the same specification within the set. The specifications could be of two types: (1) gene families, such as the Clusters of Orthologous Genes (COGs) annotation, that should be included in the sought gene-block (e.g., “COG1,” “COG2,” “COG3”) and/or (2) functional categories that are required to be “covered” by genes in a sought gene-block, where a gene could belong to multiple

![FIG. 1. The pipeline for the algorithm. Input parameters in this example are: $d = 3$, $q = 3$, $l = 2$, $C = \{\text{green, blue}\}$. (A) The input genomes, where each row represents one genome and each arrow-like shape represents a gene. Genes that belong to the same orthology group have the same color. (B) Step 1 of the algorithm (Interval Filtration): rectangles outlined in black mark surviving intervals. (C) The application of Step 2 of the algorithm (Interval Mining) to the sparse search space composed of the intervals surviving the first step (Interval Filtration) of the algorithm. The only frequent genes (according to $q = 3$) are the green, blue, and yellow ones, and therefore, only these colors are represented in the search diagram. (D) The result of the search used in Step 2 of the algorithm, containing two gene blocks. The instances of the two gene blocks are not shown in the figure; however, the first block consisting of three genes has three instances and the second block has four instances in the data illustrated in (A, B).]
functional categories (e.g., “kinase,” “regulator”). Our objective is to identify blocks of genes, denoted “instances,” that are co-localized within a window of length $d$ in $q$ or more genomes from $S$, such that all instances contain a subset of genes that abide by the user-defined specification.

This objective is achieved through a two-stage approach. First, the genomes are mined to identify all $d$-length windows that abide by the given specifications (Fig. 1B). Next, the windows surviving the first stage are data mined (using a search approach described later in the article) to identify all constrained gene blocks. The discovered blocks are ranked according to a probabilistic score and subjected to biological interpretation (Fig. 1C, 1D).

2. RELATED WORK

The approach we present in this article enables the discovery and inspection of genomic context or neighborhoods of specific query genes across a set of input genomes, where the query genes could be defined precisely by specific COGs or more generally by keywords describing a function of interest. So far, several tools that allow inspection of specific genes of interest and their genetic context were developed. A STRING web server (Snel et al., 2000; Szklarczyk et al., 2016) takes as input a set of query genes defined by sequences, gene names, or COG categories and displays their occurrences across genomes on top of a phylogenetic tree, where neighborhood relationships between the query genes are incorporated into the view. Other two web-based tools, GeConT 2 (Martinez-Guerrero et al., 2008) and SyntTax (Oberto, 2013) take as input one or more query genes and then display them and their genomic neighbors. In all these tools, orthologous genes that belong to the same functional category (e.g., COG) are marked with the same color in the graphical output. GeConT 2 (Martinez-Guerrero et al., 2008) allows flexibility in defining the query genes, using identifiers from various databases, for example: COG (Tatusov et al., 2003), KEGG (Kanehisa, 2002), Pfam (Finn et al., 2015), Rfam (Daub et al., 2015), and keywords describing the gene, similar to our approach. SyntTax (Oberto, 2013) takes as input only a single protein sequence, and its main advantage is in providing a taxonomic tree of all prokaryotes in NCBI repository and allowing restricting the analysis to specific nodes in the tree.

As opposed to our tool, none of the above tools guarantees reporting all exact Gene Blocks that comply with the query constraints. Furthermore, the gene blocks identified by our tool contain also neighboring genes that were not part of the query that abide by the quorum constraints.

On the theoretical front, the problem of discovering general (unconstrained) gene blocks in a set of input genomes has been thoroughly studied, and various models were proposed for this problem (reviewed in Bergeron et al., 2008). These models vary by the types of restrictions they impose on the problem: whether the genomes are confined to permutations (Uno and Yagiura, 2000; Heber and Stoye, 2001; Bergeron et al., 2002; Schmidt and Stoye, 2004) or alternatively duplicate copies of (paralogous) genes in a genome are allowed (Amir et al., 2003; He and Goldwasser, 2005; Amir et al., 2007; Benshahar et al., 2017), and whether they require that all input genomes have instances of the gene block (all works before 2006) or alternatively allow for instances to be found in only a subset of the input genomes (Kim et al., 2006; Ling et al., 2009 and all works afterward).

Approximate variants of the problem were also proposed, where various types of errors are allowed between the gene set pattern representing the gene block and its instances: the types of allowed errors vary, starting from models that allow only a bounded number of insertions between two genes (Bergeron et al., 2002; He and Goldwasser, 2005) or a bounded window length in which an instance is to be contained (Benshahar et al., 2017) and up to more flexible models that also allow deletions or substitutions (Rahmann and Klau, 2006; Böcker et al., 2009).

Another constrained problem variant is one where the pattern gene set representing a gene block is required to have a reference occurrence in one of the genomes, that is, an occurrence without intermitting or missing genes (Jahn, 2011). GECKO 3 (Winter et al., 2016), which is an implementation of this reference-based algorithm, is currently the only active tool for the general gene block discovery problem.

At the basis of the constrained gene block discovery problem addressed in this article stands a definition of an (unconstrained) gene block, which was previously proposed in Benshahar et al. (2017). This basic gene block is characterized by a set of genes (the “pattern” of the gene block) that are required to occur, clustered together within a window of bounded length, in a predefined quorum of the genomes. Within this problem definition, the model we follow in our solution to the problem is the most general one: It allows paralogous occurrences of genes in the input genomes, it does not require gene order conservation across distinct instances, and it does not require that instances from all input genomes participate in a gene block.
CONSTRAINED GENE BLOCK DISCOVERY

result. The approach we describe in this study does not improve the worst case time complexity of the algorithm proposed in Benshahar et al. (2017); however, we propose an approach that solves the new (constrained) problem variant defined in this article, while at the same time utilizing the constraints to speed up the search in practice using techniques previously applied to related problems in the domain of Frequent Itemset Mining (FIM) (Pei and Han, 2002). These include a branch-and-bound approach that utilizes the user-defined constraints to prune the search space and a specialized data structure for efficient counting of instance support.

3. PRELIMINARIES AND DEFINITIONS

The most basic unit we work with is the gene orthology group. In this article a gene orthology group is a cluster of orthologous genes as defined in the COG database (Tatusov et al., 2000). Thus, in the rest of this article, we use the terms “gene X,” “COG X” interchangeably to denote a member of a gene orthology group X.

Let $\Sigma$ denote a finite set of characters representing genes. A genome is represented by a sequence $S = \sigma_1 \cdots \sigma_n$ of concatenated characters from $\Sigma$. For a sequence $S$, we use $|S|$ to denote the length of $S$ and $S[i]$ to denote the $i$’th character of $S$. A subsequence of a sequence $S$ is any nonempty sequence $S_0$ that can be obtained by deleting zero or more characters from $S$. An interval of $S$ is a subsequence of $S$ with consecutive characters. For $1 \leq i \leq j \leq |S|$, we let $S[i, j] = \sigma_i \cdots \sigma_j$ denote the interval of $S$ beginning at position $i$ and ending at position $j$. Two sequences $S_1$ and $S_2$ are said to be equivalent, denoted $S_1 \equiv S_2$, if $|S_1[i] = \sigma : 1 \leq i \leq |S_1| |S_2[j] = \sigma : 1 \leq i \leq |S_2| |\forall \sigma \in \Sigma$. In other words, $S_1 \equiv S_2$ if both sequences have the same number of occurrences of each character $\sigma \in \Sigma$. Clearly, for two equivalent sequences $S_1$ and $S_2$ we have $|S_1| = |S_2|$.

3.1. Definitions for gene block discovery

Let $D = \{S_1, \ldots, S_m\}$ denote a set of sequences representing the input genomes. Let $d$ denote an integer specifying a bound on the length of the genomic substring to which a gene block is to be confined. Let $q$ denote the minimal number of genomes required to contain an occurrence of the gene block. An instance of the (general) gene block discovery problem is defined by a triple $(D, d, q)$. The goal of the gene block discovery problem is to find all sets of genes that have equivalent occurrences in areas of interest of the input genomes. We formalize this in the following definition.

3.1.1. Definition 1 (gene block). A gene block $(P, \{S_1, \ldots, S_{supportP}\})$ in $(D, d, q)$ is a set of genes $P$ and a set of genomes $\{S_1, \ldots, S_{supportP}\}$ satisfying the following:

(1) $P$ is a permutation set of genes from $\Sigma$ (i.e., a set of distinct letters).
(2) For each $1 \leq i \leq supportP$, $S_i \in D$.
(3) For each $1 \leq i \leq supportP \supseteq T_i$ such that $T_i$ is a subsequence of some $d$-length interval of genome $S_i \in D$ and $P \equiv T_i$.
(4) $S_1 \neq S_2, \ldots, \neq S_{supportP}$.
(5) $supportP \geq q$.

3.1.2. Definition 2 (closed gene block). We say that a gene block $(P, \{S_1, \ldots, S_k\})$ is closed in $(D, d, q)$ if there is no other block $(P', \{S'_{1'}, \ldots, S'_{r'}\})$ in $(D, d, q)$ where $P$ is a subset of $P'$ and $\{S_1, \ldots, S_k\} \subseteq \{S'_{1'}, \ldots, S'_{r'}\}$.

3.1.3. Definition 3 (Closed Gene Block discovery problem [CGB]). The Closed Gene Block Discovery problem is the problem of computing all closed gene blocks in a given problem specification $(D, d, q)$ and reporting the blocks, as well as their corresponding instances.

3.2. Definitions for constrained gene block discovery

In our application, each gene $\sigma_i \in \Sigma$ is annotated with one or several specifications (“colors” in Fig. 1), denoting its functional annotations or other properties of interest, that is, gene $\sigma_i \in \Sigma$ is colored $c_k$ if it is annotated with biological feature $c_k$. 

Now that the general Gene Block Discovery problem has been formalized, we are ready to define a constrained variant of it. Let $C = \{c_1, \ldots, c_L\}$ be a set of user-defined specifications, such that $c_i$, for $i = 1, \ldots, L$, denotes a functional annotation category of a gene orthology group. In our model, repetitions of the same specification within the set $C$ are allowed, and for each $c_i \in C$, a gene annotated with functional annotation $c_i$ is required to appear in the sought constrained gene block.

Example: in Figure 1, $C = \{\text{green}, \text{blue}\}$, that is, we only seek gene blocks that contain at least one green specification and at least one blue specification. Since the gene blocks are constrained by the $C$ requirement, along with the requirement imposed by the interval-length bound parameter $d = 3$, the relevant intervals to consider in the data are the ones highlighted by black rectangles demonstrated in Figure 1B. Figure 1D illustrates that indeed all solutions contain both a green and a blue gene.

An additional constraint to consider is $\ell$, which bounds from below the number of genes in a gene block.

### 3.2.1. Definition 4 (constrained closed gene block)
A constrained closed gene block $(P, \{g_1, \ldots, g_{\text{support}}\})$ in the problem specification $(D, d, \ell, C, q)$ is a set of genes $P$ and a set of genomes $\{S_1, \ldots, S_{\text{support}}\}$ satisfying the following:

1. $(P, \{S_1, \ldots, S_{\text{support}}\})$ is a closed gene block in the problem specification $(D, d, q)$ according to Definition 2.
2. $\ell \leq |P|$.
3. For each specification $c_i \in C$, $P$ contains at least one distinct gene with functional annotation $c_i$.

In the constrained variant of the closed gene block discovery problem, which we define below, the goal is to find all constrained closed gene blocks in the problem specification $(D, d, q)$, that is, all closed gene blocks from the problem specification $(D, d, q)$, that also abide by the constraint specifications $C$ and $\ell$.

### 3.2.2. Definition 5 (Constrained Closed Gene Block discovery problem [CCGB])
The Constrained Closed Gene Block discovery problem is the problem of computing all constrained closed gene blocks in a given problem specification $(D, d, \ell, C, q)$ and reporting the blocks, as well as their corresponding instances.

### 3.3 Monotonicity and anti-monotonicity of candidate gene blocks
The constraints and parameters mentioned above in the definitions of the gene block discovery problems can be classified into several categories, such as monotonic, antimonotonic, and convertible.

#### 3.3.1. Definition 6 (antimonotone, monotone)
A constraint $c$ is antimonotone if and only if whenever a gene set $S$ violates $c$, so does any superset of $S$. A constraint $c'$ is monotone if and only if whenever a gene set $S$ satisfies $c'$, so does any superset of $S$ (Pei et al., 2002).

Claim 1: The CCGB constraints obey the following properties:

1. Constraints $q$ and $d$ are antimonotone with respect to increasing candidate gene set size.
2. Constraints $C$ and $\ell$ are monotone with respect to increasing candidate gene set size.

### 4. SOLVING CCGB
In this section we propose a general solution for solving CCGB that exploits the constraints and properties given in Section 3 for efficient search.

#### 4.1. High level overview
The general strategy of our algorithm is based on two consecutive steps (illustrated in Fig. 1): In the first step, denoted “Interval Filtration,” the goal is to compute a subset of all $d$-length intervals from the input genomes that abide by the $d$ and $C$ specifications. In the second step, denoted “Interval Mining,” the goal is to search for constrained closed gene blocks that abide by the given $d$, $q$, $\ell$, and $C$ specifications within the set of candidate intervals surviving the first step of the algorithm.
4.1.1. The interval filtration step. In this step, the query is first processed to identify the least frequent functional category. Using a table calculated in a preprocessing stage that contains for each COG the number of its appearances in the data, along with a classification table retrieved from NCBI containing each COG with its annotation, we can deduce the frequency of every functional category in C. The lowest frequent category $c_1 \in C$ is marked as the “least frequent category.” Genes annotated by $c_1$ will serve as seeds in the search for intervals complying by $C$. This step first locates all occurrences of the seeds in the data and extends them $d$ genes left and right, creating a $2d$-1 genomic interval. If this “double length” interval complies with the full constraint $C$, each of its $d$-length intervals is (incrementally) examined for compliance with $C$. Each $d$-length interval complying with $C$, annotated with its corresponding genome of origin, is passed as input to the second, Interval Mining, step of the algorithm.

4.1.2. The interval mining step. We observe that the task addressed in this step resembles a related method that is well established in the data mining community, denoted “Frequent Itemset Mining” or FIM (Borgelt, 2012). FIM is a classical data mining method that was originally developed for market-basket analysis. It aims at finding regularities in the shopping behavior of customers of supermarkets, mail-order companies, and online shops. In particular, it deals with the task of identifying sets of products that are frequently bought together in the same transaction. The data in the FIM problem are composed of records, each containing a transaction, that is, a set of items bought together. In our case, the data that are input to the Interval Mining step consist of $d$-length intervals that survived the Interval Filtration step. In both problems, the objective is to data-mine the enumeration space of subsets of items (or genes, respectively) to identify frequent subsets.

In the domain of electronic commerce, users are often interested in a specific subset of frequent itemsets, yielding a need for constrained variants of FIM. That is, users can compose constraints about items appearing in the patterns, the length of the patterns, model-based constraints, aggregates related to the patterns, and even grammatical syntax constraints on the sought patterns, such as requiring that they abide by regular expressions. While such constraints can be applied as a postprocessing step, integrating them into the mining algorithm can dramatically reduce the execution time. A few works address constrained variants of FIM (Srikant et al., 1997; Boulicaut and Jeudy, 2001; Pei and Han, 2002; Buzmakov et al., 2017). Thus, it makes sense to borrow some of the computational approaches developed for Constrained FIM and adapt them to our case. In the next section, we propose to adapt one of the existing FIM approaches to the implementation of the Interval Mining step of our solution.

We note in advance that our problem is somewhat different than the constrained FIM problem in two aspects. The first one is that the input data we receive include gene duplications (corresponding to paralogous genes), as opposed to the classical FIM problem which assumes permutation of transactions. This can immediately be solved by renaming distinct copies of the same gene that occur within the same interval with distinct letters and incrementing the size of the gene alphabet by a factor of $d$. Thus, in the following sections we assume that the above renaming strategy is applied and reformalize a gene block pattern definition as a permutation over the new extended alphabet.

Another difference relates to the possibility of having multiple identical instances of a gene block on the same genome. Since the support of a gene block is defined by the number of genomes it is contained in (rather than by the number of intervals in which it is contained—as would be the case if the problem was equivalent to standard FIM), some technical book-keeping measures need to be taken, to correctly count candidate support. We do not include these details in this article due to space constraints.

4.2. A constrained FIM approach to CCGB interval mining

Previously related Constrained FIM approaches succeed to effectively use the constraints to speed up the search. This is achieved by utilizing the properties of the constraints (antimonotonicity, monotonicity, etc.) to integrate them as early as possible into the search, to effectively prune the exponential enumeration space of $|\Sigma|^d$ gene sets. We note, however, that the CCGB problem defined in this study combines both antimonotonic and monotonic constraints, and this combination raises some challenges in efficient integration of these constraints within the search.

4.2.1. Antimonotone constraints $q$ and $d$. Based on the antimonotone constraint $q$ that bounds from below the number of genomes in which a gene block is required to have an instance, the search space for gene blocks is naturally structured according to the subset relationships between gene sets, which form a
partial order on $|\Sigma|^d$. Therefore, it is natural to use a branch-and-bound gene block mining algorithm to search in a data structure that represents the partial order $\subseteq$ on $|\Sigma|^d$, thus enumerating the gene blocks in increasing size. In the worst case the running time of such a search is at least linear in the number of $O(|\Sigma|^d)$ possible solutions. However, the fact that instance support count is antimonotone with respect to increasing gene set size allows for the admissible pruning (i.e., pruning that does not interfere with the correctness of the results) of the search space by the $q$ constraint.

Another consequence of the antimonotonicity property obeyed by constraint $q$ is that it allows to compute a compressed format of the output, without loss of information, by restricting the reported blocks to closed gene blocks (defined in Section 2). Due to the antimonotonicity property of instance support count, the set of all frequent gene blocks in a given problem specification can easily be recovered from the set of closed gene blocks for the same problem specification, since any gene block is either closed or possesses a uniquely determined closed superset.

4.2.2. Monotone constraints $C$ and $l$. A more challenging objective is to integrate the monotone properties ($C$ and $l$) into the search pruning, because these properties go “against the grain” of incremental set enumeration. In this study, we observe that our problem has high resemblance to a previously studied variant of the Constrained FIM problem, denoted “Model-based Constrained FIM” (Pei and Han, 2002). In this problem variant of FIM, the constraints denote a prespecified set of items (genes in our case) that should be included as a subset of any frequent itemset result (Pei and Han, 2002). This problem variant was addressed by two well-studied FIM approaches: Apriori (Srikant et al., 1997) and Frequent-Pattern Growth (FP-Growth) (Pei and Han, 2002). In the rest of this section, we shortly review the two approaches and explain why we chose the latter as the basis of our solution for the Interval Mining Step.

In the Apriori algorithm, based on the antimonotonicity property of the frequent itemset constraint that is equivalent to our $q$. The essential idea behind the Apriori algorithm is to iteratively generate the set of frequent patterns of length $(k+1)$ from the set of frequent patterns of length $k$ (for $k \geq 1$), and to check their corresponding occurrence frequencies in the database. Therefore, an intuitive methodology to integrate constraints into Apriori-like approaches is to use antimonotonic constraints to prune candidates. However, Apriori-like approaches are quite challenging in terms of incorporating monotonic constraints to speed up the search. This seems to indicate that Apriori-like approaches would not be a good fit to problems such as CCGB, which is characterized by a composition of monotonic and antimonotonic constraints.

Example: To exemplify this point, consider the gene set $ab$ in our running example (Fig. 2). This gene set violates the constraint imposed by parameter $l$, which is set to three in this example. However, an Apriori-like algorithm cannot prune such a gene set. Otherwise, its super-pattern $abcd$, which satisfies the constraint, will not be generated. As observed in this example, when applying monotonic constraints to prune an enumeration by increasing set size, supersets of a valid gene set could well be valid and thus inadmissibly pruned out. Thus, within a level-wise framework, no direct pruning based on such a constraint can be made without risking the correctness of the output. Although we exemplified this point through constraint $l$, the same problem arises when attempting to apply the monotone constraint $C$ to the pruning of a level-wise approach.

Another approach to mine frequent itemsets is the Frequent Pattern Growth approach (Pei and Han, 2002), or FP-Growth for short. This approach is based on imposing a lexicographic order $R$ on the items. The order enables to encode the input data set in a prefix tree data structure, denoted FP-tree (Tan et al., 2005), thus compactly combining transactions sharing the same prefix. Using the FP-tree, the FP-Growth method uses a divide and conquer strategy to obtain efficient instance support counting of decreasing projected databases (Pei and Han, 2002). By growing frequent patterns by an appropriate order $R$, the FP-Growth approach can effectively mine itemsets constrained by a composite of antimonotonic and monotonic specifications and thus to avoid, or dramatically reduce, candidate generation and testing. The FP-Growth method will be explained and illustrated in the next section.

Considering the points listed above, we note that, for our specific gene block discovery problem variant, the FP-Growth approach is a good fit, for several reasons. The first reason is that our problem definition indeed yields a composite of antimonotonic and monotonic constraints. The second reason relates to the expected density of the input data set: The pruning used in the Interval Filtration stage is expected to leave just those interval candidates that comply by $C$. Consequently, these intervals are likely to intensively share common genes with each other. This observation is relevant when $C$ contains functional annotations and even stronger when $C$ is composed of specific COGs. Therefore, the search space to be mined during the Interval Mining stage, based on the intervals surviving the Interval Filtration stage, is expected to be dense.
and thus to benefit from a prefix-tree representation of the search space: An FP-tree data structure is suitable for dense datasets since many intervals will share common prefixes so that the database could be compactly represented. Furthermore, the divide and conquer strategy of the FP-Growth creates subproblems of decreasing projected-database size, reducing dramatically the execution time of the instance support counting operations.

4.3. Converting the CCGB constraints to a sorted problem specification

In this section we show how to convert the CCGB monotonic constraints C and ℓ to corresponding antimonotonic constraints, by first sorting the input, as well as the enumerated patterns by some consistent order R, and then applying sequential data mining to the sorted problem specification. This is achieved by proposing some reformulation of the CCGB monotonic constraints, followed by an explanation of how they

![FIG. 2. Effective pruning of prefix-driven FP-Growth under constraints: q = 2, ℓ = 3, d = 4, C = {blue, red, red}. This example is explained in the body of the text. Intervals shown in (A) come from four different genomes. (B) The “Hasse Diagram” representing the naive search space for the intervals from (A). (C) The subset of edges from (B) traversed when applying a constrained pattern growth approach. (D) The projected database of every pattern induced sub problem.]
are implemented within the pattern-growth framework. We also show how to convert the antimonotonic constraint \( q \) to an equivalent constraint adapted to the sorted problem specification.

Assume that any enumerated pattern gene set, as well as the intervals to be considered as instances in the Interval Mining step, is lexicographically sorted into sequences by some prespecified order \( R \). Also assume that the search space of candidate gene sets is enumerated sequentially according to \( R \), in increasing prefix length. Note that, if a gene set \( S' \) is a subset of another gene set \( S \), the sorted \( S' \) is a subsequence of \( S \) in the ordered sequential variant of the problem, where both \( S \) and \( S' \) are sorted by the same order \( R \). Thus, the original CCGB constraints can be converted to the following constraints, without affecting the search results (Pei and Han, 2002).

**4.3.1. Definition 7 (convertible constraints).** Given an order \( R \) over the set of genes from \( \Sigma \), a gene set \( S' = i_1, i_2, \ldots, i_m \) is called a prefix of gene set \( S = i_1, i_2, \ldots, i_w \) w.r.t \( R \), when items in both gene sets are listed according to order \( R \) and \( m \leq w \). \( S' \) is called a proper prefix of \( S \) if \( m < w \).

A constraint \( C \), is convertible antimonotone provided there is an order \( R \) on genes such that whenever a gene set \( S \) satisfies \( C \), so does any prefix of \( S \). A constraint \( C \), is convertible monotone provided there is an order \( R \) on genes such that whenever a gene set \( S' \) violates \( C \), so does any gene set \( S \) such that \( S \) is a prefix of \( S' \). A constraint is convertible whenever it is convertible antimonotone or convertible monotone.

Note that any monotone (respectively, antimonotone) constraint is trivially convertible monotone (respectively, convertible antimonotone): just pick any order on items.

Now that the search space of candidate gene sets is enumerated sequentially according to \( R \), in increasing prefix length, the objective is to “prune” a prefix (i.e., a path from the root of the enumeration tree to some node in the tree) if extending it by additional characters cannot lead to a pattern complied by the prespecified constraints. Thus, we distinguish between “nonpromising prefixes,” that is, prefixes that surely cannot be extended to results complying with the given constraints, versus “promising prefixes,” that is, prefixes which cannot be safely pruned.

**4.3.2. Definition 8 (”promising prefix”).** Let \((R, I, d, \ell, C, q)\) denote an ordered problem specification for the Interval Mining step, where \( R \) denotes an arbitrary order on \( \Sigma \) and \( I \) denotes the intervals from \( D \) that survived the Interval Filtration step. A “promising prefix” \((S, \{g_1, \ldots, g_{\text{support}_S}\})\) in \((R, I, d, \ell, C, q)\) is a sequence of genes \( S \) and a set of genomes \( \{g_1, \ldots, g_{\text{support}_S}\} \) satisfying the following:

1. \( S \) is a permutation of genes from \( \Sigma \), ordered by \( R \).
2. For each \( 1 \leq i \leq \text{support}_S \), \( g_i \in D \).
3. \( g_1 \neq g_2, \ldots, \neq g_{\text{support}_S} \).
4. For any genome \( g_i \), such that \( g_i \) is counted in \( \text{support}_S \) : \( \exists t_i \) such that \( t_i \) is a subsequence of some interval \( I_j \in I \) and \( S=t_i \).
5. \( |S| \geq q \).
6. \( |S| \leq d \).
7. The following assumption cannot be dis-validated based solely on \( S \) : \( \exists S' \) such that \( S \) is a prefix of \( S' \), \( S' \) is a permutation of genes from \( \Sigma \) ordered by \( R \), and \( S' \) abides by the following constraints:
   7.1. \( |S'| \geq \ell \)
   7.2. \( S' \) abides by \( C \)

An “admissible pruning heuristic” is one that does not degrade the correctness of the results, that is, using this heuristic will not cause the search to miss out on any of the results claimed by the problem definition nor to report additional results that are not claimed by the problem definition. In the claim below, we use the term “exact search” to denote a search that uses only admissible heuristics, and thus, the results reported by this search fully comply with the results claimed by the formal problem definition.

Claim 2: An exact CCGB search based on a “prefix extension and prune” strategy can be proposed based on the constraints of Definition 8.

Proof: Constraints 1–5 from Definition 8 are a trivial sequential conversion of the antimonotone constraint \( q \) of the original CCGB problem by any arbitrary order \( R \) on the genes from \( \Sigma \). A pattern-growth method takes advantage of the converted antimonotone property of \( q \) and grows longer patterns from shorter ones that satisfy these constraints. Any prefix that does not satisfy Constraints 1–5 is pruned immediately from the search. Constraint 6 can be enforced by bounding the depth of the search tree by \( d \).
CONSTRANDED GENE BLOCK DISCOVERY

In Contrast to constraints 1–6, Constraint 7.1 and 7.2 are conversions of the monotonic constraints \( \ell \) and \( C \) of the original CCGB problem. These two monotonic constraints trivially yield equivalent convertible monotonic constraints by converting the problem specification to a sequential one ordered by \( R \) and using enumeration by prefix extension. However, the obtained convertible monotonic constraints cannot be directly applied to the sequential search space variant without risking admissibility, since an enumerated prefix \( S \) does not hold enough information to rule out all its putative super patterns \( S' \) based on these two constraints. However, a pattern-growth method can take advantage of constraints 7.1 and 7.2 to grow longer prefixes from shorter ones that satisfy these constraints and to admissibly prune any prefix that is not promising according to constraints that are partial conversions of the original ones:

4.3.3. Constraint 7.1: sequential conversion of the monotone constraint defined by parameter \( \ell \). Let \( A = a_1 a_2 \ldots a_f \) be the sequence obtained by ordering, according to \( R \), all single genes surviving the constraint \( q \) (first level of the DAG shown in Fig. 2C). Let \( S = s_1 s_2 \ldots s_g \) be a candidate prefix considered for pruning. Clearly, for some \( 1 \leq t \leq f \), \( s_g = a_t \). If \( f - t < \ell - g \), then the sequentially ordered problem specification \( (R, I, d, \ell, C, q) \) does not contain a sequence \( S' \) such that \( S \) is a prefix of \( S' \) and \( S' \geq \ell \). Thus, prefix \( S \) can be admissibly pruned.

Example: The pattern \( S = "d" \) in Figure 2C cannot be developed to a pattern of size of at least \( \ell = 3 \), as we have \( f - t = 5 - 4 < 3 - 1 = \ell - g \).

4.3.4. Constraint 7.2.: sequential conversion of the monotone constraint defined by parameter \( C \). Let \( C = \{c_1, \ldots, c_L\} \) be the set of functional requirements and let \( S = s_1 s_2 \ldots s_g \) be a candidate prefix considered for pruning. Let \( \text{color}(s_g) \) denote the color (i.e., functional category) of gene \( s_g \), and let \( B_g = b_1 b_2 \ldots b_f \) be the sequence obtained by ordering, according to \( R \), big space all single genes from sequence \( A \) (A is defined in constraint 7.1 above) that are annotated by \( \text{color}(c_k) \). Let \( t \) denote the index of \( s_g \) in \( B_g \), and let \( \text{fulfilled}(S, s_g) \) denote the number of genes annotated by \( \text{color}(s_g) \) in \( S \). Let \( q' = |\{c_j \in C : c_j \neq \text{color}(s_g)\}|. \) If \( f - t + \text{fulfilled}(S, \text{red}) < q' \) then the sequentially-ordered problem specification \( (R, I, d, \ell, C, q) \) does not contain a sequence \( S' \) such that \( S \) is a prefix of \( S' \) and \( S' \) abides by \( C \). Thus, prefix \( S \) can be admissibly pruned. Note that once the examined prefix extends beyond the length in which all constraints from \( C \) have been satisfied, that is, \( > L \), constraint 7.2 no longer needs to be considered in the pruning process.

Example: The pattern \( S = "ad" \) in Figure 2C cannot be developed to a pattern that fulfills the constraints requiring two red genes, as we have \( f - t + \text{fulfilled}(S, \text{red}) = 2 - 2 + 1 < q' = 2 \).

4.4. The search: a frequent pattern growth approach to interval mining

In this section we describe a pattern-growth based approach for implementing the Interval Mining step of CCGB. Given the problem specification \( (R, I, d, \ell, C, q) \), where \( R \) denotes an arbitrary order on \( \Sigma \) and \( I \) denotes the intervals from \( D \) that survived the Intervals Filtration stage (Step 1). The algorithm first sorts the genes in the intervals \( I \) by their order according to \( R \). Then, the search space of candidate gene sets is enumerated sequentially according to \( R \), in increasing prefix length, and pruned so that only “promising prefixes,” according to Definition 8, are extended to longer candidates. The algorithm grows longer patterns from shorter ones by dynamically traversing the search tree in Depth First Search (DFS) order, while in parallel constructing a projected database that will serve for efficient instance support count for the considered candidates. This is done recursively: for each pattern surviving level \( k \), a dedicated projected database is formed, which contains only intervals having a suffix starting with that pattern. In parallel, “Promising prefixes” are identified in the prefix tree as those abiding by the constraints described in Definition 8 and Claim 2 from Section 4.3.

We refer the reader to Figure 2 for a step-by-step illustration of an application of FP-Growth approach to the Interval Mining step of CCGB. In this example, consider the task of applying the Interval Mining step of CCGB to the problem specification \( (R, I, \ell = 3, d = 4, q = 2, C = \{\text{blue, red, red}\}) \). First, the genes within each interval of \( I \) are sorted by the order \( R \) (see Fig. 2, in this example the order is lexicographic). Next, a branch-and-bound search algorithm is used over a growing enumeration space of sequences ordered by \( R \) from \( |\Sigma|^d \).

Figure 2B illustrates a Directed Acyclic Graph, denoted “Hasse Diagram” (Borgelt, 2012), representing the naive search space for this problem specification, enumerating all possible \( R \)-ordered permutations over
The alphabet was first reduced to include only frequent items from $\Sigma$ of length up to $d=4$ from $I$. (Note that the alphabet was first reduced to include only frequent items from $I$ that abide by the instance-support quorum $q$). A naive algorithm would traverse all $O(|\Sigma|^d)$ dotted edges in this Hasse Diagram. However, the search is pruned by the converted constraints $q$, $\ell$, and $C$ mentioned above.

Figure 2C illustrates the subset of edges traversed when applying a constrained pattern growth approach to this problem specification. The approach is denoted “Pattern Growth,” since longer patterns are grown above. This Hasse Diagram. However, the search is pruned by the converted constraints, induced by the requirement for closed itemsets, deep into the mining process. We refer the reader to Wang et al. (2003) for details regarding additional admissible heuristics applied to mining closed patterns with Pattern Growth.

Among the frequent genes from the example, only genes $a$, $b$, and $c$ satisfy constraint 7.1, and only $a$ and $b$ satisfy both constraints 7.1 and constraint 7.2. Thus, $c$, $d$, or $e$ are not “promising prefixes” according to Definition 8 and will not be further extended. The set of candidates satisfying the constraint, to be further extended, can be partitioned into two subsets: (1) Those having COG $a$ as prefix. (2) Those having COG $b$ as prefix. The frequent candidates having $a$ as a proper prefix can be found in the subset of intervals containing $a$, which is called an “$a$-projected database.” Since $a$ appears in every interval in the $a$-projected database, it is now omitted from the intervals shown in the projected database and moved to the header of the rectangle. The $a$-projected database contains four candidates: $bcd$, $cde$ (from Genome 2), $cde$ (from Genome 3), and $bcd$. The frequent COGs in the $a$-projected database are: $b$, $c$, $d$, and $e$, listed in lexicographic order. Since $ae$ and $ad$ do not satisfy constraints 7.1 (nor 7.2), there is no need to create either $ae$-projected or $ad$-projected databases. Thus, in the next level, the $a$-projected database is split into $ab$-projected and $ac$-projected databases and so on.

To summarize the results illustrated in this example, the complete set of gene blocks satisfying the CCGB constraints contains four gene blocks: $acd$, $abcd$, and $acde$. Constraints were computed for only 22 gene set candidates (number of edges in Fig. 2C), among which only 11 gene set candidates required the costly database scanning operation for instance support counting (number of green edges in Fig. 2C). Thus, the application of a pattern-growth approach to the Interval Mining step of CCGB in this example generated and tested only a small subset of patterns.

Note that since the CCGB problem addressed in this study is that of mining closed patterns, the FP-Growth based CCGB algorithm can be further sped up by applying some techniques to push additional constraints, induced by the requirement for closed itemsets, deep into the mining process. We refer the reader to Wang et al. (2003) for details regarding additional admissible heuristics applied to mining closed patterns with Pattern Growth.
However, we note that when implementing the FP-growth algorithm, the support count operation is not expected to take $O(nm)$ time as, for a given subproblem, only the projected database, rather than the complete database, needs to be scanned. In addition, in the FP-Growth algorithm, the projected database is stored in an FP-tree, which is expected to be dense in our problem variant, as it utilizes common prefixes. Therefore, it is expected that for the typical queries $T_{\text{check, db}}$ is much smaller than $O(nm)$ (Borgelt, 2012). The search space described above is bound by $|\Sigma|^d$; however, when assuming no constraints besides interval length and instance support quorum (parameters $d$ and $q$, respectively), there are at most $O(m^{2d})$ possible gene blocks [i.e., all possible subsets of genes from $O(nm)$ $d$-length intervals]. For each such candidate gene block, at most $|\Sigma|$ support count operations are executed by the search: once per each trial of extension of this gene block by a distinct gene. This results in $O(mn^{2d+|\Sigma|})$ enumerated sets, each in $O(T_{\text{check, db}})$ time, yielding an overall time complexity of $O(T_{\text{check, db}} mn^{2d+|\Sigma|})$. In total, we have $O\left(T_{\text{check, db}} \min\left\{|\Sigma|^d, mn^{2d+|\Sigma|}\right\}\right)$. We note that, in the theoretical case where memory resources pose no bottleneck to scalability, an $O(mn^{2d} T_{\text{check, db}})$ time algorithm can be proposed, eliminating a factor of $O(|\Sigma|)$ by constructing an enumeration tree consisting of only gene sets that appear at least once in any one of the intervals in $I$.

As for the space complexity, a given node will need to save the projected database and the status of the remaining yet unfulfilled constraints in $C$, thus requiring $O(D + |C|)$ space per node. As the search tree is scanned by DFS, it only needs to keep track of the information in the recursion of the current node and the ones of its ancestors. The longest branch in the search tree is of length $d$, so in total the space requirement is $O(d(D + |C|))$.

5. METHODS

5.1. Ranking score

An important issue is how to rank the discovered CCGBs. In this study, we used the same ranking system described in (Benshahar et al., 2017) for computing a ranking score for a gene block.

5.2. Dataset

In our study, we used the dataset from Svetlitsky et al. (2018), which contains COG ID (COG) (Tatusov et al., 2000) annotations of 1487 fully sequenced prokaryotic strains. This dataset contains multiple strains for many of the species. To generate a dataset where every species is represented by a single strain, we kept, per each species, the strain that has the highest proportion of genes with COG annotations. This resulted in a reduced dataset containing 991 genomes.

For the reduced dataset, the alphabet $\Sigma$ consists of 4830 COG IDs. Each gene (coding sequence) is labeled by at most one COG, and unmapped genes are marked by an additional character “--”. The COG database is manually curated, where each COG ID has a functional annotation and is associated to a functional category. The COG semantics can be utilized for functional interpretation of the results. This dataset is used in Section 6 to obtain rich biological results amenable to taxonomic distribution analysis.

5.3. Postprocessing of the resulting gene blocks

To enhance the functional analysis of the obtained gene blocks, we performed a postprocessing analysis. Let $(P, \{S_1, \ldots, S_{\text{support}}\})$ be a gene block that was obtained based on specification $(D, d, \ell, C = c_1, \ldots, c_q)$. We parse it as follows: Any gene $\sigma_\gamma \in P$ that was used to fulfill one of the constraints $c_1, \ldots, c_q$ in $C$ is denoted a “query fulfilling COG,” while any other gene $\sigma_\gamma \in P$ that was not used to fulfill any of the constraints is denoted a “hitchhiker COG.” Note that for any functional constraint $c_j \in C$, there could be more than one gene fulfilling $c_j$, and also vice versa there could be a single gene fulfilling more than one constraint, resulting in various gene blocks that differ not only by their hitchhiker genes but also by the genes fulfilling the query. As will be demonstrated in the next section, the hitchhikers bear important contextual information that will help decipher the query.

5.3.1. Clustering gene blocks with overlapping query fulfilling COGs. We apply a greedy incremental clustering procedure, similar to CD-HIT (Li, 2015), that clusters together gene blocks that have a significant overlap between their query COGs. The resulting clusters are denoted “bins.” Briefly, gene
blocks are first sorted in decreasing order of the number of query fulfilling COGs. The first gene block, with the largest number of query COGs, forms the first bin and becomes its representative. Then, the query COGs of each remaining gene block are compared to the query fulfilling COGs of the existing bin representatives. If more than 50% of the COGs match, it is grouped into that bin. Otherwise, a new bin is defined with that gene block as the representative.

This first level clustering divides the gene blocks into bins, such that all the gene blocks in the same bin represent generally the same system, with slight variation in the query fulfilling COGs.

We then run second level clustering, by applying the same clustering procedure on every bin that was obtained by the first level of clustering, but now requiring the proportion of matching genes to be 100%. This procedure results in clusters denoted “sub-bins,” where all the gene blocks in the same sub-bin share the exact same query fulfilling COGs; however, they differ in their hitchhiker COGs. Inspection of sub-bins allows identification of different gene neighborhoods of the same set of queries fulfilling COGs.

Since our goal in this study is to inspect the genomic context of specific query fulfilling genes, we apply an additional filtration procedure to the computed sub-bins, which removes sub-bins that have no gene blocks with hitchhiker COGs.

5.3.2. Calculating enrichment in specific taxonomic groups. We retrieved taxonomic classification for every genome in our dataset from NCBI (2018). Then, for every gene block \( P, \{S_1, \ldots, S_{support}\} \) and for every taxonomy class \( t \), we calculated the statistical significance of the overlap between \( t \) and the genomes represented by \( \{S_1, \ldots, S_{support}\} \) using a hypergeometric \( p \)-value score. This process yields a vector \( (p_1, \ldots, p_t) \) for every gene block where \( p_i \) is the enrichment score for the taxonomy class \( t_i \). Every such vector was then filtered so that only \( p \)-values smaller than 0.05 remained.

6. RESULTS

To test the functionality of our tool, we focused on two different case studies related to the ABC transporters (Table 1). The ABC transporters form one of the largest known protein families and are widespread in bacteria, archaea, and eukaryotes (Davidson et al., 2008). The transporters couple ATP hydrolysis to active translocation of a wide variety of substrates ranging from ions, sugars, amino acids, vitamins, lipids, antibiotics, and drugs to larger molecules such as oligosaccharides, oligopeptides, and even high molecular weight proteins (Higgins, 1992). In microorganisms, ABC transporters are central to antibiotic and antifungal resistance, and in human they are associated with several genetic diseases, including cystic fibrosis, Tangier disease, obstetric cholestasis, and drug resistance of cancers (Gottesman and Ampudkar, 2001). The structure of a prokaryotic ABC transporter usually consists of three components; typically, one or two integral membrane proteins (permeases), one or two peripheral proteins that bind and hydrolyze ATP (ATPases), and a periplasmic substrate-binding protein. Many of the genes for the three components form operons as in fact observed in many bacterial and archaeal genomes (Kanehisa, 2002).

6.1. Query 1: uncovering the function of an uncharacterized transporter by context

Our first query was designed to test the ability of our tool in uncovering the function of an uncharacterized ABC type transport system, by analyzing hitchhiker COGs that are linked to this system. To do so, our query contained the following keywords [periplasmic, permease, atpase, POORLY

<table>
<thead>
<tr>
<th>Case study</th>
<th>Keywords (C)</th>
<th>u</th>
<th>l</th>
<th>q</th>
<th>No. of found gene blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Uncovering the function of an uncharacterized transporter by context</td>
<td>periplasmic, permease, atpase, POORLY CHARACTERIZED (*3)</td>
<td>10</td>
<td>3</td>
<td>30</td>
<td>21</td>
</tr>
<tr>
<td>(2) Studying the context of ABC transporters in defense mechanisms</td>
<td>periplasmic, permease, atpase, Defense (*3)</td>
<td>10</td>
<td>3</td>
<td>10</td>
<td>272</td>
</tr>
</tbody>
</table>

ABC, ATP-binding cassette.
Table 2. Top-Scoring 11 Gene Blocks Found by Query 1, All Located in the Highest Scoring Bin

<table>
<thead>
<tr>
<th>Gene block No.</th>
<th>Score</th>
<th>Hitchhiker COGs</th>
<th>No. of genomes</th>
<th>No. of instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2279</td>
<td>[COG0213, COG0295]</td>
<td>379</td>
<td>521</td>
</tr>
<tr>
<td>2</td>
<td>983</td>
<td>[COG0295]</td>
<td>91</td>
<td>92</td>
</tr>
<tr>
<td>3</td>
<td>874</td>
<td>[COG0213]</td>
<td>108</td>
<td>109</td>
</tr>
<tr>
<td>4</td>
<td>746</td>
<td>[COG0213]</td>
<td>93</td>
<td>94</td>
</tr>
<tr>
<td>6</td>
<td>504</td>
<td>[COG0213, COG0274, COG0295]</td>
<td>38</td>
<td>39</td>
</tr>
<tr>
<td>7</td>
<td>427</td>
<td>[COG0213, COG0274]</td>
<td>41</td>
<td>42</td>
</tr>
<tr>
<td>8</td>
<td>405</td>
<td>[COG0274, COG0295]</td>
<td>39</td>
<td>39</td>
</tr>
<tr>
<td>10</td>
<td>378</td>
<td>[COG0274]</td>
<td>49</td>
<td>50</td>
</tr>
<tr>
<td>12</td>
<td>337</td>
<td>[COG1816]</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>13</td>
<td>296</td>
<td>[COG0583]</td>
<td>39</td>
<td>47</td>
</tr>
<tr>
<td>14</td>
<td>288</td>
<td>[COG2188]</td>
<td>38</td>
<td>38</td>
</tr>
</tbody>
</table>

The gene blocks share the same query fulfilling COGs (COG1744, COG1079, COG4603, COG3845) that describe an uncharacterized ABC-type transport system and come associated with 6 different hitchhiker COGs across 10 of the gene blocks. The original gene block number, the score, the hitchhiker COGs, the number of genomes, and the number of instances are indicated for each gene block. COG descriptions are found in Table 3.

COGs, Clusters of Orthologous Genes.

Characterized (*3) and was run with parameters as indicated in Table 1. This run resulted in 21 gene blocks. After binning and filtration (see Section 5), two bins were left, representing two different uncharacterized systems. We focus on the first bin, which contained the top scoring gene block. Altogether, this bin contained 11 gene blocks, clustered into a single sub-bin. The gene blocks in this sub-bin share four query fulfilling COGs and vary by their hitchhiker combinations, which are basically obtained as distinct subsets from a pool of six hitchhikers (Table 2). Two of the hitchhikers are transcriptional regulators (transcription factors, TFs). One of the TFs, GntR (COG2188), seems to be linked to the transport system, since in most instances it appears directly upstream or downstream of its genes, and in the same strand, while the second TF, LysR (COG0583), does not show any consistent proximity or strand orientation with the transport system. Examination of the TF COG annotations does not provide additional information that could suggest that this uncharacterized transport system transports a specific substrate (Table 2).

Table 3. The Description for Each of the Query Fulfilling Clusters of Orthologous Genes and Each of the Hitchhiker Clusters of Orthologous Genes from Table 2 is Given

Query fulfilling COGs

| COG1744 | M; Cellular processes and signaling; Cell wall/membrane/envelope biogenesis; Basic membrane lipoprotein Med, periplasmic binding protein (PBP1-ABC) superfamily; |
| COG1079 | R; Poorly characterized; General function prediction only; ABC-type uncharacterized transport system, permease component; |
| COG4603 | R; Poorly characterized; General function prediction only; ABC-type uncharacterized transport system, permease component; |
| COG3845 | R; Poorly characterized; General function prediction only; ABC-type uncharacterized transport system, ATPase component; |

Hitchhiker COGs

| COG0213 | F; Metabolism; Nucleotide transport and metabolism; Thymidine phosphorylase; |
| COG0295 | F; Metabolism; Nucleotide transport and metabolism; Cytidine deaminase; |
| COG0274 | F; Metabolism; Nucleotide transport and metabolism; Deoxyribose-phosphate aldolase; |
| COG1816 | F; Metabolism; Nucleotide transport and metabolism; Adenosine deaminase; |
| COG0583 | K; Information storage and processing; Transcription; DNA-binding transcriptional regulator, LysR family; |
| COG2188 | K; Information storage and processing; Transcription; DNA-binding transcriptional regulator, GntR family; |
Interestingly, the other four hitchhikers are Deoxyribose-phosphate aldolase (COG0274), Adenosine deaminase (COG1816), Thymidine phosphorylase (COG0213), and Cytidine deaminase (COG0295), all classified under the “Nucleotide transport and metabolism” category (Table 3) and describe enzymes that participate in nucleotide catabolic processes. Cross-referencing the 379 genomes from gene block 1 with gene blocks 2, 3, 4, 6, 7, 8, 10, and 12 (Table 2) revealed that in 130 genomes out of 379, the transporter is physically associated with one or more of the 4 hitchhikers (other combinations of these hitchhikers are also possible; however, they did not pass the quorum constraint).

Examination of the instances of gene block 6 (data not shown), that contained three of these hitchhikers (Fig. 3A and Table 2), revealed that all the hitchhikers are found in the same strand directly upstream or downstream to the uncharacterized ABC-type transport system. Furthermore, when we plotted the phylogenetic placement of the found instances for this gene block on top of the phylogenetic tree, we observed that they are distributed across different taxonomic groups (Fig. 3B). Based on the widely conserved context of the transport system we can propose with high confidence that the uncharacterized ABC-type transport system is transporting nucleotides.

While the COG terms in this example are all uncharacterized, we wondered about the annotation of the genes that fall under these COG terms. To this end we extracted from our dataset the genes from the found

![FIG. 3. A graphical view of gene block (number 6) that was found by the first query (see keywords in Table 1). COG numbers are written under each colored arrow. The block consists of four query fulfilling COGs and three hitchhiker COGs. The white arrows indicate genes that were part of the window (d=10) but do not belong to the block. COGs surrounded by purple rectangle represent the queried transport system, while COGs surrounded by orange rectangle represent the hitchhikers. Description of the COGs is supplied; terms that matched the keywords are shown in bold. (B) Phylogenetic distribution of instances in gene block from (A). COG, Clusters of Orthologous Genes.](http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3949563/figures/fig3.jpg)
instances that are labeled with the transport system COGs: COG1744, COG1079, COG4603, and COG3845. Surprisingly, we found variability in gene annotations that point to unresolved function of these genes. For example, for COG1079 that represents one of the permease components in the system, we saw very general annotations for genes that appear under this category, for example, “ABC transporter permease,” “inner membrane translocator,” “transmembrane component of ABC transporter,” to more specific annotation, yet contradicting, for example, “amino acid ABC transporter permease,” “carbohydrate ABC transporter permease,” “nucleoside ABC transporter membrane protein,” and “monosaccharide ABC transporter permease.” In a more extreme case, we found an instance in the genome of *Melissococcus plutonius* ATCC 35311, where the components of the system were annotated with contradicting functional assignments (Table 4).

We then searched the literature for biological evidence to support the function proposed by the hitchhikers that were found by our tool. We found that the ABC-type system that is composed of the COGs: COG1744, COG1079, COG4603, and COG3845, was studied in *Lactococcus lactis* (Martinussen et al., 2010). The transport system in this species is composed of the three-gene operon *nupABC* that encodes the ATPase and two permeases. The fourth substrate-binding periplasmic component is encoded by the gene *bmpA* that is located 60 kb upstream of the *nupABC* operon and is required for a functional uptake of all common nucleosides (nucleotides without a phosphate group). Interestingly, three of the hitchhikers that we identified are located upstream of *bmpA* and are encoded by the genes *pdp*, *deoC*, and *cdd* (Martinussen et al., 2010). Altogether, the experimental data in this study suggested that BmpA–NupABC system is indeed an ABC transporter capable of transporting all common nucleosides.

### 6.2. Query 2: the genomic context of ABC transporters in defense mechanisms

To explore the genomic context of ABC transporters that are involved in defense mechanisms, we ran the second query (Table 1), requiring the keywords [periplasmic, permease, atpase, Defense (*3*)] to appear in the found gene blocks. The search yielded 272 gene blocks that were binned into 2 bins, with 31 and 1 sub-bins, respectively. The two bins represent two different types of transport systems, where the first one is enriched in *Gammaproteobacteria*, while the second one is enriched in *Alphaproteobacteria*. In our analysis we focused on the first system represented by gene blocks from Bin 1, since these contained the top scoring gene blocks. Examination of one of these gene blocks revealed that six COGs were query fulfilling genes, and two additional COGs were hitchhikers captured within the required gene block window. Further zooming into the description of the COGs uncovered that the found gene block consists of two different systems (Fig. 4). The first one is an ABC-type antimicrobial peptide (AMP) transport system answering our basic query, containing one periplasmic, two permeases, and two ATPase components. This ABC type system is known also as *SapABCDF*, which stands for “sensitive to antimicrobial peptides” (Cupp et al., 2002).

AMPs are small peptides, which permeabilize bacterial membranes, and thus serve as a key component of the innate immune defense systems against pathogenic bacteria, in many organisms throughout the animal and plant kingdoms (Brogden, 2005). For successful survival in the host, bacteria have a series of AMP resistance mechanisms (Joo et al., 2016). One such mechanism is achieved by the *sapABCDF* operon (Parra-Lopez et al., 1993; Mason et al., 2005). In this antimicrobial resistance mechanism, the *sapA* component of the transport system binds AMPs. This binding leads to increased expression of the other components in the ABC transporter, which transfer the AMPs into the cytosol for intracellular proteolytic degradation (Shelton et al., 2011).
Interestingly, adjacent to the ABC-type transport system, we found the phage shock protein (Psp) stress response system that includes the operon PspABC and the transcription factor PspF, which positively regulates the operon (Darwin, 2013). The Psp response has been most studied in Escherichia coli, where it has been shown to involve the products of a larger operon, PspABCDE, that is invoked under a wide variety of membrane stresses. The name for the proteins came from the discovery that the filamentous phage f1 greatly induced the expression of the operon by expression of protein IV (pIV). The pIV protein is a member of the secretin family that forms an outer membrane pore through which the phage is extruded (Model et al., 1997). In many other enterobacteria, the pspF and pspABC genes are highly conserved (Huvet et al., 2009); however, they lack the pspD and pspE genes.

In addition to phage infection, the Psp stress response is induced also by other environmental stresses that can negatively affect the cell envelope, including exposure to organic solvents, extreme heat or osmotic shock, and high ambient pH (Joly et al., 2010). Furthermore, the Psp response has also been shown to be required for the formation of biofilms and antibiotic-resistant persister cells in E. coli (Vega et al., 2012). These observations led to the hypothesis that the Psp system helps cells to maintain the integrity of the bacterial cell envelope in unfavorable conditions.

Examination of the instances, across 19 species (data not shown), that contain the gene block from Figure 4, revealed that PspABC is found on one strand, while the pspF gene, which is a transcription factor, is found on the other strand immediately upstream of the ABC type transport system. To assess the frequency of the two systems in our dataset, we first counted all genomes that have at least one instance of the transport system sapABCDF (by inspecting all gene blocks that were reported in this query). We then ran a complementary query, to locate the Psp operon using the parameters: $d = 10$, $q = 10$, $l = 3$,
C = \{COG1221, COG1842, COG1983\}, and counted the genomes that have at least one instance of this operon. Interestingly, enrichment analysis on taxonomic groups (see Section 5) revealed that both operons are found only in Gammaproteobacteria (which consists of 154 distinct species in our dataset). Our analysis revealed that the transport system is found in 84 genomes, and the \textit{Psp} operon is found in 60 genomes, with an overlap of 58 genomes between the 2 systems. In 50 out of these 58 genomes, the 2 operons are found together in proximity within the window that was specified in the search. The physical linkage of the two operons could suggest that they acquired a mutual regulatory mechanism, or that one of the systems is regulating the response of the other, possibly through the function of the \textit{PspF} transcription factor. Further experimental perturbations of the locus are required to explore this hypothesis.

7. DISCUSSION

In this article, we formulate a new variant of the well-known gene cluster discovery problem, where the sought gene blocks are a superset of a set of genes required to comply by user-specified constraints. The newly formulated problem inspires the utilization of techniques from the FIM domain, to obtain an efficient solution that scales up to the rapidly growing genomic databases, and in particular to those of microbial genomes. To the best of our knowledge, such approaches have not been previously considered by related works on gene block discovery.

An interesting open problem for future work would be to extend the constraints defined in this article to include some additional, more expressive types of constraints. For example, we can extend the “AND” formulas specified by our parameter \( C \) to a general 3CNF formula (which can be obtained from every Boolean formula) of biological functional specifications. Other interesting constraints were already incorporated in the related applications of Constrained FIM (Pei and Han, 2002) and it would be interesting to consider them in the context of gene blocks.

Annotation of genes with their correct function is challenging and time consuming if performed using wet bench experiments. The rapidly growing accumulation of completely sequenced genomes from different environments provides a great opportunity to assist in this task by comparative genomics, for example, by identifying conserved gene blocks and by inspecting their conserved neighborhoods, which may have been already annotated. In the current study, we applied our developed approach to uncover the function of a previously uncharacterized ABC-type transport system. The unknown function is reflected not only in the COG descriptions but also in the variable annotations of the genes that were assigned under these COG terms.

Our search revealed strong genomic context of the transport system with genes that are involved in nucleotide transport and metabolism, suggesting that the transport system is transporting nucleotides. For experimental validation of our predicted functional characterization of this gene block, mutagenesis and gene deletion experiments could be conducted to assess the relevance and contribution of the gene block to the suggested pathway (Martinussen et al., 2010). Indeed, an independent search of the literature for the COGs that appeared in the found gene blocks supported the suggested function in one bacterial genome (Martinussen et al., 2010). Thus, the approach suggested in this work could be incorporated into annotation pipelines that are run routinely and, based on already accumulated data, could assist in precise annotations of newly sequenced genomes.

In addition to uncovering gene function, genomic context may help to predict association among known individual genes or even among complete operons. The second query that we tested helped us to uncover previously unreported association between two operons. The first is \textit{SapABCDF}, which represents AMP transport system. The second is the \textit{Psp} operon that has an important role in stress response and also in helping cells to maintain the integrity of the bacterial cell envelope in unfavorable conditions. The function of the two operons seems to be related, and the gene blocks that were found pointed also to their conserved physical linkage. The physical proximity of these two operons could suggest that they acquired a mutual regulatory mechanism or that one of the systems is regulating the response of the other. This type of prediction could be further explored in an experimental setting by co-expression analysis, for example, Monje-Casas et al. (2001). Thus, our approach can help uncover new association and possibly new regulatory relationships between distinct bacterial systems.
ACKNOWLEDGMENTS

The authors thank David Bouhadana for his contribution during the early stages of this research. The authors also thank Dina Svetlitsky for sharing her biological knowledge and data. The research of J.E. and M.Z.-U. was partially funded by the Israel Science Foundation (Grant Nos. 179/14 and 939/18).

AUTHOR DISCLOSURE STATEMENT

The authors declare there are no competing financial interests.

REFERENCES

CON鄄STRAINED GENE BLOCK DISCOVERY


Address correspondence to:
Dr. Isana Veksler-Lublinsky
Department of Software and Information Systems Engineering
Ben Gurion University of the Negev
P.O.B. 653 Beer-Sheva
8410501 Israel
E-mail: vaksler@post.bgu.ac.il

Dr. Michal Ziv-Ukelson
Department of Computer Science
Ben Gurion University of the Negev
P.O.B. 653 Beer-Sheva
8410501 Israel
E-mail: michaluz@cs.bgu.ac.il