Accelerating Gene Context Analysis Using Bitmaps

Alex Romosan
Lawrence Berkeley National Laboratory
1 Cyclotron Rd
Berkeley, CA 94720
ARomosan@lbl.gov

Arie Shoshani
Lawrence Berkeley National Laboratory
1 Cyclotron Rd
Berkeley, CA 94720
Shoshani@lbl.gov

Kesheng Wu
Lawrence Berkeley National Laboratory
1 Cyclotron Rd
Berkeley, CA 94720
KWu@lbl.gov

Victor Markowitz
Lawrence Berkeley National Laboratory
1 Cyclotron Rd
Berkeley, CA 94720
VMMarkowitz@lbl.gov

Kostas Mavrommatis
Lawrence Berkeley National Laboratory
1 Cyclotron Rd
Berkeley, CA 94720
KMavrommatis@lbl.gov

ABSTRACT

Gene context analysis determines the function of genes by examining the conservation of chromosomal gene clusters and co-occurrence functional profiles across genomes. This is based on the observation that functionally related genes are often collocated on chromosomes as part of so called "gene cassettes", and relies on the identification of such cassettes across a statistically significant and phylogenetically diverse collection of genomes. Gene context analysis is an important part of a genomic data management system such as the Integrated Microbial Genomes (IMG) system, which has one of the largest public genome collections. As of January 2013, IMG contains 3.3 million gene cassettes across 8,000 genomes. A gene context analysis in IMG performs many millions of comparisons among the cassettes and their functions. Using a traditional relational database management system, these cassettes and their functional characteristics are represented by a correlation table of more than 2 billion rows along with a dozen auxiliary tables. This correlation table requires 16.5 hours to build and a typical query requires 5 to 10 minutes to answer.

We developed an alternative approach that encodes the cassettes and their functions using bitmaps. Reading the input data now takes about 1.5 hours and constructing the bitmap representations takes only 8 minutes. This amounts to less than one tenth of the time needed to build the correlation table. Furthermore, fairly complex queries can now be answered in seconds. In this work, we considered three basic forms of queries required to support gene context analysis and devised two different bitmap representations to answer such queries. These queries can be answered in less than a second. A more complex query, which we referred to as a "killer query", requires the examination of multi-way cross-products of all cassettes. We developed a progressive pruning strategy that effectively reduces the number of possible combinations examined. Tests have shown that we can now answer "killer queries" in seconds. Even with an extremely complex "killer query" involving 161 genomes (needing a 161-way cross-product), our algorithm took less 10 seconds. A query involving this many genomes is expected to take so much time using a traditional DBMS that it has never been attempted before.

Working with the IMG developers, we have verified our implementation and have integrated it into the production version of IMG.

Categories and Subject Descriptors

H.2.4 [**Query processing**]: *set-valued query processing, cross-product query processing, biological queries*

1. INTRODUCTION

Computational methods for determining the function of genes in newly sequenced genomes have been traditionally based on sequence similarity to genes whose function has been identified experimentally. Additional predictions can be made through gene context analysis which examines the conservation of chromosomal gene cassettes across genomes. Based on the observation that functionally related genes often have similar gene context, gene context analysis can reliably predict a gene's function by identifying similar gene contexts across a statistically significant and phylogenetically diverse collection of genomes. Tools to predict a gene's function are a critical part of genomic data analysis systems such as the Integrated Microbial Genomes (IMG) [5].

Developed at Lawrence Berkeley National Laboratory, IMG is extensively used world-wide for genome analysis. IMG's analysis toolkit includes search and visualization tools to conduct gene context analysis across a large set of publicly available archaeal and bacterial genomes. Gene context analysis starts with first identifying "gene cassettes," which are defined as a sequence of co-located genes, *i.e.* genes separated by less than 300 nucleotides, as shown in Figure 1. A detailed explanation of how cassettes are constructed is given in Section 2. Thus, each genome has multiple gene cassettes associated with it. The average number of cassettes per genome is about 400 (see Section 2.3 for a sample distribution). Since each gene is associated with several functional characterizations (functions for short) each cassette is associated with a set of functions. The functional characterization of a gene involves comparing the gene structure (sequence) to functional resources such as Pfam

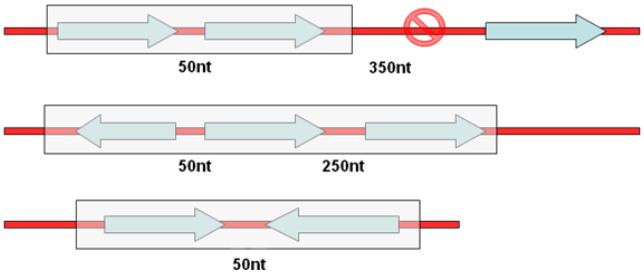


Figure 1: Illustration of gene cassettes. A gene cassette is defined as a sequence genes separated by less than 300 nucleotides. Examining the conservation of chromosomal gene cassettes across genomes facilitates gene context analysis.

[2] (Protein families) and COG [8] (Clusters of Orthologous Groups of proteins). A typical function set for a specific cassette may be: "cog0087 cog0088 cog0089 cog0090 cog0091 cog0092 cog0185 cog0197 pfam00181 pfam00189 pfam00203 pfam00237 pfam00252 pfam00276 pfam00297 pfam00573 pfam00831 pfam03947 pfam07650".

Currently, the total number of functions is about 20,000. Thus, in order to store these values in a conventional relational database, one has to either have 20,000 columns, store the functions as a list of values, or as a correlation table of (cassette, function). Because the number of values associated with each cassette is usually a small subset of the 20,000 possible values, it makes sense to store the values as lists, or as a correlation table. However, these structures are not very helpful when running complex queries such as finding all cassettes that have two-or-more functions in common, since this requires multiple runs over the data with all possible combinations. To provide real time response, all possible matches are searched ahead of time, and stored in the database as views. There are a number of difficulties in using these views. The first challenge is that it often takes many days to generate all views. Additionally, the disk space required to store all the views is significantly larger than the original data. As the number of genomes grows, both the time needed to generate the views and the space required to stored the views grow very quickly. Furthermore, as the volume of the data increases, even searching over views became too slow for on-line interaction. We chose a radical approach that avoids these difficulties, by representing the data as bitmaps, and using our compressed bitmap technology, called FastBit (see Section 3) for such searches.

The key contribution of this work include:

- Propose two bitmap representations for the cassettes and their functions,
- Propose algorithms to perform common operations of gene context analysis using the new data structures,
- Demonstrate the effectiveness of the new algorithms and data structure using the massive amount of data available through IMG,
- Develop a robust implementation to be distributed with the production IMG system.

In the next section, we present a brief explanation of gene context analysis which is based on genome annotation, and an overview of the IMG system and how it addresses the problem of gene context analysis. This is followed by an overview of FastBit indexing. In Section 4 we describe three types of queries typical to gene context analysis, but cannot be easily processed by conventional relational systems. This is followed by showing how bitmap technology can be applied for each type of query to achieve near real-time processing that can support on-line interaction without inflating the data size with views. For each of the query types we show how FastBit indexing accelerated the processing further, and provide real measurements of data sizes and retrieval times.

2. GENE CONTEXT ANALYSIS

2.1 Genome data annotation

Genome data captures information about raw DNA sequence data, along with genes characterized in terms of functions and pathways. A gene represents an ordered sequence of nucleotides located on a particular chromosome that encodes a specific product (i.e., a protein or RNA molecule). Characterizing a gene consists of determining its biological context, including its location on a chromosome within a (species specific) genome, and its associated functional roles in cellular pathways. A key characteristic for genome is its taxonomic (phylogenetic) lineage, including its domain, phylum, class, order, family, genus, species and strain. Pathways can be viewed as ordered lists of reactions, whereby each reaction involves compounds which are reactants (substrates, products), catalyzed by enzymes. Pathways can be combined in pathway networks, whereby pathways can be associated via reactions that share common components. Pathways are associated with genes via gene products that function as enzymes that serve as catalysts for individual reactions of metabolic pathways. Accordingly, pathways provide a biologically meaningful framework for examining functional relationships between genes, rather than individual gene functions.

Genome annotation refers to the process of assigning biological meaning to the raw sequence data by identifying gene regions and determining their biological functions. Gene annotation is a combination of automated methods that generate a "preliminary" annotation in terms of predicted genes (also called Open Reading Frames or ORFs, which represent the sequence of DNA or RNA located between the start-codon and stop-codon sequence) and associated functions and pathways based on sequence similarity or profile searches. Start-codon and stop-codon sequences are specific known sequences in the beginning and end of a gene sequence. They determine the direction of a gene as shown in Figure 1.

Computational methods for determining the biological function of genes have been traditionally based on sequence similarity to genes whose function has been identified experimentally. Function prediction methods can be extended using gene context analysis approaches such as examining the conservation of chromosomal gene cassettes across genomes. Context analysis is based on the observation that genes with related (coupled) functions are often both present or both absent within genomes and tend to be collocated (on chromosomes) in multiple genomes [1]. Context analysis then

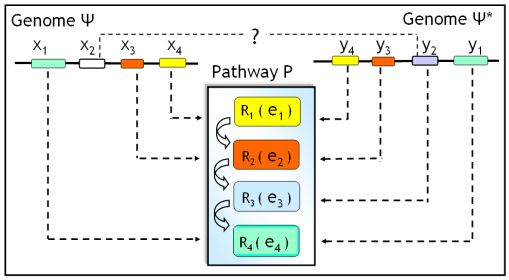


Figure 2: A representation of two genomes and their functional annotation specified in terms of the association of their genes (x, y) with enzymes (e) that serve as catalysts for reactions (R) in a pathway (P).

relies on the identification of such events across a statistically significant and phylogenetically diverse collection of genomes.

Consider the example shown in Figure 2, where pathway P involves reactions R_1, R_2, R_3, and R_4: genes x_1, x_3, and x_4 of genome Φ are associated with pathway P via enzymes e_1, e_3, and e_4, respectively; genes y_1, y_2, y_3, and y_4 of genome Φ^* are associated with pathway P via enzymes e_1, e_2, e_3, and e_4, respectively. Following rules above, x_2 may be associated with P via enzyme e_2.

2.2 IMG

The Integrated Microbial Genomes (IMG) system developed at Lawrence Berkeley National Lab serves as a community resource for comparative analysis of publicly available genomes in a comprehensive integrated context [5]. IMG integrates publicly available draft and complete genomes from all domains of life with a large number of plasmids and viruses.

Data warehouse constructs were employed for specifying IMG's data model in terms of primary (fact) objects characterized in the context of other (dimension) objects. Each dimension is further characterized by one or several category attributes which are sometimes organized in a classification hierarchy. Genes are the primary objects in IMG and are characterized in the context of individual genomes, functions, such as protein family and domain characterizations based on COG clusters and functional categories [8], protein families, Pfam [2], and pathways, such as metabolic pathways. IMG's data warehouse is implemented using Oracle 13 Database Management System.

IMG contains information on functional units such as *chromosomal cassettes*. A chromosomal cassette is defined as a stretch of genes (regardless of their orientation) with intergenic distance smaller or equal to 300 base pairs, whereby the genes can be on the same or different strands of the chromosome, as shown in Figure 1. The choice of 300 base pairs or less was determined experimentally as having high correlation between cassette functions. The identification of common gene cassettes across organisms is based on participation in COG or Pfam clusters and provides insights in their function [7]. Groups of at least two common genes between two or more chromosomal cassettes are defined as *conserved chromosomal cassettes*. In order to identify common genes between chromosomal cassettes, genes need to be assigned to groups of equivalent genes. For this grouping, the commonly accepted clusters of orthologous genes (COG) and Pfam assignments are used. If a gene is associated with multiple clusters, such as multiple Pfam domains, each individual domain is included in the chromosomal cassette.

Genome data analysis in IMG consists of operations involving genomes, genes, and functions which can be selected, explored individually, and compared. The composition of analysis operations is facilitated by genome, scaffold, gene and function "carts" that handle lists of genomes, scaffolds, genes and functions, respectively.

Several tools support gene context analysis in IMG [6]. Thus, gene cassettes can be searched using "Cassette Search" and "Phylogenetic Profiler for Gene Cassettes". "Cassette Search" allows users to find genes that are part of chromosomal cassettes involving specific protein clusters. First, users select the protein cluster underlying the cassettes, the protein cluster identifier for the search, the logical operator used for the search expression and the order of presenting the search results, as illustrated in Figure 3(i). The search is carried out across all the genomes in IMG (default) or can be limited only to a subset of genomes using various filters. Next, the "Cassette Search Result" lists the genes that satisfy the search condition, together with the identifiers of the cassettes they are part of, their associated protein cluster identifiers and names, and their genomes, as illustrated in Figure 3(ii). The cassette identifiers provide links to the "Chromosomal Cassette" details page, as illustrated in Figure 3(iii).

The "Phylogenetic Profiler for Gene Cassettes" allows users to find genes that are part of a gene cassette in a genome of interest (so called query genome) and are part of related gene cassettes in other genomes: users select the query genome by using the associated radio button in the "Find Genes In" column, the protein cluster used for correlating gene cassettes, and the genomes for gene cassette comparisons with the query genome by using the associated radio buttons in the "Collocated In", as illustrated in Figure 3(iv). Next, the "Phylogenetic Profiler for Gene Cassette Results" provides a summary of the results, including a table with the first column listing the size of the groups of collocated genes in the query genome and the second column listing the number of such groups conserved across the other genomes involved in the selection. The details part of these results consists of a table that displays groups of collocated genes in each chromosomal cassette in the query genome that satisfy the search criterion, as illustrated in Figure 3(v). The conserved part of a chromosomal cassette involving an individual gene in the query genome can be examined using the links provided in the "Conserved Neighborhood Viewer Centered on this Gene" column of results table, as illustrated in Figure 3(vi).

2.3 Data size and analysis time

The content of IMG has grown steadily since the first version released in March 2005, and includes (as of May 1st 2013) 5,804 bacterial, archaeal, and eukaryotic genomes, as well as 2,809 viral genomes bringing its total genome content to 8,613 genomes with about 23 million genes. The number

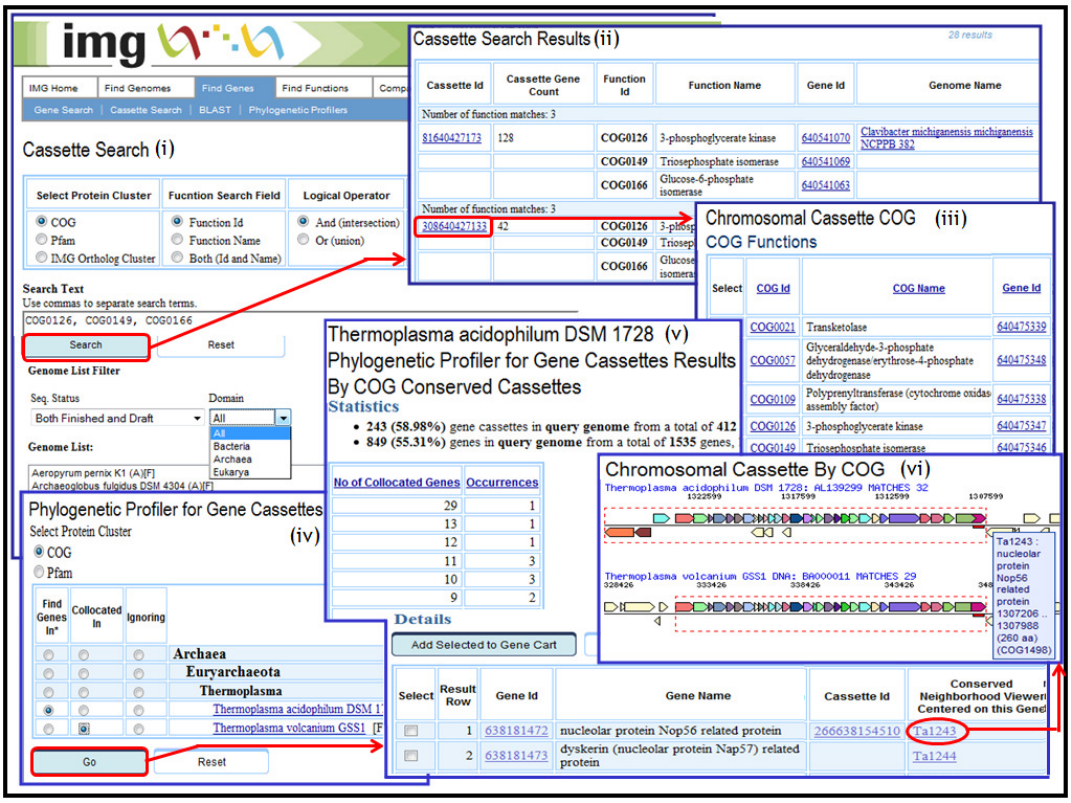


Figure 3: Gene Cassette Search Tools in IMG.

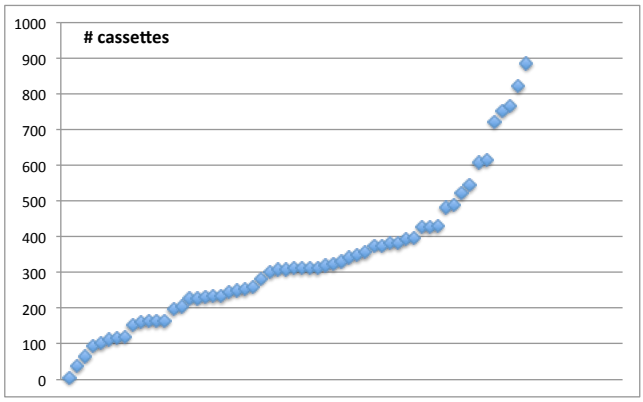


Figure 4: The number of cassettes in a random sample of genomes. For ease of display, the genomes are ordered according to the number of cassettes.

of cassettes varies from genome to genome, ranging from a few to about a thousand, a sample of about 60 of them are shown in Figure 4.

There are about 3.3 million gene cassettes in IMG as of May 2013. Using traditional relational database modeling, about 2 billion rows were needed to store the information about cassettes and their correlations based on COG and Pfam associations. In addition, there are additional tables for the different pre-calculated views. The construction and ingesting of the correlations as relational tables was taking about 16.5 hours. A typical search such as the step (iv) in Figure 3 could take 5-10 minutes.

With FastBit, only two tables with about 3.3 million rows are needed to store the gene cassette related information in addition to the FastBit data structure, and it takes 1.5 hours to load and index these tables. Building the index itself takes only 8 minutes, the rest of the time is to read the pre-processed cassette data out of the relational database system. Once the index is built, a typical search such as the step (iv) in Figure 3 takes now less than a minute, usually several seconds.

3. OVERVIEW OF FASTBIT INDEXING

Consider a table, such as a relational table, with rows (tuples) and columns (attributes). FastBit indexing uses a vertical data organization, also referred to as column organization. The use of this technique grew out of the need to search many variables from very large datasets. For example, a high-energy experiment consists of billions of collision objects, or "events," each associated with hundreds of searchable variables (properties). A typical search only involves a handful of variables, making it highly desirable to partition the data by variable, and to store their values into separate files, in order to avoid reading unnecessary data from disk. This way of organizing data is known in the database community as "vertical partitioning." It is well suited for scientific applications and data warehousing applications where existing records are not modified. In contrast, the tradi-

ID	X	bitmaps			
		b_0	b_1	b_2	b_3
0	2	0	0	1	0
1	1	0	1	0	0
2	3	0	0	0	1
3	0	1	0	0	0
4	3	0	0	0	1
5	1	0	1	0	0
6	0	1	0	0	0
7	0	1	0	0	0
8	2	0	0	1	0

Table 1: A bitmap representation of a column with 4 possible values

tional horizontal data organization requires all variable values of a record to be retrieved if any values are needed.

Typically, a bitmap index is built for a variable of a dataset. In the example shown in Table 1 we take a one-variable dataset. The variable is named X and its values are listed in the second column of the table. The first column under the heading of ID contains Row Identifiers used internally in most DBMS systems. The variable X contains integers with four possible values, $0 - 3$. Corresponding to these four values, there are four bitmaps, shown as four columns in the table on the right under the heading $b_0 - b_3$, each bitmap corresponding to one possible value of X. A bit in a bitmap is set to 1 if the value is equal to the value represented by the bitmap, 0 otherwise. For example, b_2 represents the value 2, and the first bit of b_2 is set to 1 because the value of X is 2 in the first row (with ID 0). To answer a query involving $X > 1$, one can produce a bitmap representing the rows that satisfy the condition by ORing b_2 and b_3. Similarly, other conditions involving X alone can be answered with bitwise logical operations.

The key technology in FastBit is a bitmap compression method called the Word-Aligned Hybrid (WAH) compression. Typically, bitmap indexes are too large to be stored in computer memory, and in order to answer a query using an index, the relevant part of the index must be moved into the computer's memory before computation. The conventional wisdom is that the time needed to read parts of the index into memory is much greater than the time needed to perform the computation after data is in memory. However, answering a query using compressed bitmap indexes in fact requires more time to perform computations on the bitmaps, much more than the time to read these bitmaps into memory. Therefore, the most direct way to improve the efficiency of a compressed bitmap is to perform computation on the compressed bitmaps without decompressing them. WAH was invented in 2004 specifically for this purpose. In many tests, WAH compressed indexes have been shown to be 10 times faster than the best-known bitmap indexes. In addition to being compute-efficient, WAH is also effective in reducing index size. The size of a FastBit index is on average one-third the size of the original data, which is about one-tenth the size of the widely used B-tree index.

The main idea of WAH is as follows. It uses a run-length encoding method, where long sequences of 0s (or 1s) are represented as counts in a word, and short sequences are represented lierally as-is. The reason for 31-bit groups is to align the count words and the literal words as explained next. This alignment is the key to performing logical operations over the bitmaps without decompression.

When a computer word has 32 bits, it logically divides incoming bitmap into 31-bit groups. A group with a mixture of 0s and 1s is stored literally in a 32-bit word, with the extra bit indicating it is storing literal bit values. Such a word is called literal word. Adjacent groups with only 0s or 1s are called a fill; 0-fill or 1-fill depending on the bit value. A fill is represented by its length and the bit value. WAH compression uses one word to present a fill, this word includes a bit to indicate that it is a fill, a bit representing the bit value of the fill and the remaining 30 bits represents the length in number of 31-bit groups. By insisting that the counting groups of 31 bits (rather than 32 bits), this method guarantees that all compressed words (*i.e.* literal and fill words) align. This facilitates the logical AND and OR computations required for queries to be performed directly on the compressed bitmap sequences. For further details refer to [9, 12, 11]. The FastBit software is available as open source, and instructions on downloading it can be found at `https://sdm.lbl.gov/fastbit/`.

FastBit was also proven to be optimal in that the amount of bitmap data touched for a query is proportional to the size of the result [10]. Thus, the more selective the query, the less time it takes to process it. Another property of FastBit, which is a result of compression and effective computation, is that it can be used for high cardinality data; that is, for variables that can assume a large number of possible values, which is the case with numeric and floating point data. As it will be discussed below, this feature was also useful for representing and processing the large number of possible functions (i.e., functional characterizations) of cassettes, where the 22,500 functions are stored as compressed bitmaps.

4. THREE TYPES OF QUERIES

For our development we used a real dataset of about 3.3 million cassettes with about 22,500 functions. One can think of this data as a table with 3.3 million rows and 22,500 columns. We will express our queries relative to this simple view. The average number of functions associated with each cassette is about 400 out of the 22,500 possibilities.

4.1 Query type 1: large-scale set-valued attribute

Consider the following query: "given a set of query functions, find all the cassettes that have all these functions." A variation on this query is "given a cassette and a set of target genomes, find all the cassettes in these genomes that have all the functions of the given cassette." In the first variant, the target cassettes are all the cassettes and in the second variant, the target cassettes are are only cassettes from the given genome. To answer this query, the set of query functions is a subset of the functions of the target cassettes.

In terms of a data model, the ID column in Table 1 is the cassette identifier in this case and the column X will the functions of each cassette. This *functions* column is "set-valued"

because each cassette is associated with a set of functions. Processing set-valued queries are known to require special algorithms [3, 4] and are not generally available on commnercial database systems. In the previous implementation of IMG, this one-column table is normalized following the common practice with relational database systems. This leads to a large correlation table with a column for cassette ID and a column for functions (where each row of the column has only a single function as the value). This organization produces a table of about 2 billion rows. Query 1 can be expressed in SQL as "select id from table where functions in (list-of-functions) group by id." Because the table size and the relatively large number of functions in a typical list of functions, this query often require several minutes to answer using a typical workstation.

In this work, we directly store the functions for each cassette as a set value and index this column with a bitmap index. Instead of having each bitmap represent whether or not a row has a specific value, the bitmaps are organized as shown in Figure 5. This bitmap index consists of 22,500 bitmaps and each bitmap is about 3.3 bits long.

In this representation, each function is associated with a bitmap and each bit in a bitmap corresponds to a cassette that has this function. If a cassette has a specific function, then the corresponding bit is set to 1, otherwise to 0. To find cassettes with a given set of functions, we take the bitmaps associated with these functions and perform bitwise AND operations among them. The resulting bitmap will have 1s corresponding to the cassettes which have all the specified functions. In short, with our bitmap representation, answering a type 1 query translates to a series of bitwise AND operations.

Using FastBit provides a natural solution for managing the bitmaps and performing bitwise AND operations, since it compresses the bitmaps, and performed logical operations directly on the compressed bitmaps. With FastBit, type 1 queries can be answered in a fraction of a second on a single processor (e.g., 0.07 seconds over 6 functions, 0.23 seconds over 20 functions) for the database size we had. This suggests that if the database grows by a factor of 100, queries can still be answered in seconds on a single processor. We note that these calculations were performed in memory since the entire compressed index in this case was only 309 MBs and fits in memory.

4.2 Query type 2: k-or-more matches on a set-valued attribute

This query type can be expressed as follows: "given a query cassette, find all the cassettes in the target genomes that have two-or-more of the functions of that cassette." In a more general form, a user may ask for k or more common functions with the query cassette. The reason for such a query is that biologists are interested in finding conserved regions (cassettes) which share at least two functions with cassettes from other genomes. An example of this is shown in Figure 6.

In theory, we could translate a type 2 query into a series of type 1 queries and therefore reuse the data structure presented above. However, this approach will produce a large

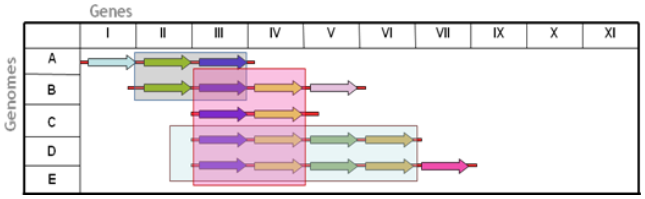


Figure 6: Example of two-or-more conserved regions across multiple genomes (organisms). Three examples are shown: in gray, 2 genes across 2 genomes; in pink, 2 genes across 4 genomes; in light blue, 4 genes across 2 genomes.

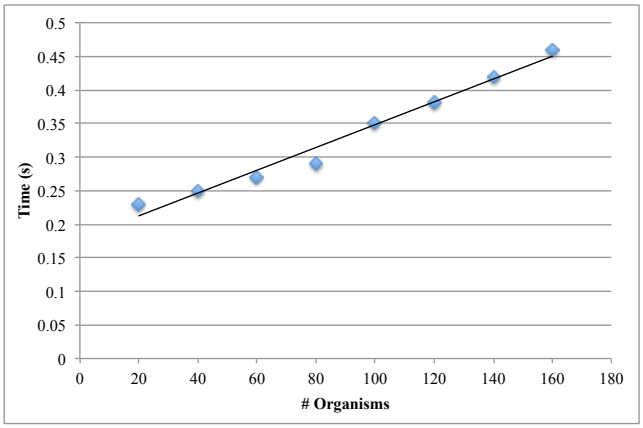


Figure 8: Time (seconds) needed to answer type 2 queries involving a different number of organisms.

number of type 1 queries for each type 2 query. Suppose that the number of functions for a cassette is 20 (a typical case, although many cassettes have hundreds of functions). We need to convert a type 2 query into a series of type 1 queries with all combinations of 2 target functions, all combinations of 3 target functions, and so on. This will produce 2^{20} combinations of target functions, or about one million type 1 queries for one type 2 query. Even if each type 1 query can be answered very quickly, answering one million of them could still take hours. For the example of 0.07 seconds mentioned above, such a search will require about 70,000 seconds, or about 20 hours. Clearly, translating type 2 queries into type 1 queries is not an effective approach.

We propose to answer type 2 queries with a horizontal representation of the bitmaps, as shown in Figure 7. In this representation, a bitmap is generated for each cassette and each bit in a bitmap corresponds to whether or not a particular function is present in the cassette. To find the common functions between two cassettes, we simply perform a bitwise AND on their associated bitmaps. To answer a type 2 query, we take the bitmap associated with the query cassette and perform a bitwise AND operation with the bitmap corresponding to each cassette in the target genomes. If the resulting bitmap has two or more 1s, then the corresponding cassette shared two or more functions. The threshold of two can be changed to another number k, and it is possible to constrain the number of common functions to be between k and m.

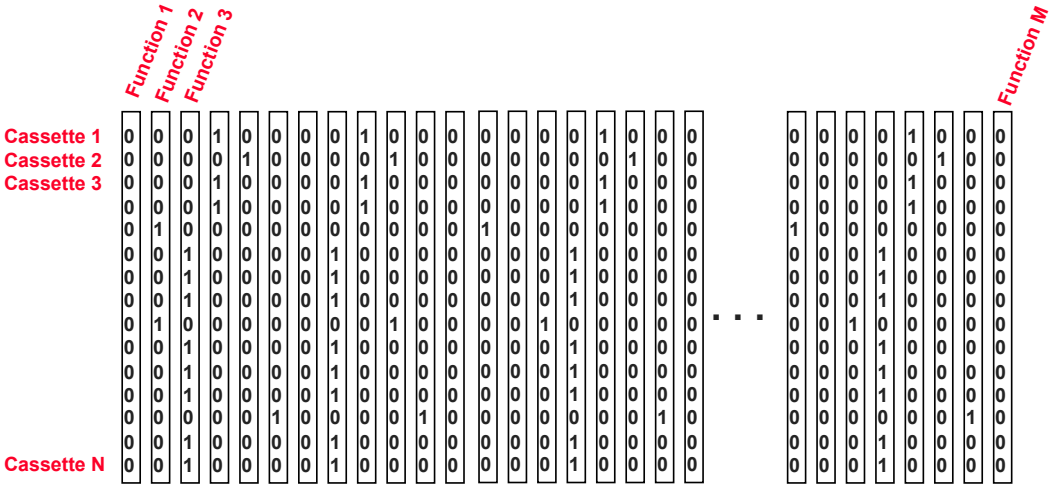


Figure 5: Representing cassette functions as columns of bitmaps.

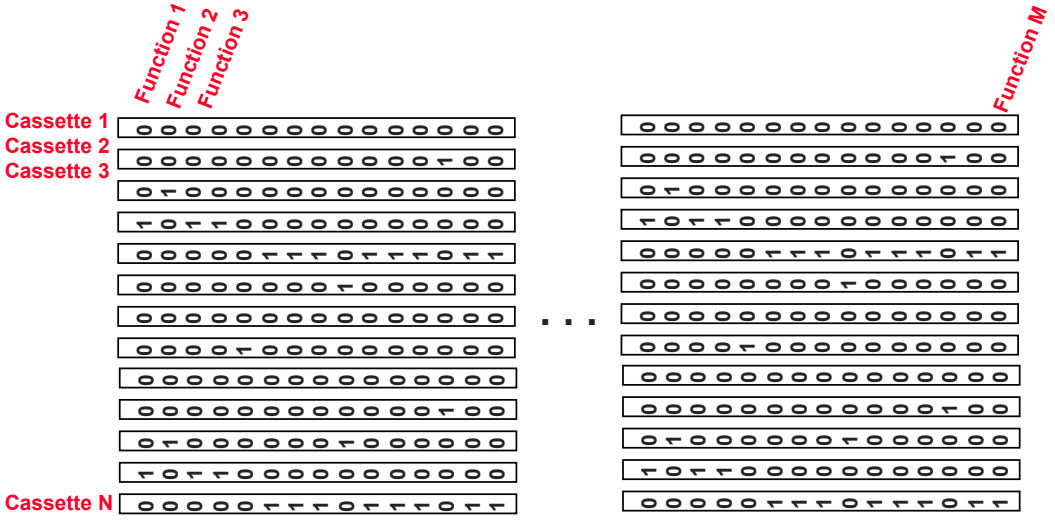


Figure 7: Horizontal organization of bitmaps facilitates two-or-more type queries.

Again, we use FastBit to manage the bitmaps and perform bitwise AND operations. Figure 8 shows that answering such a query against 20 to 160 genomes (organisms) took 0.2 seconds to 0.45 seconds on a single processor. Note that the number of cassettes per genome is about 400, and therefore the number of cassettes in 160 genomes is about 64,000. These are typically comparisons performed in actual use cases we observe. In the extreme case, performing this query against all 3.3 million cassettes took about 24.5 seconds. We can answer these queries efficiently because the compressed bitmaps we use fit in memory. Their total size is only 265 MBs.

When outputting the results from this query, it is useful to group the results according to the common properties the cassettes share. To support this requirement, we added a function to assign an order to the compressed bitmaps managed by FastBit. With this ordering function, we are able to take advantage of the standard template libraries to manage the cassettes with shared functions. Given two bitmaps, here are the rules we use to determine which bitmap is "smaller" than the other.

1. The bitmap with fewer total number of bits is "smaller."
2. For two bitmaps with the same number of bits, the one with fewer 1s is "smaller."
3. For two bitmaps with the same number of bits and the same number of 1s, the actual bit values are compared, in which case, a bit of 0 is considered smaller than a bit of 1.

4.3 Query type 3: k-or-more matches over a cross product of cassettes

This query type represents the most complex version of the comparison operations performed during gene context analysis. In addition to the two-or-more condition, it is desired to find cassettes that are in common within *all* specified genomes. This query type can be formulated as follows: "given a query genome and several reference genomes, find for each cassette in the query genome exactly one cassette from *each* of the reference genomes that have k-or-more functions in common". An illustration of this query is given in Figure 9.

Figure 10 (which is a more detailed view of Figure 3(iv)) shows a screen shot from the IMG system of such a request. Specifically, what is requested in the screen shot can be expressed as follows: "Find collocated genes (conserved gene regions) in Aeropyrum pernix K1 that are also collocated in Ignicoccus hospitalis & Staphylothemis marinus & Hyperthermus butylicus & Metallosphaera sedula". The result of this search is shown in Figure 11, where the cassettes found are shown in a table format.

We referred to this query type as the "killer query" because a type 3 query decomposes into a large number of type 2 queries where each type 2 query is composed by taking one cassette from the query genome and one cassette from each of the reference genome. This process of generating the type 2 queries is effectively producing a cross-product of all the cassettes in the query genome and the reference genomes. Clearly, the number of type 2 queries generated grows exponentially with the number of genomes involved in the type 3 query. For example, suppose we have 3 genomes with 2 cassettes each (one genome is given, and the other two are references). Then we need the cross product of 3 cassettes, one from each genome. The number of combinations is 2^3. In general, if we have on average X cassettes per genome, and Y genomes, the complexity is X^Y. Recall that each genome has on average about a few hundred cassettes, hence even a query which only involves a few genomes could produce a tremendous number of type 2 queries.

To simplify the processing of this type of query, we consider only "maximal" answers. For example, if a pair of cassettes have four functions in common, then they also have 4 combinations of three functions in common and 12 combinations of two functions in common. In this case, the "maximal" answer is the single set with 4 functions in common. We will not list out the combinations with 3 functions or 2 functions even though they are also valid answers.

To further prune the solution space that needs to be examined, we take advantage of the following two properties: the first is the commutative property in forming a cross product. Because of this property, it is possible to progress step-wise; *i.e.* find common cassettes between two genomes, take the result and apply it to cassettes from the third genome, etc. as shown in Figure 12. The second property that makes this algorithm effective, is that in practice there are many functions which are not common to all the cassettes, and therefore do not persist as we progress from each step to the next.

We note that given a correlation table representation of the form (cassette, function), it is possible to formulate SQL queries that will express this type of query. However, our interest is in running such queries in near real-time, i.e within seconds. Our experience with the IMG system that uses a relational database was that such queries take many minutes, and therefore all possible combinations were pre-computed and stored as views. As the database grew larger, this solution was not scalable. Running such queries with compressed bitmap technology has proven to be far superior, and provide query responses withing seconds as illustrated in Figure 13.

As mentioned previously, the answers to a type 2 query are ordered according to the bitmaps representing the common functions. This data structure allows us to efficiently record the combinations of cassettes sharing the same properties and allows us to find shared functions with additional cassettes easily without going back to the original bitmaps associated with the cassettes. Furthermore, it reduces the number of bitwise logical AND operations needed because combinations of cassettes sharing the same functions only need to participate in the AND operation once. This is another algorithmic detail that significantly reduces the computational cost.

In theory, the number of possible combinations of functions grows exponentially with the number of genomes involved. In practice, the number of combinations of functions that ac-

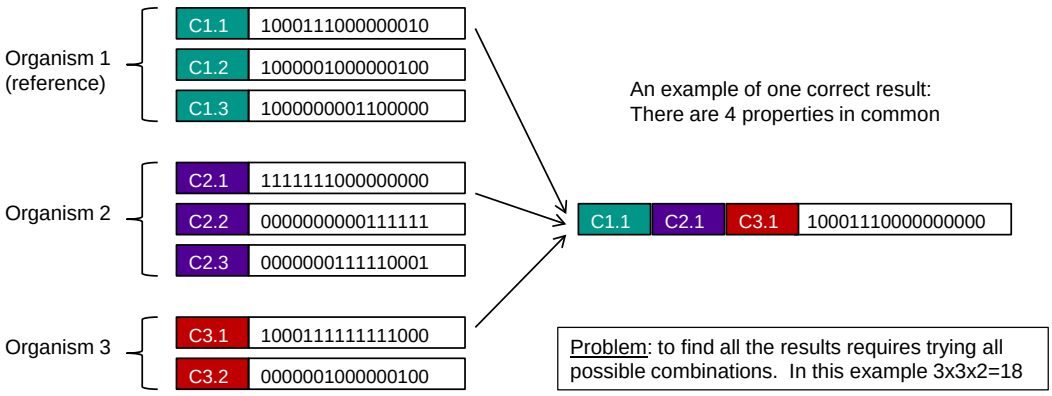


Figure 9: An example of an instance where one cassette from each organism has two-or-more properties in common.

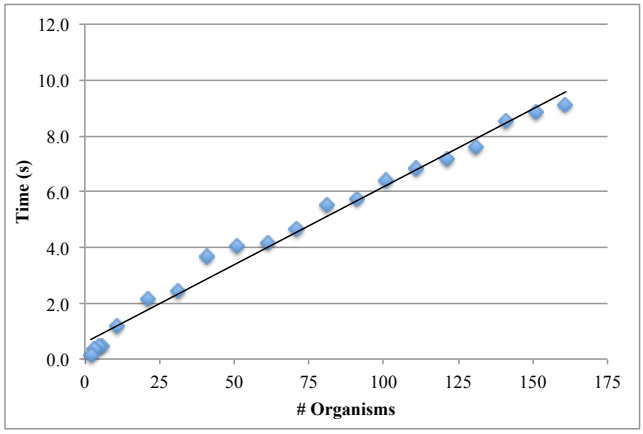


Figure 13: Time (seconds) needed to answer type 3 queries involving a different number of organisms. The solid line is the linear regression line.

tually appear in our list grows much slower because there are relatively few cassettes with long lists of functions in common. As more and more genomes are introduced into our algorithm, the number of unique combinations of functions that we need to keep track of often decreases. Therefore, our solution algorithm does not scale exponentially with the number of genomes, but in fact it scales nearly linearly with the number of genomes as shown in Figure 13. In our tests, as many 161 organisms are used in a single type 3 query. Even in this case, it took less than 10 seconds. We did try queries involving more organisms, however, they all produce no answers and therefore do not require more time than 10 seconds either. Note that using the traditional DBMS, users rarely dare to try a type 3 query with more than a couple of genomes because a type 3 query involving k genomes will require a k-way self-join. As k increases, the amount of time needed to answer the query can and does grow exponentially.

5. SCALING UP

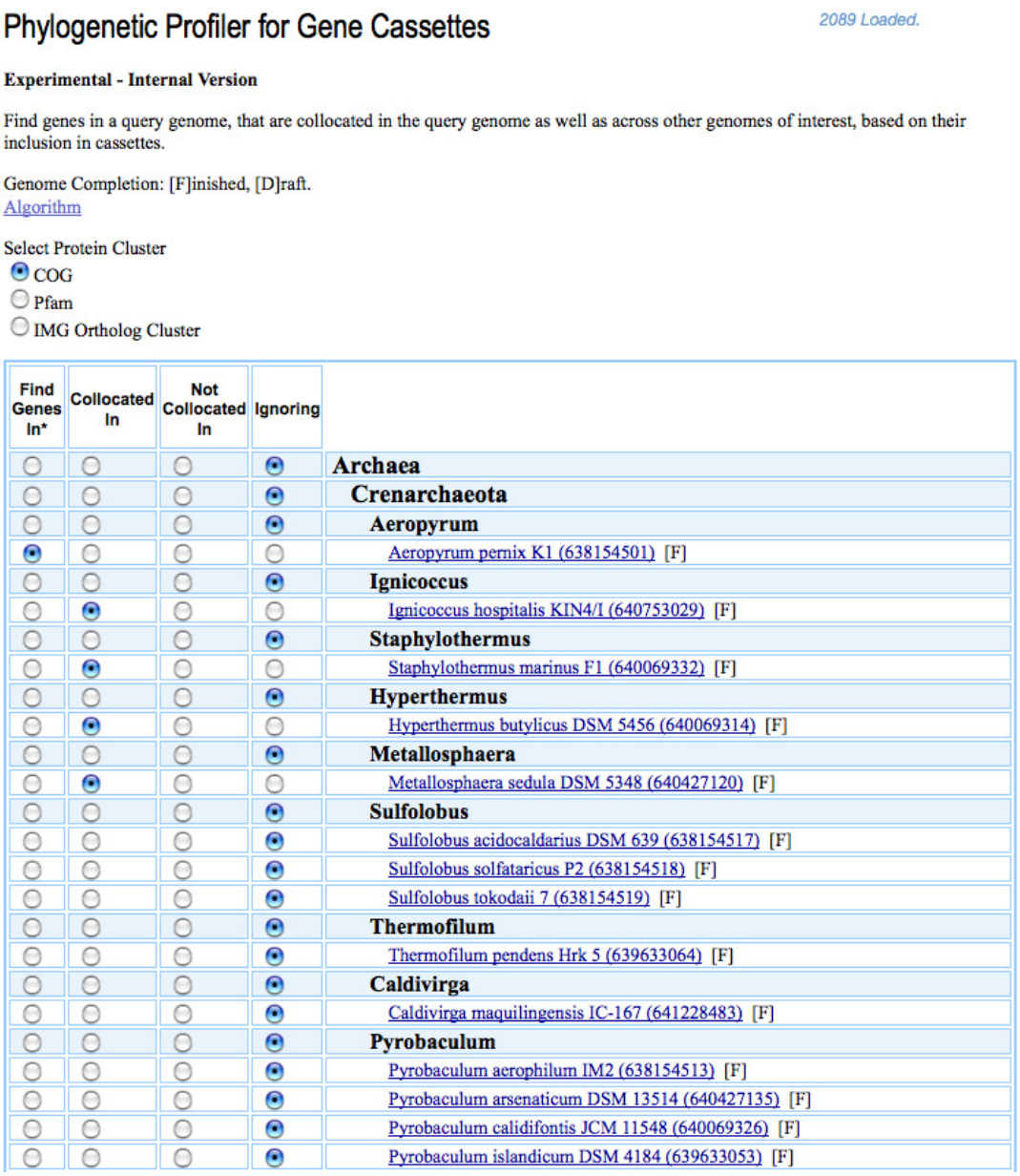
So far the requirements of the IMG system involve millions of cassettes. However, the number of genomes sequenced is growing faster than Moore's law (doubling every 18 months). One of the reasons is the sequencing of combination of genomes, called meta-genomes. For example, a soil sample has in it a large number of bacteria, each with its own genome. Sequencing the combination of genomes introduces volume and complexity at unprecedented levels. We are faced with the question whether bitmap techniques would easily scale using multi-core hardware.

Fortunately, the FastBit technique lends itself to parallelization. For vertical organization of bitmaps, it is possible to partition the bitmaps by a pre-selected number of rows. For example, if we have 1 billion rows, it would be possible to partition them into 100 chunks of bitmaps for every 10 million rows, with each chunk to be provided to one of the 100 cores for parallel processing. This is possible because of the property that each compressed word (whether a count word or a literal word) has matching boundaries because it is a multiple of 31 bits as explained in Section 3. However, the compression may produce unbalanced chunks, depending on how uniform the bitmaps are. For example, the first 10 million rows could compress better than the second 10 million rows, etc. Thus, we may need to develop algorithms which will partition bitmaps into balanced chunks dynamically, by adjusting the number of rows accoding to their compression efficiency.

In addition, combining the results from each chunk once the parallel processing completes may benefit from a tree structured approach as shown in Figure 14. Here again, the partitioning may not be balanced because of the variable number of cassettes in each genome, and the compression factor. We expect though that dynamic balancing will benefit this parallelization process as well. In this case, we believe that a good strategy is to assign work to processing nodes based on the total size of chunks given to each node. We will need to consider such strategies once we scale up to billions of cassettes.

6. CONCLUSIONS

In this paper, we described queries in a biological domain which cannot be efficiently processed in a conventional relational database (or other database systems, such as NoSQL



Phylogenetic Profiler for Gene Cassettes

2089 Loaded.

Experimental - Internal Version

Find genes in a query genome, that are collocated in the query genome as well as across other genomes of interest, based on their inclusion in cassettes.

Genome Completion: [F]inished, [D]raft.
Algorithm

Select Protein Cluster
COG
Pfam
IMG Ortholog Cluster

Find Genes In*	Collocated In	Not Collocated In	Ignoring	
○	○	○	◉	**Archaea**
○	○	○	◉	**Crenarchaeota**
○	○	○	◉	**Aeropyrum**
◉	○	○	○	Aeropyrum pernix K1 (638154501) [F]
○	○	○	◉	**Ignicoccus**
○	◉	○	○	Ignicoccus hospitalis KIN4/I (640753029) [F]
○	○	○	◉	**Staphylothermus**
○	◉	○	○	Staphylothermus marinus F1 (640069332) [F]
○	○	○	◉	**Hyperthermus**
○	◉	○	○	Hyperthermus butylicus DSM 5456 (640069314) [F]
○	○	○	◉	**Metallosphaera**
○	◉	○	○	Metallosphaera sedula DSM 5348 (640427120) [F]
○	○	○	◉	**Sulfolobus**
○	○	○	◉	Sulfolobus acidocaldarius DSM 639 (638154517) [F]
○	○	○	◉	Sulfolobus solfataricus P2 (638154518) [F]
○	○	○	◉	Sulfolobus tokodaii 7 (638154519) [F]
○	○	○	◉	**Thermofilum**
○	○	○	◉	Thermofilum pendens Hrk 5 (639633064) [F]
○	○	○	◉	**Caldivirga**
○	○	○	◉	Caldivirga maquilingensis IC-167 (641228483) [F]
○	○	○	◉	**Pyrobaculum**
○	○	○	◉	Pyrobaculum aerophilum IM2 (638154513) [F]
○	○	○	◉	Pyrobaculum arsenaticum DSM 13514 (640427135) [F]
○	○	○	◉	Pyrobaculum calidifontis JCM 11548 (640069326) [F]
○	○	○	◉	Pyrobaculum islandicum DSM 4184 (639633053) [F]

Figure 10: Selection of one query genome and 4 reference genomes in an IMG interface for query type 3.

Select	Result Row	Gene Id	Gene Name	Length	Cassette Id	Conserved Neighborhood Viewer Centered on this Gene
☐	1	640082911	50S ribosomal protein L6P	561	319640069310 (100544975)	Hbut_1316
☐	2	640082912	50S ribosomal protein L32E	369		Hbut_1317
☐	3	640082913	50S ribosomal protein L19E	462		Hbut_1318
☐	4	640082914	50S ribosomal protein L18P	636		Hbut_1319
☐	5	640082915	50S ribosomal protein S5p	642		Hbut_1320
☐	6	640082916	50S ribosomal protein L30P	555		Hbut_1321
☐	7	640082917	50S ribosomal protein L15P	456		Hbut_1322
☐	1	640083164	DNA-directed RNA polymerase subunit A\'	2646	379640069310 (100387328)	Hbut_1574
☐	2	640083165	DNA-directed RNA polymerase subunit A"	1275		Hbut_1575
☐	3	640083166	50S ribosomal protein L30e	339		Hbut_1576
☐	4	640083167	putative transcription elongation factor	432		Hbut_1577
☐	5	640083168	30S ribosomal protein S12P	444		Hbut_1578
☐	1	640082128	conserved uncharacterized protein	1182	133640069310 (100076473)	Hbut_0521
☐	2	640082129	universally conserved protein	1431		Hbut_0522
☐	3	640082130	predicted ABC transporter	756		Hbut_0523
☐	1	640082900	50S ribosomal protein L22	471	319640069310 (100257081)	Hbut_1305
☐	2	640082901	30S ribosomal protein S3P	744		Hbut_1306
☐	3	640082903	ribonuclease P protein component 1	336		Hbut_1308
☐	1	640082255	conserved crenarchaeal protein	288	156640069310 (100368135)	Hbut_0650
☐	2	640082256	hypothetical protein	816		Hbut_0651
☐	3	640082257	universally conserved protein	996		Hbut_0652

Figure 11: The result of the query expressed in Figure 10. Three cassettes were found that are common to ALL reference genomes. One cassette consists of 7 genes, one of 5 genes and 3 of 3 genes.

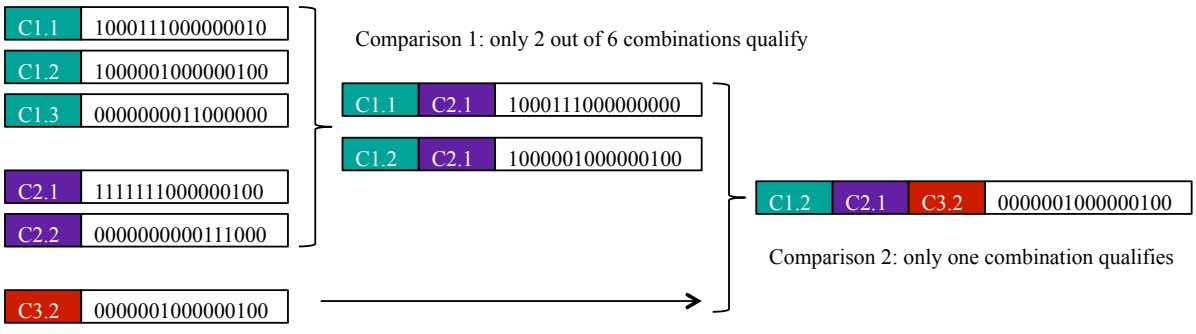


Figure 12: Taking advantage of commutative property for query type 3. In this example, genome 1 has 3 cassettes, genome 2 has 2 cassettes, and genome 3 has 1 cassette.

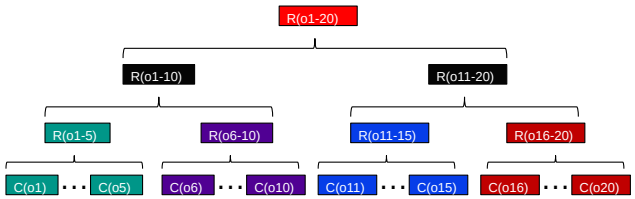


Figure 14: Tree-like processing of cassettes in multiple genomes in parallel. In this example, cassettes from 20 organisms (labeled C(1) through C(20) were partitioned into 4 groups each having cassettes from 5 genomes. The resulting function tuples from each is passed on to the second level of the tree, etc.

databases). These types of queries benefit greatly from a bitmap representation of the data, and compression techniques over bitmaps that support efficient processing of logical operations. The queries fall into three types: (1) AND conditions over a set-valued attribute with very high cardinality; (2) performing k-or-more search combinations over the set-valued attribute; and (3) performing searches over groups of set-valued attribute values, where matches are found in all groups.

We have shown that for dealing with such queries a compact representation of the data is needed in the form of compressed bitmaps. We have also shown that specialized processing is necessary for each type of query. We also discussed potential extensions to our approach for higher volumes of biological sequence data expected in the near future. We believe that these techniques can also apply to other biological pattern analysis of set-valued attribute structures.

7. ACKNOWLEDGEMENTS

Support for this work was provided through Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research under Contract No. DE-AC02-05CH11231.

8. REFERENCES

[1] P. M. Bowers, M. Pellegrini, M. J. Thompson, J. Fierro, T. O. Yeates, D. Eisenberg, et al. Prolinks: a database of protein functional linkages derived from coevolution. *Genome Biol*, 5(5):R35, 2004.

[2] R. Finn, J. Mistry, J. Tate, P. Coggill, A. Heger, and et al. The pfam protein families database. *Nucleic Acids Research*, 38:D211–D222, January 2010.

[3] S. Helmer and G. Moerkotte. A performance study of four index structures for set-valued attributes of low cardinality. *The VLDB Journal*, 12(3):244–261, 2003.

[4] N. Mamoulis. Efficient processing of joins on set-valued attributes. In *SIGMOD*, pages 157–168. ACM, 2003.

[5] V. Markowitz, I. Chen, K. Palaniappan, K. Chu, E. Szeto, and et al. Img: the integrated microbial genomes database and comparative analysis system. *Nucleic Acids Res.*, 40:D115–D122, 2012. See also `http://img.jgi.doe.gov/`.

[6] K. Mavromatis, K. Chu, N. Ivanova, S. Hooper, V. Markowitz, and N. Kyrpides. Gene context analysis in the integrated microbial genomes (IMG) data management system. *PLoS ONE*, 4(11):e7979, 2009.

[7] R. Overbeek, M. Fonstein, M. D'Souza, G. Pusch, and N. Maltsev. The use of gene clusters to infer functional coupling. *PNAS*, 96(6):2896–901, 2009.

[8] R. Tatusov, N. Fedorova, J. Jackson, A. Jacobs, B. Kiryutin, and et al. The cog database: an updated version includes eukaryotes. *BMC Bioinformatics*, 4:41, 2003.

[9] K. Wu, E. Otoo, and A. Shoshani. On the performance of bitmap indices for high cardinality attributes. In *On the Performance of Bitmap Indices for High Cardinality Attributes*, pages 24–35, 2004.

[10] K. Wu, E. Otoo, and A. Shoshani. Optimizing bitmap indices with efficient compression. *ACM Transactions on Database Systems*, 31:1–38, 2006.

[11] K. Wu, A. Shoshani, and K. Stockinger. Analyses of multi-level and multi-component compressed bitmap indexes. *ACM TODS*, 35, 2010.

[12] K. Wu, K. Stockinger, and A. Shoshani. Breaking the curse of cardinality on bitmap indexes. In *Statistical and Scientific Database Management – SSDBM*, pages 348–365, 2008.