A Bitwise Encoding Scheme Designed to Improve the Speed of Large Scale Gene Set Comparison

Tham H. Hoang, Pujan Joshi, Seung-Hyun Hong and Dong-Guk Shin
Department of Computer Science and Engineering, University of Connecticut, Storrs, CT, USA

Abstract—Comparing one gene set against a large number of gene sets is a method that is ubiquitously used in many high-throughput gene expression studies. A conventional way of comparing one gene set against a large number of target gene sets is through database programming. However, this approach can be inefficient if the sizes of the target gene sets become massively large. We propose a coding scheme designed to convert both query and target gene sets into bit streams and use bitwise Boolean operations such as intersections and set differences for comparisons. Our approach can efficiently represent gene sets containing a small number of gene identifiers, thus suggesting the feasibility of storing a large amount of target gene sets into memory entirely. Our method drastically reduces disk I/Os which are inevitable in the database programming approach. Our method can offer a constant time performance independent of the size of the target gene sets.

Keywords: Enrichment analysis, bitwise operators, query system, encoding method, gene set interactions.

1. Introduction

The current biomedical research community faces a significant challenge in handling the genomics data that is inundating the field. Particularly, the recent advance in Next Generation Sequencing (NGS) technology has been dramatically changing the way biomedical scientists analyze experimental data. One noticeable trend has been: (i) a group of closely collaborating scientists produces massive amounts of data with a specific agenda in mind, (ii) they deposit the collectively obtained data sets in a public repository, and (iii) the public uses them as references for various types of research inspired by an individual laboratory’s interest. There are multiple examples of such community generated genomics data projects. The 1000 Genomes project aims at uncovering structural variations using whole genome sequencing on samples taken from over 2500 individuals with known geographic and ethnic origins [1]. TCGA is to explore the entire spectrum of genomic changes involved multiple human cancer types [2]. ENCODE aims at identifying key functional elements in the human genome [3]. In addition there is a Gene Expression Omnibus (GEO) which is a “catch-all” database repository of many different types of high throughput gene expression data which were submitted by thousands of scientists [4]. All these public data sets wait for utilization by bioinformaticians or computational biologists at large.

We have been analyzing numerous high-throughput gene expression data in attempts to help many local biomedical scientists derive meaningful interpretations from their respective experiments. Steps include, for example, subgrouping gene expression patterns using cluster methods like a Pattern-Based clustering (PBC) program [5] and then performing the subsequent “meta-analyses” [6]. Typically two types of meta-analysis can be attempted: (i) enrichment analysis (e.g., Webgestalt [7]) and (ii) comparing the locally generated data with the similar publicly available or related ones. Both approaches involve gene set comparisons. Enrichment analysis includes comparing a “query gene set” (e.g., differentially expressing “DE” genes) against the “target gene sets” such as well-curated collections of gene sets in Gene Ontology or pathway databases (e.g., KEGG). The goal is to find whether or not query set includes an unexpectedly large number of genes of some a priori known functional group(s). For example, biologists can discover if the identified DE genes clearly point (towards) that “Wnt signaling pathway” is suppressed or whether they are more or less involved in the biological process known as “osteogenesis” and so on. Typically Fisher test is used to produce a p-value which measures the statistical significance of the enrichment. Unlike the first approach, the second approach comparing the DE gene set against the data set available from the public repository is not well-established. In fact, how to carry out the second type of analysis is an active area of bioinformatics research; there are many alternative ways of exploiting the genomics data publicly available and how to use data to better analyze DE genes is a widely-regarded question. Nevertheless, we conjecture that most of the comparison studies will involve comparing DE genes against some target gene sets of many different types (e.g., gene sets derived from ChIP-Seq, ChIA-PET, microRNA, RNA-Seq, DNA-Seq, etc.). As the scale and magnitude for such gene set comparison is rapidly increasing, it is imperative to design an efficient way of carrying out the gene set comparison.

2. Problem Formulation

Figure 1 contrasts the two methods for doing the gene set comparison, one using the conventional database programming and one using the coding scheme we report in this paper. First, lets say there is a very large collection of gene sets, say, a database of target gene sets, which is denoted by $DB_t = \{G_1, G_2, \ldots, G_n\}$. Here each $G_i, 1 \leq i \leq n$, includes one or more gene identifiers. Let there be, namely, a query gene set, $G_q$, which also includes one or more gene identifiers. The goal is how fast one can compare the query set $G_q$, with each gene set stored in $DB_t$, which should be typically stored in a database. Figure 1a illustrates the database programming approach. One can write a SQL statement that performs intersection between $G_q$ and each
gene set in $DB_t$. In this approach the DBMS will fetch a group of gene sets from the database (via disk I/O) and carry out the intersection in memory. Figure 1b illustrates the scenario in which someone designs an effective data compression method, the entire gene sets of $DB_t$ can be stored in the memory and the entire intersection operations can occur in memory. As the figure suggests, each target gene set of $DB_t$ is encoded once and the encoded gene set is stored in the disk. This encoded gene set of $DB_t$ is read once and can reside in memory and is repeatedly used in comparing a series of input query gene sets. Our coding scheme approach would be far more efficient than the SQL based approach because processing of each target gene set will not incur any additional disk I/O which is not the case in the SQL approach.

![Fig. 1: SQL-based approach and coding approach with memory and disk I/O usage in both approaches.](image)

One issue in our approach is estimating realistically how big a $DB_t$ our coding scheme can store in memory of an ordinary computer workstation. Our plan is given a byte (8 bits), we use one bit to present a gene identifier. Each byte ranges from 00000000 to 11111111 and each bit position in the byte is tied to a gene identifier uniquely. Each byte can then represent up to 8 gene identifiers depending on which bit is set to “1” as opposed to “0”. If we assume the size of the entire known human genes is 24000, this coding scheme will entail up to 3000 bytes (= 24000/8) to express a gene set including the entire known human gene identifiers. If we estimate that a typical DE gene set has less than 1000 genes in it, then this coding scheme may need to use up to 1000 bytes. The worst case is when only one bit is set in each byte. A different coding scheme can be used which can squeeze in up to $2^8 - 1 = 255$ gene identifiers in a byte. In this case only about 100 bytes would be enough to store the entire 24000 genes. The former encoding scheme has a lesser data compression factor but the bitwise operation needed in this scheme is far more straightforward than the one needed in the latter coding scheme. In the former scenario, DE sets from one million experiments can be stored in 1GB memory which is affordable in typical workstation computers.

The rest of the paper is organized as follows. In Section 3, we present related works of using bitwise coding schemes in genomics data analysis. In Section 4, we provide the basics of bitwise data structure and the associated Boolean operations. We also discuss other topics related to motivating the use of the encoding scheme. In Section 5, we describe the details of how gene sets are encoded and decoded, and how encoded gene set data structure is processed. The running time complexity of the operation is also analyzed and discussed. Section 6 is the conclusion.

3. Related Works

Two of the tools applying bitwise data structure and operations to perform high-throughput data analysis are BiForce [8] and BOOST (BOolean Operation-based Screening and Testing) [9]. In BiForce, the goal is to uncover gene-gene interactions (epistasis) from genome-wide association studies (GWAS). It uses Boolean bitwise operations and multithreaded parallelization to speed up comparisons of billions of single nucleotide polymorphism (SNP) combinations obtainable from a pair-wise genome scan. Its coding scheme introduces three types of genotype in which the type is represented by the values of the set \{0, 1, 2\}. Each value is mapped to a set of 3 bits $S_i^0, S_i^1, S_i^2$ where $S_i^j, 0 \leq i \leq 2$, is set to 0 or 1 among 3 bits depending on whether the genotype is present or missing. By default, all bits are set to 0. If a genotype is present, one and only one of 3 bits is turned on as 1, otherwise, none of 3 bits is set. For example, let $S_1^0 = [1 0 1]$ denoting genotype 0, and $S_2^3 = [1 1 1]$ denoting genotype 1. Then by using the bitwise AND operation on each pair in $S$’s and counting the 1’s of the outcome, one can produce the binary traits table, say, $n_{1,2} = |S_1^0 \land S_2^3| = |1 0 1 \land 1 1 1| = [1 0 1] = 2$. The authors report that BiForce completed analyses of the eight metabolic traits within 1 day on a 32-node cluster computer, identifying nine epistatic pairs of SNPs in five metabolic traits and 18 SNP pairs in two disease traits using two sets of about 340K SNPs from GWAS cohorts.

BOOST also uses bitwise data structure and operators on epistasis data obtainable from GWAS. The authors introduce a Boolean representation of genotype data. Each row is to represent one specific type of genotype consisting of two-bit strings: one obtained from control samples and the other obtained from test samples. Each SNP has 3 rows for three types of genotypes instead of one row as in BiForce. Each bit in the string represents one sample, and its value (0 or 1) indicates whether the sample has the corresponding genotype or not. The authors give an easy to follow example in [9] illustrating that the method uses the bitwise operation AND on each corresponding row, then counts the number of 1’s (e.g., hamming weight) taking the advantage of 64 bit AND operation.

There have been other works in bioinformatics that also exploit bitwise operation. A coding scheme and bitwise operators have been applied to mapping high-throughput bisulfite sequencing reads to the reference genome in the system called BSMA [10]. BSMA combines genome hashing and bitwise masking to achieve bisulfite mapping where DNA sequences are converted into binary strings by encoding each DNA nucleotide into two bits (i.e., A:
00, C: 01, G:10, T:11). The authors suggest that due to this encoding scheme high-throughput bisulphite reads can be mapped at whole genome level with feasible memory and CPU usage. We note that none of the aforementioned existing works uses the coding scheme of our work presented in this paper.

High throughput computing requires that the tool should aim to “gene sets” instead of individual gene. Gene set interactions with multiple categories have been integrated into an analysis toolkit called WebGestalt from Wang et al. [7]. Recursively applying on a pair interaction makes it possible to combine information from any number of gene sets. However, it has a limitation of performing multiple combinations at the same time in some experiments. Oftentimes, the comparisons of all combinations will show a bird’s eye perspective and better understanding the biological system. We also want to calculate the information content variables (e.g., information gain) for enrichment analysis of gene set interactions. Other interactions such as gene and gene or gene and environment ones have been discussed [11], and in general perspective of genetics [12].

4. Methods

4.1 Bitwise Conversion

Boolean logical operation has been used widely for the query system which computes typical comparisons such as intersection and set difference. In order to compare two gene sets using bitwise operator, we need to convert the gene set into bits. For example, let A and B be two DE gene sets obtained from two experiments. Performing intersection of the two gene sets could be finding the genes that are consistently up- or down-regulated in both experiments. Difference query displays the genes that are expressed in experiment A but not in experiment B, or vice versa [7]. Figure 2 is an example of bitwise encoding and bitwise AND operation between query and target gene sets.

![Fig. 2: A hypothetical example of performing logical bitwise AND operator between query gene set and target gene set.](image)

Bitwise encoding is a conversion process of the query and target gene sets into coded key and value map. We will segment the total genes into an eight bit format. For example, let 20000 be the total number of universe genes. Then we will have 20000/8 = 2500 keys (bytes) with each key representing presence of 8 genes in a gene set. The genes in key $k$ ($1 \leq k \leq 2500$) represents genes which have index from $(k-1) \times 8 + 1$ to $k \times 8$. If gene $i$ exists, the gene position index will be calculated using modular 8 calculation, because we have an 8 bit structure. In Figure 2, in query gene set, key 1 has one gene (gene 1), key 2 has 1 gene (gene 9), and key 3 has 4 genes (17, 21, 22, 23), and key 2500 has one gene (19993). Similarly target gene set has genes with ids of 5, 9, 17, 22, 23, 19996. We apply AND bitwise operator between value of a specific key of query gene set and the same key in the target gene set. Genes with ids of 9, 17, 22, 23 are in common and there is no common gene in key 2500.

4.2 Bitwise Coding Approach for Gene Set Comparison

In this section, we describe new bitwise coding approach and compare it with the SQL-based approach. The work flow of bitwise approach is shown in Figure 3.

![Fig. 3: The work flow of coding approach has three steps: encoding, operating and decoding. Encoding phase has two off-line steps for encoding universe gene set and all the target gene sets once into memory. Decoding phase includes one off-line step to create decoding map.](image)
produce various parameters (e.g., information gain, ratio, count, etc.) which will be needed to derive biological interpretations. One noticeable fact is that such comparison can be performed entirely in memory (see Figure 3). Decoding phase is to translate the comparison output obtained in bitwise structure into a list of gene identifiers. This translation step will need to use a decoding table that specifies how the key and value of each gene are mapped into gene identifiers. Such decoding will be an one-time process and can be done off-line only when doing so is needed. The decoding process is \((key - 1) \times 8 + value\). For example, if the commonly found gene has \(key = 10\), \(value = 2\) then it is \((10 - 1) \times 8 + 2 = 74^{th}\) gene in the universe gene set. Three steps denoted in Figure 3 as “Offline” are processes of encoding the universe gene set, encoding the target gene set and generating the decoding map.

Figure 4 shows the work flow of the alternative SQL-based approach with two major steps: loading phase and operating phase. Loading phase includes reading target and query gene sets. Here, the DBMS query system will be responsible for generating the output of comparing the input query gene set and the database stored target gene sets. This SQL-based approach will incur disk I/Os to bring in blocks of target gene sets into the memory which would become the dominating factor responsible for making the SQL-based comparison process slow.

### 4.3 Computing Information Gain

To illustrate how bitwise operations can be used on coded data structure, we show an example of computing information gain by comparing a pair of gene sets. Figure 5 depicts the overview of the process, which is analogous to construct a decision tree [13]. In Figure 5, each node of the tree contains a question. Every internal node points to one child node as a possible answer to its question and the questions are formed into a hierarchy. In decision trees, one of the most commonly used measurements is entropy. To estimate how a pair of gene sets are inter-related, let \(H(X)\) and \(H(Y)\) be gene set entropies for \(X\) and \(Y\), respectively. Overlapped region in Figure 6 is \(I(X;Y)\), the mutual information derived from both patterns (e.g., joint distribution) and \(H(X|Y)\) and \(H(Y|X)\) denote conditional distributions (see Figure 6).

\[
\begin{align*}
\text{Gene set} & \quad \text{Shared} & \quad \text{Shared} & \quad \text{Information} \\
\text{pair} & \quad \text{genes?} & \quad \text{function?} & \quad \text{gain} \\
A-B & 36 & \text{Yes} & 0.004 \\
A-C & 0 & \text{No} & 0 \\
A-D & 32 & \text{No} & 0.002 \\
\end{align*}
\]

\[
I(X;Y) = H(X) + H(Y) - H(X,Y) \quad (2)
\]

\[
X = \{x_1, x_2, \ldots, x_n\}
\]

\[
\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)
\]

\[
X \text{ is a set of possible values } x_1, x_2, \ldots, x_n \text{ and corresponding probability distribution } p(x_i) = P(X = x_i), i = 1, 2, \ldots, n. \text{ The entropy of } X \text{ is } H(X), \text{ calculated as in Equation } 1.
\]

\[
H(X) = \sum_{i=1}^{n} p(x_i) \log_2 p(x_i)
\]

\[
\text{Mutual information is defined as the mutual information in both random gene sets } X \text{ and } Y \text{ in Equation } 2. 
\]

\[
I(X;Y) = H(X) + H(Y) - H(X,Y)
\]
Target gene set and query gene set are assumed to be statistically independent. The conditional entropy \( X \) given \( Y \) can be measured. Likewise, the conditional entropy \( Y \) given \( X \) can be computed using Equation 3.

\[
H(X|Y) = H(X) - I(X;Y) = H(X,Y) - H(Y) \\
H(Y|X) = H(Y) - I(X;Y) = H(X,Y) - H(X)
\]

(3)

Figure 5 is given to illustrate the use of information gain in solving the following problem: Given a query gene set \( A \) and the three target gene sets \( (B, C, D) \), can one determine which target gene set is more predictable of the given query gene set? To solve this problem we need to build the contingency table as illustrated in Figure 6 for each pair of gene sets (e.g., \( A-B, A-C \) and \( A-D \)). The table has four components which are labeled, respectively, true positive (TP), true negative (TN), false positive (FP) and false negative (FN). Given the pair of target and query gene sets, TP value is number of common genes between the two gene sets. TN/FP is number of genes present in query/target gene set but absent in target/query gene set. FN value is the number of universe genes that are neither in the target gene set nor in the query gene set. We discuss in the subsequent section how fast information gains can be computed using our proposed encoding scheme.

5. Case study and Discussion

In our case study, we have implemented bitwise coding method and compared the results with SQL approach using a randomly generated dataset of 30 query gene sets \( (Q_1, Q_2, \ldots, Q_{30}) \) and 1000 target gene sets \( (T_1, T_2, \ldots, T_{1000}) \). The number of genes in each set varies from 51 to 499 and is randomly selected from universal pool of 20000 genes. The target gene sets are encoded and stored in variable number of blocks ranging from 73 to 143. We have analyzed this dataset by running both SQL and bitwise approach to identify top five pairs \( P_i \) of \( (Q_j, T_k) \) for two comparison cases: Intersection and Mutual Information Content. Our goal is to identify intersection between query gene sets \( Q_s \) and target gene sets \( T_s \) and also to identify mutual information contents (difference) in each pair. To balance the environmental factors, we run each experiment 10 times and report average running time. We also discuss time complexity of the methods and briefly explain the performance benefit.

5.1 Gene Set Intersection

Algorithm 1 identifies top five pairs of \( Q_s \) and \( T_s \) for highest intersection of genes. In this algorithm, gene sets are compared using conventional SQL-based approach where two tables are joined and row counts are generated using an Oracle DBMS. We compare all the possible pairs and rank the pairs based on the number of common genes.

In algorithm 2, we have used bitwise approach to achieve the same goal as Algorithm 1. The target gene sets are assumed to be already encoded and stored in the database. The encoded sets are loaded into the memory in key-value HashMap for each block. This facilitates efficient comparison between two sets using the block. A fast bitwise AND operation is performed for every common block of both sets. A result map is created and result of bitwise operation is stored as values for those corresponding keys.

Algorithm 1 Generate_Top5_Intersection_use_SQL

1: Initialize variables and database connection.
2: Read target gene set.
3: Read query gene set.
4: for (all queries) do
5: Read target gene set and match gene id and output the ordered list of gene set id and count.
6: Get the first element of list
7: Look up the Map with this element and store query list.
8: end for
9: Sort the query list.
10: Output top five of gene set combinations.

Algorithm 2 Generate_Top5_Intersection_use_Bitwise

1: Initialize variables and environment, database connection.
2: Read target gene set.
3: Read query gene set.
4: Encode list of target gene set maps
5: for (all queries) do
6: Encode query map of index and value.
7: Retrieve target map given target id.
8: Get AND value of pair-wise of query map and target map into a list.
9: Sort the list descending.
10: Get the first element of list
11: Look up the map with this element and store query list.
12: end for
13: Sort the query list.
14: Output top five of gene set combinations.

5.2 Mutual Information Content

Mutual information content is computed in terms of information gain (IG) using both the methods. Algorithm 3 illustrates SQL-based approach to compute information gain between each possible pairs of query sets and target sets. We first create conditional table using set of SQL queries and use this table to compute information gain. Each pair of sets and their information gain are stored in a list and top five pairs are extracted by sorting the list in descending order of their corresponding IGs. Information gain is also computed using bitwise approach for the same dataset.

As illustrated in algorithm 4, conditional table is generated using bitwise AND operation and the values are used in computation sequence of the algorithm for final result.

Figure 7 shows average running time for one pair of query set and target set for two approaches. It is clear that TP in conditional table is computed significantly faster in bitwise approach than SQL approach. Even with decoding overhead, it is observed that True Positive (TP) calculation in bitwise approach takes seven times less than that of SQL approach. IG time only includes the calculation time when having all elements in conditional table.
Algorithm 3 Generate_Top5_Difference_use_SQL
1: Initialize variables and database connection.
2: Read target gene set (target id and count).
3: Read query gene set (query id and count).
4: Encode list of target gene set maps
5: for (all queries) do
6: Read target gene set and match gene id and output the ordered list of gene set id and count.
7: Calculate the conditional table values and information gain, store into a list
8: Sort the list descending.
9: Get the first element of list
10: Look up the Map with this element and store query list.
11: end for
12: Sort the query list.
13: Output top five of gene set combinations.

Algorithm 4 Generate_Top5_Difference_use_Bitwise
1: Initialize variables and database connection.
2: Read target gene set (target id and count).
3: Read query gene set (query id and count).
4: Encode list of target gene set maps
5: for (all queries) do
6: Encode query map of index and value.
7: Retrieve target map given target id.
8: Get AND value of pair-wise of query map and target map into a list.
9: Calculate the conditional table values and information gain, store into a list.
10: Sort the list descending.
11: Get the first element of list.
12: Look up the map with this element and store query list.
13: end for
14: Sort the query list.
15: Output top five of gene set combinations.

SQL approach is directly dependent on the number of genes in each set and also number of sets that are being compared against. As shown in Figure 8, running time for SQL approach is increasing linearly with the number of target gene sets. On the other hand, performance of bitwise approach is very consistent and is not affected much by the number of target sets. For a small number of comparisons, bitwise method performance is comparable with SQL approach, however, it performs significantly better as the number of comparison increases.

We also experimented with the efficiency of coding approach by checking the time taken by the program for sets with various number of genes. Conventionally with SQL, bigger sets will make the process slower. However, the proposed bitwise approach is not much affected by the number of genes in the set because the encoded data structure size is more or less fixed. Figure 9 shows the speed comparison for entire information gain computation. When calculating information gain with SQL, average running time increases linearly to the number of target gene sets. However with the proposed bitwise method, it virtually unaffected by the number of target gene sets. Figure 9(a) is the average running time of the program of 30 query gene sets in combinations with 1000 target gene sets. Figure 9(b) shows the average running time of 4 types of bitwise queries (1 query, 10 queries, 20 queries and 30 queries) and 1 query using SQL in comparisons with 1000 target gene sets.

Fig. 7: Comparison of average running time for calculation of four elements in conditional table and information gain for a combination of target gene set and query gene set.

Fig. 8: Running time of 1 query, 2 query and 3 query gene sets in comparisons with 1000 target gene sets.

Fig. 9: Running time of 30 query gene sets in comparisons with 1000 target gene sets. Each comparison includes a query gene set and a target gene set.
5.3 Time Complexity

In this section, we analyze time complexity of both approaches. The running time analysis for a single interaction between a query gene set and a target gene set for our bitwise coding approach can be explained as follows. Let \( n_q \) be the number of genes in query set, \( B_q \) be the number of blocks in this set and \( n_t \) be the number of blocks in the target gene set. The total time can be calculated by aggregating running times to encode query set, compare query gene set and target gene set, and decode the result (see Equation 4).

\[
C_{\text{bitwise}} = C_{\text{enc}} + C_{\text{comp}} + C_{\text{decode}} \quad (4)
\]

\( C_{\text{enc}} \) is the time to read all genes in a query set and encode into key and value format. The complexity is equal to the input size when reading all the indexes into an array. Therefore, \( C_{\text{enc}} = O(n_q) \) since \( n_q \) is the number of genes in a query set. \( C_{\text{comp}} \), is the time taken to compare the blocks with indexes that are common in both sets. In order to do this, we only need to scan \( \min(B_q, B_t) \) number of blocks. We have \( C_{\text{comp}} = O(\min(B_q, B_t)) \). When \( B_t = B_q \), the computation time is linear with the function of \( B_t \) (or \( B_q \)). And finally, decoding time \( C_{\text{decode}} \) is the time to look up decoding table which has the same data structure of blocks, i.e. \( C_{\text{decode}} = O(\min(B_q, B_t)) \). Thus, average running time (same order as Equation 4) is presented below.

\[
C_{\text{bitwise}} = O(n_q) + O(\min(B_q, B_t)) + O(\min(B_q, B_t))
\]

(5)

For SQL approach, time complexity is the time to look up the database to retrieve the result (see Equation 6). Let \( n_t \) be the maximum number of genes in a target gene set.

\[
C_{\text{SQL}} = O(n_q) + O(\max(n_q \log(n_q), n_t \log(n_t)))
\]

(6)

In this approach, both sets have to be sorted and that takes time \( O(\max(n_q \log(n_q), n_t \log(n_t))) \) since the average sorting time of \( n \) indexes in an array is \( O(n \log(n)) \). When \( n_q = n_t \), the running time will be \( O(n_q) + O(n_q \log(n_q)) \approx O(n_q \log(n_q)) \). Now if we compare the time complexities of bitwise approach and SQL approach, we have \( C_{\text{bitwise}} \ll C_{\text{SQL}} \). Bitwise coding method which encodes the universe gene set and target gene sets offers a way to compare two gene sets with the minimum number of blocks (bytes). In this approach the missing blocks (e.g., block with all zero values.) can be totally ignored in the comparison, thus offering a greater memory space saving and reduction in bitwise operation.

6. Conclusion

Gene set comparison is one of the key essential operations that is universally needed in many high-throughput genomic data analyses. Examples include functional enrichment analyses, ChIP-Seq peak finding, gene-gene interaction analysis, and so on. One important observation in these gene set comparison practices is that the majority of the processes is finding and comparing memberships of gene identifiers (in hundreds) in a large number of gene sets (in millions). We presented a gene identifier encoding scheme which could store a very large number of gene sets entirely in memory. This coding scheme suggests the feasibility of storing the target gene sets in memory and using it over and over again for the analysis of the high quantity of arriving query sets. Such an arrangement can be very useful in supporting learning methods and data mining methods which need to test a large number of gene identifier combinations against a fixed albeit large number of target gene sets. Our case studies demonstrated our coding scheme could offer linear time performance proportional to the number of arriving input query sets.

7. Acknowledgment

Work by TH was funded by a grant from the Vietnam Education Foundation (VEF). Work by PJ and DGS was supported in part by NIH Grant R21AA023212. Work by SHH was supported in part by NIH Grant R01AR063702-01A1. The opinions, findings, and conclusions stated herein are solely of the authors and do not necessarily reflect the official view of VEF or NIH.

References