db4DNASeq: An Object-oriented DNA Database Model associated with Sequence Search Method

Keng Hoong Ng[a], Kyuk Wei Shoo[b], Wei Liam Diong[c], Somnuk Phon-Amnuaisuk[d], Chin Kuan Ho[e]

[a,b,c,d,e]*Faculty of Information Technology, Multimedia University, Jln Multimedia 63100 Cyberjaya, Selangor, Malaysia*

[a]*E-mail :* ***khng@mmu.edu.my***

[b]*E-mail :* ***shoo.kyuk.wei05@mmu.edu.my***

[c]*E-mail :* ***diong.wei.liam05@mmu.edu.my***

[d]*E-mail :* ***somnuk.amnuaisuk@mmu.edu.my***

[e]*E-mail : ckho@mmu.edu.my*

ABSTRACT

DNA database consists of many nucleotide sequences, it is not only supporting typical database queries, but it also needs to facilitate sequence search and alignment. In this paper, we present an object-oriented nucleotide database which is designed not only for the convenience of executing normal database operations such as insertion, modification or data querying in a fast manner, but it also supports a fast search method on database sequences with reasonable tradeoff between time and memory usage.

Keywords

Object Database, sequence search, database query, Db4o, hash table

1.0 INTRODUCTION

Nucleotide sequence databases have become major resources for many researchers in biological science. These databases have frequently changed and expanded due to the advance technologies used in sequencing and also the rapid development of molecular biology nowadays. These biological data tend to be voluminous, hence the development of efficient and scalable solutions are desired to provide easy-to-use and interactive interfaces for accessing these data.

Molecular biologist started keeping DNA and protein sequences in text files due to they are text strings in nature. The use of DBMSs (database management systems) to store sequences was motivated by the large volume of genetic data (Seibel & Lifschitz, 2001). Although the usage of database technology in this research area is not uncommon, but most of them use only limited functionalities in DBMS (Letovsky, 1999).

There are many molecular databases, such as Genbank Sequence Database, EMBL (European Molecular Biology Laboratory) Nucleotide Database and the Ensembl Sequence Database. These are not complete database systems but file systems with own storage, manipulation and access methods. Moreover, some data items in these databases are mislabeled, incomplete or faulty annotated, or are simply redundant with other existing entries. In fact, a lot of users still work with flat files downloaded from the above public repositories and sequence searching is done through programs like BLAST (Altschul et al., 1990), SSAHA (Ning et al., 2001), FASTA (Lipman & Pearson, 1988) and etc.

Biological databases can be designed using relational data model, and implemented in relational DBMS. MAGIC-SPP (Liang & Sun, 2006) is a DNA sequence processing package which uses Oracle 9i relational database to store the details of sequence data. Systems like EnsMart, Atlas and DBGET/ LinkDB provide data in a relational form. The data in these databases are stored as primitive data types in tables and there are relationships exist among the tables. Sequences and their annotated biological features can be accessed and retrieved using SQL (Shah et al., 2005).

DNA databases implemented using relational DBMS rely on data models based on relations and tuples. These models do not provide abstract data types or inheritance over class-subclass hierarchies. On the other hand, the sequence data is often very complex and some typical data types used in conventional flat file include nested records, list and sets. As a result, these data can not easily be mapped into a relational database (Davidson et al., 1997). Object-oriented model was adopted in sequence database due to its ability to model these complex data accurately.

The DDBJ (DNA Data Bank of Japan) is a DNA database system that was created using an object-oriented design approach (Okayama et al., 1998). The approach enables straightforward manipulation of sequences and their related data, and also quick updates of the DDBJ system. OPM (Object Protocol Model) is a semantic-based object-oriented model that provides specific constructs to support the modeling of scientific experiments (Chen & Markowitz, 1995). Another object-oriented system is AceDB (Durbin, 1994), organization of data is in classes. However, it does not support class hierarchies or inheritance.

There have been a lot of works on modeling of sequence data using object-oriented approach. However, most of the data models are designed for the purpose of accessing the properties of sequences but not optimized for sequence matching. Although third-party search tools such as BLAST or FASTA can be used for this purpose, but wrapper classes are required for adjusting parameters and parsing results. This will cause overheads to the system and affect the search performance in terms of speed.

In this paper we present a conceptual model on DNA sequences using object-oriented approach. The data model was implemented using db4o (www.db4o.com), which is an open-source object database that can perform faster than conventional databases. Import or update of data in the database can be done through a user interface. Furthermore, data search or sequence match can be performed on these persistent objects. The rest of the paper is organized as follows: Section 2 reviews the materials and methods. Section 3 presents our results and discussion. Section 4 concludes our paper.

2.0 MATERIALS AND METHODS

This section presents the information on sample databases and the selected object database for implementation, our database design in UML class diagram, the overview of the database architecture, and the search algorithm uses to perform matching in sequences.

2.1 DNA Sample Data

In the implementation of our program, we use several DNA databases downloaded from the Ensembl anonymous FTP site (www.ensembl.org). In this public repository, users can find a lot of species' databases and their formats can be in FASTA, GenBank, EMBL and etc. The downloaded DNA databases are *Drosophila Melanogaster*, *Aedes Aegypti* and *Cavia Porcellus*.

2.2 Db4o – The Selected Object Database

Db4o (Db4o, 2007) is an open source object database that enables JAVA and .NET developers to reduce development time, cost and achieve exceptional level of performance. We choose this open source database is due to its simplicity and power. Its API is fairly easy to understand and a complete database can be produced by a novice programmer within short period of time. Its simplicity can be shown by some examples of Db4o codes in Table 1. From the table, we can see that only one or two lines of codes are needed for creating/ opening a database, loading objects from a class and storing objects into the database.

Table 1: Examples of db4o codes

Create	ObjectContainer db = Db4o.openFile("C:\\DNA.odb");
Load	ObjectSet result = db.get(DNA.class);
Store	DNA dna1 = new DNA("AAACCCGGG", "test_description"); db.set(dna1);

Based on our own research, we find out that parsing of DNA sequences in Db4o (object database) are faster than MySQL, which is a relational database. A total of 21181 sequences from flat files are parsed to both databases, the Db4o takes about 2.3 seconds to complete the task while the latter spends about 3.6 seconds (Table 2). Db4o shows significant performance when it comes to storing or querying objects with complex data structures.

Table 2: Parsing speed of sequence database to db4o and MySQL

Db4o	MySQL
average 2330ms	average 3635ms

2.3 Database Design and Architecture

We use Unified Modeling Language (UML) to represent the data model (Larman, 2001). The class diagram (figure 1) shows the design of our database. Basically, there are three major classes in the database model, which are Sequence, Occurrence and SHashTable. Generalization in object-oriented model enables DNA sequences with different formats to be stored in a super class named Sequence.

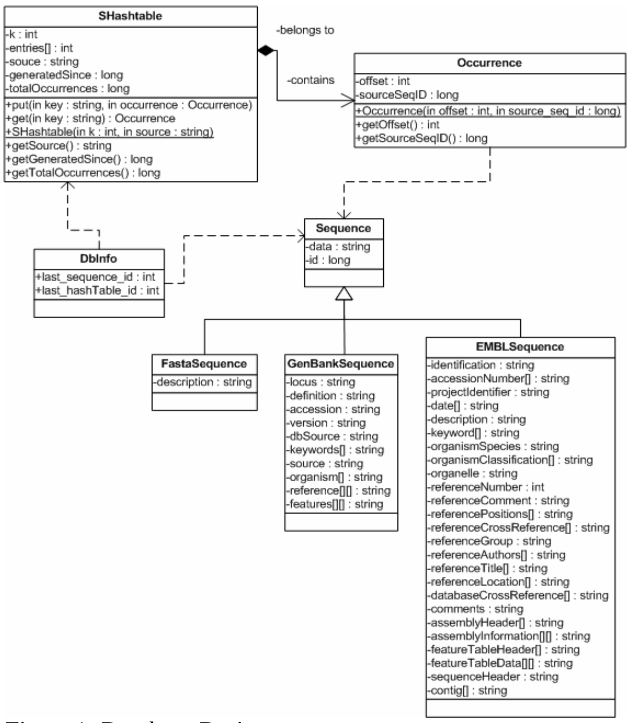

Figure 1: Database Design

The FastsSequence, GenBankSequence and EMBLSequence classes inherits all attributes and operations from the super class. Besides that, each sub-class can define its own attributes and operations. Sequence data will then be partitioned into many contiguous subsequences that are p bases long, where p is an integer. In this case, a DNA sequence with n bases long will contain ($n - p + 1$) overlapping subsequences. The occurrences of each subsequence will be stored into the SHashtable class. The DbInfo class is responsible for keeping track of update operations in SHashtable.

Figure 2 shows the architecture of the db4DNASeq, which is our proposed object-oriented DNA database. Firstly, sequence data from flat files will be parsed to the database. The parser is able to handle three types of commonly used data format and there are FASTA, Genbank and EMBL. Users can parse any new sequence database, delete or update the existing databases. The db4DNASeq database not only can be used for typical database queries, but it also allows user to carry out sequence matching between query sequences and the database. In this case, show species that belong to a family is an example of typical query in sequence database.

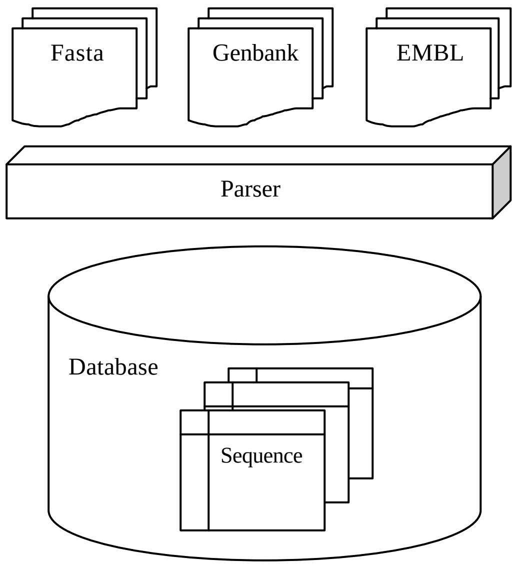

Figure 2: Database architecture

2.4 Search Algorithm for Database Sequences

A search algorithm has been integrated with the object database to perform matching between query sequences and the database. The search method will be based on the fast and reliable algorithm called SSAHA (Ning et al., 2001). In the first stage, all sequences in the database have to be divided into small subsequences with p bases long, which will be termed as p-tuples. Then, hash table needs to be constructed for storing all possible tuples. In our case, the p size is set to 8 by default.

Anyway, user can change the p value from minimum 2 to maximum 12. Since there are only four types of nucleotides (A, T, C and G) in DNA sequence, so there will be 4,194,304 (4^{11}) possible p-tuples if the p is set to 11. Each tuple will be transformed into a string of binary digits, and then it will be converted to an integer, as its storage index in the hash table. The occurrences of a particular p-tuple will be added to a list at the hashed index. For a given query sequence Q with length n, it is proceed base-by-base along Q from base 0 to base $n - p$. At base t, the list of positions of the occurrences of the p-tuple $w_t(Q)$ from the pre-computed hash table will be taken. Each position has two values, sequence index and offset.

From the list, a list of objects called 'Hit' is computed and a 'Hit' takes three values, sequence index, shift and offset. 'Shift' denotes the difference of position of occurrence between a particular p-tuple in hash table and current p-tuple of query sequence. The list of hits will then be sorted by its index and shift value. The final part of this search method is scanning through the sorted list for $2p - 1$ consecutive hits with identical index and shift values.

2.5 Implementation

In order to evaluate our proposed object database using Db4o for DNA sequences, an initial implementation is produced. It is done on a computer with 1.83 GHz processor and 2 GB of RAM. Prior to sequence search or data querying, an object database will be built. The downloaded flat-file based databases of the three fruit fly species will be preprocessed and then parsed to classes in object database. We implement three parsers to handle file formats such as FASTA, GenBank and EMBL.

SHashtable class is used to store hash table after it has been generated. Several hash tables with different *p*-tuple size can be built and saved as persistent objects in this class. The hash table works like a 2D array where the inner element is of type *ArrayList<Occurence>* and the outer layer is a fixed size array. The list of occurrences can be resizable in case there is any update in the future. Adding *n* elements into an instance of *ArrayList* requires O(n) time, and it does not affect much on the performance since hash table will be generated once only. We apply indexing mechanism on attributes in this class to speed up the retrieval process. As a result, the speed is much faster than the retrieval without indexing.

```
public class Occurrence{

    private final int sequence_index;
    private final int offset;

    public   Occurrence(int   seq_index,   int
    ofset){
        this.sequence_index=seq_index;
        this.offset=ofset;
    }
    public int getSequenceIndex(){
        return sequence_index;
    }
    public int getOffset(){
        return offset;
    }
}
```

Figure 3: Attributes and operations in Occurrence class

Occurrence class (figure 3) is created and it is responsible for storing the sequence ID and its offset position for a *p*-tuple. For instance, if the word ATGG appears many times in many sequences in the database, then the word will have a list of occurrences. *Sequence* class will act as a super class to classes such as *FastaSequence*, *GenBankSequence* and *EMBLSequence*. The sub-classes are required since each DNA sequence format has its own unique attributes. In addition, extra methods can be defined in each sub-class to facilitate the query process.

Several classes are also defined in the implementation but objects from these classes will not be stored as persistent objects in the database. The *Hit* class is for the purpose of storing matches temporarily and these matches are added into a list. After that, all matches in the list will be sorted according to its index, shift and offset. We use merge-sort rather than quick-sort for the sorting, it is because the worst case running time for the former is O(n log n), but it is O(n^2) for the latter (Goodrich et al., 2002). Next, finding of continuous hits with same index and shift value is carried out and these continuous hits will be regarded as a match between query sequence and the database sequence.

The *Result* class is used to display matching result. Each instance takes a fixed length of query sequence and a matched database sequence, and then the alignment between them can be shown. We enhance the sensitivity of this search method by expanding the matched sequence to its left and right in order to maximize its match. Figure 4 shows one of our implemented GUIs, where users can execute sequence search on this interface.

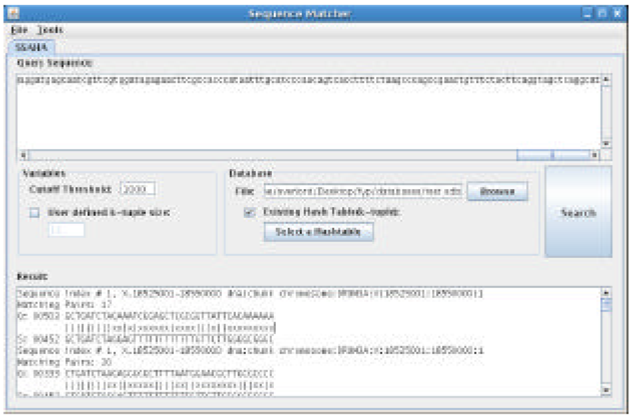

Figure 4: User interface for sequence search

3.0 RESULTS AND DISCUSSION

In this section, we show and then discuss the evaluation results of typical database queries and sequence searches that we have conducted in this object-oriented nucleotide database. For querying the database, three approaches can be used for any Db4o database and they are Query by Example (QBE), Native queries and Simple Object Data Access (SODA) queries. We test the database with some queries and the outcome is fast and reliable, where we execute a query and the retrieval can be done on the fly.

```
Query DNAquery=db.query();
DNAquery.constrain(EMBLSequence.class);
DNAquery.descend(“OrganismSpecies”).constra
in(“Drosophila”);
DNAquery.descend(“projectIdentifier”).const
rain(“Pn34451”);
ObjectSet result=DNAquery.execute();
listResult(result);
```

Figure 5: An example of SODA query

Figure 5 shows an example of query that retrieves sequence objects that belong to species *Drosophila* and its project identifier is *Pn34451*. The execution of the query is faster than the file-based method where in the latter method, we need to write codes to open a file, search all the data, retrieve the targeted data and then close the file. In addition, any updates to the database sequences can be done in just a few lines of codes and the speed is only within milliseconds. For instance, we only have to provide the following codes (figure 6) for updating the organism's classification to *Prokaryote* for sequence with ID *AC011569*. In object database, new methods can also be defined in order to facilitate the execution of a specific query. This is contrast to relational model where it uses available SQL functions for queries and sometimes a specific query is unable to be executed due to constraints in SQL.

```
List<EMBLSequence> Sequence=db.query(new
Predicate<EMBLSequence>(){
  public Boolean match(EMBLSequence Sq){
        return Sq.getSequenceID()
        .equals(”AC011569”);
  }});
EMBLSequence Seq=Sequence.next();
Seq.setOrgsmClass(“Prokaryote”);
db.set(Seq);
```

Figure 6: An update operation in db4DNASeq database

Since our object database (db4DNASeq) is also designed for the fast search method called SSAHA, hence we evaluate the search performance of the database with some query sequences. At first, the length of each query sequence is fixed at 100bp, and then the tests start with 10 randomly generated query sequences, follow by 50, 100, 200 and 300 sequences. These tests are performed on the generated hash tables with word size set to 6, 7, 8 and 9. Two main factors are focused in these tests and there are the time taken to load partial hash table from database to memory, and the memory consumption for each test.

Table 3: Elapsed time for loading partial hash table from database to memory

	Number of Query Sequences				
Word Size	10	50	100	200	300
6	0.30s	0.30s	0.32s	0.33s	0.33s
7	0.69s	0.67s	0.71s	0.76s	0.78s
8	2.01s	2.21s	2.38s	2.42s	2.02s
9	6.34s	5.94s	6.16s	6.13s	6.27s

The reason we concentrate on both factors but not the matching result in this evaluation is because the search algorithm we use is based on SSAHA, so there is no doubts on the outcome. The main objective is to evaluate the effectiveness in terms of speed and memory usage of the hash tables that are stored in the object database. This is something different from the original SSAHA program, where it uses large amount of memory and processing power to generate a hash table and store the complete hash table in the memory when we execute the program. In some cases, it might require a few gigabytes of memory just for the hash table (Ning et al., 2004).

Table 3 shows the elapsed time for loading a partial hash table from the database to memory for each test. The partial in this case means that only the matched *p*-tuples in hash table will be loaded into memory but not the whole hash table. Hence it will use less memory and it is preferred in computer without large RAM. From the result, it is clear that the word size will affect the elapsed time but the number of query sequences does not have impact on it. The probable explanation for this scenario is when the word size increase, the number of *p*-tuples in hash table also increase. For instance, 4096 (4^6) possible *p*-tuples when word size is 6, and it increases to 262144 (4^9) when the word size is set to 9. As a result, the same randomly generated query sequences will have more matches in hash table with word size 9 compares to hash table with word size 6, while the occurrences per specific word (*p*-tuple) will be higher in the latter case.

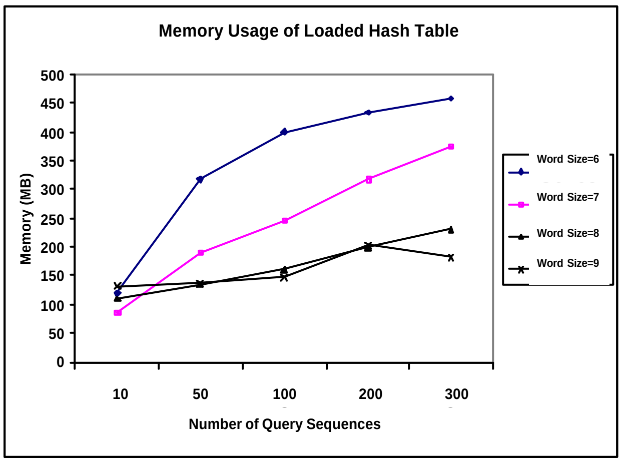

Figure 7: Memory usage of hash table

The above explanation is further supported by the memory usage of the loaded hash table (figure 7), where the result shows that memory consumption is higher in smaller word size hash table. It is due to the increment of occurrences in words, and occurrences are objects in the object database. In other word, we can say that the sensitivity of searching in hash table with smaller word size is better than hash table with bigger word size. We also observe that higher numbers of query sequences contribute to greater memory usage, and it is caused by the growing numbers of matches in hash table.

4.0 CONCLUSION

We have designed an object-oriented DNA database, and implemented the database model using one of the reliable open-source object DBMS, which is Db4o. The developed database is evaluated by a series of database operations such as insertion, updating and querying. The result shows that all these operations can be done and completed in a fast manner. The database is also tested with a fast search method, where it uses reasonable memory for the hash table and it is suitable for computer with lower RAM.

REFERENCES

Altschul, S., Gish, W., Miller, W., & Lipman, D. (1990). A basic local alignment search tool. *J. Mol. Biol., 215,* 403 – 410.

Chen, I.M., & Markowitz, V. (1995). An overview of the object protocol model (OPM) and the OPM data management tools. *Information Systems, 20(5),* 393 – 418.

Davidson, S.B., Overton, C., Tannen, V., & Wong, L. (1997). Biokleisli: A digital library for biomedical researchers. *International Journal on Digital Libraries, 1,* 36 – 53.

Db4o: http://www.db4o.com(accessed in Dec 2007).

Durbin, R., & Mieg, J.T. (1994). The AceDB Genome Database. *Computation Methods in Genome Research,* 45 – 56.

Goodrich, M.T., & Tamassia, R. (2002). *Algorithm Design: Foundations, Analysis, and Internet Examples*: John Wiley & Sons, Inc.

Kasprzyk, A., Keefe, D., *et al*. (2004). EnsMart: A Generic System for Fast and Flexible Access to Biological Data. *Genomre Research, 14*, 160 – 169.

Larman, C. (2001). Applying UML and Pattems, 2nd edition, Prentice Hall.

Letovsky, S. (1999). *Bioinformatics: Database and Systems:* Kluwer.

Liang, C., Sun, F., *et al*. (2006). MAGIC-SPP: a database-driven DNA sequence processing package with associated management tools. *BMC Bioinformatics, 7,* 115 – 129.

Lipman, D.J., & Pearson, W.R. (1998). Improved tools for biological sequence comparison. *Proc. Natl Acad. Sci. USA, 85,* 2444 – 2448.

Ning, Z., Cox, A.J., & Mullikin, J.C. (2001). SSAHA: A Fast Search Method for Large DNA Databases. *Genome Research, 11*, 1725 – 1729.

Ning, Z., Spooner, W., *et al*. (2004). The SSAHA Trace Server. *Proceedings of the 2004 IEEE Computational Systems Bioinformatics Conference,* 544 – 545.

Okayama, T., Tamura, T., *et al*. (1998). Formal design and implementation of an improved DDBJ DNA database with a new schema and object-oriented library. *Bioinformatics, 14,* 472 – 478.

Rother, K., Steinke, T., *et al*. (2005). Columba: an integrated database of proteins, structures, and annotations. *BMC Bioinformatics, 6,* 81 – 91.

Seibel, L.F.B., & Lifschitz, S. (2001). A Genome Database Framework. *LNCS, 2113,* 319 – 329.

Shah, S.P., Huang, Y., *et al*. (2005). Atlas – a data warehouse for integrative bioinformatics. *BMC Bioinformatics, 6,* 34 – 49.

Weems, D., Miller, N., *et al*. (2004). Design, implementation and maintenance of model organism database for Arabidopsis thaliana. *Comp Funct Genom, 5,* 362 - 369.

Zhou, X., & Song, I.Y. (2005). Conceptual Modeling of Genetic Studies and Pharmacogenetics. *LNCS, 3482,* 402 – 415.