Making High-level Queries on Diverse Genome Data: A Structured Genome Document Database System Based on GXML and GQL

Aaron Stokes [1] stokes@ics.es.osaka-u.ac.jp
Hideo Matsuda [2] matsuda@ics.es.osaka-u.ac.jp
Akihiro Hashimoto [2] hasimoto@ics.es.osaka-u.ac.jp

[1] CREST, JST (Japan Science and Technology)
[2] Department of Informatics and Mathematical Science, Graduate School of Engineering Science, Osaka University, 1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan

Abstract

Complete DNA sequences (genomes) and associated data are being made available worldwide at an astonishing rate. Through computer analysis of such data, molecular biologists hope to gain an overall understanding of the genome, such as by predicting large-scale gene networks. However, this is difficult because diverse genome data are scattered across many highly heterogeneous databases, and because existing database systems lack the facilities to expose and analyze functional relationships among the data. To address these problems, we propose a new type of genome database system. Since a genome can be thought of intuitively as a kind of 'document', our system uses a structured document language based on XML to effectively represent genomes and associated data. The information-rich structures of the genome documents help cope with data diversity and heterogeneity. A powerful query language is introduced that exposes important biological relationships among the genome data. We have obtained favorable results from several experiments, demonstrating the usefulness of our method in building a top-down view of genome functionality.

1 Introduction

With recent advances in biotechnology, tremendous amounts of sequence, gene, protein, pathway, and other genome data are being accumulated available at an exponentially-increasing rate. By using computer analyses to identify correspondences and correlations among such data, researchers hope to elucidate the functions of putative genes, infer networks of gene interactions, and generally gain a more complete understanding of the genome.

However, it is not trivial to perform computer analyses on genome data because (a) genome databases are complex, highly heterogeneous, geographically dispersed, and constantly evolving; (b) interfaces to databases provide only limited functionality, such as retrieval by key word or by similarity search; and (c) data are often described in human-readable formats (such as free-text) that are difficult for computers to analyze.

Several methods have been proposed to address these problems [10]. These methods are classified into *integration* and *transformation* approaches. In the integration approach, all the genome data are first converted into some consistent form based on a global schema incorporating a specific data model (mainly, an object-oriented model), and then queries are performed on the integrated database. Typical examples are IGD [14] and ACeDB [17].

On the other hand, the transformation approach incorporates a powerful query language (such as BioKleisli [8]), which can perform arbitrary transformations among heterogeneous data sources. Researchers can use the query language to transform data to appropriate formats and then issue queries on the transformed data.

However, neither of these approaches is entirely suitable for use in hunting out correspondences and correlations among genome data, for the following reasons.

Mycoplasma genitalium

```
...
(ORF with functional annotation)
gene          109262..109675
              /gene="MG081"
CDS           109262..109675
              /gene="MG081"
              /note="similar to GB:U00089 SP:P75550 ...
              /codon_start=1
              /transl_table=4
              /product="ribosomal protein L11 (rpl11)"
...
(ORF without functional annotation)
gene          158022..158246
              /gene="MG131"
CDS           158022..158246
              /gene="MG131"
              /note="hypothetical protein; ...
              /codon_start=1
              /transl_table=4
              /product="M. genitalium predicted ..."
...
```

Aquifex aeolicus

```
...
(ORF with functional annotation)
gene          3665..4390
              /note="aq_009"
              /gene="rplC"
CDS           3665..4390
              /gene="rplC"
              /codon_start=1
              /product="ribosomal protein L03"
...
(ORF without functional annotation)
gene          complement(9657..10157)
              /gene="aq_022"
CDS           complement(9657..10157)
              /gene="aq_022"
              /codon_start=1
              /product="putative protein"
...
```

Figure 1: In these GenBank Release 110 entries for two different organisms, the strategies used for storing ORF ID (bold type) and gene name (underlined) information are inconsistent.

- In the transformation approach, users need to know some details about the original data formats to be transformed, and then specify how to perform the transformation for each data source. Not only is this tedious, but it may also be extremely difficult when there are inconsistencies within the same data format (see Fig. 1).
- Due to the rapid progress of biotechnology, not only data but also the relationships among the data are updated frequently. Integration methods encounter problems of schema evolution [1] due to their use of global schemas, and in transformation methods the queries themselves may need to be modified regularly.
- Even if genome projects submit their data to the same database, some degree of heterogeneity remains in the data representation due to the differences in their original schemas. Therefore, transformation of data is not trivial in many cases.

In our research, we have developed a different approach. Our focus is not on the *integration* of genome data (although this is a necessary step, of course), but rather on the effective *representation* of data.

Since an entire genome (here we refer to a bacterial chromosome) can be thought of as a kind of 'document' (with genes as words, functionally-related genes as sentences, etc.), our method represents genomes and associated information as self-descriptive structured documents, using a language called GXML [15]. This representation is more effective than traditional models because it expresses genomic relationships more directly and it makes data more readily accessible. Moreover, the self-descriptiveness of the documents helps cope with diversity and heterogeneity among the data. We have also introduced a powerful query language, called GQL, that acts on multiple GXML documents to readily expose important biological relationships within and between genomes [15, 16].

Our ultimate goal is to make it easy to perform complex, top-down biological analyses that clarify genome mechanisms within the cell. In this paper, we take our next step by proposing a complete genome database system based on GXML and GQL. In the system, GXML genome documents are stored, indexed, and efficiently queried via GQL queries.

The paper is organized as follows. In the following section, we use an interesting example to illustrate the unique querying capabilities our method provides. We then give a more detailed description of GXML and GQL in Section 3. In Section 4, we go on to describe the proposed GXML/GQL database system and some techniques we consider for its efficient implementation. Section 5 presents some results we obtained using a prototype system. We conclude the paper with some remarks.

```
WHERE
  <pw>
    <pwname>$pwname</>
  </> AS $pw IN "ecoli.gxml"
ORDER $pw BY $pwname
CONSTRUCT
  <neighborsbypathway>
    $pw
    WHERE
      <feature><alias>$a1</></> AS $f1
      <feature><alias>$a2</></> AS $f2
        IN "ecoli.gxml",
      neighbors($f1,$f2),
      upstream($f1,$f2),
      pwneighbors($f1,$f2,$pw)
    ORDER $f1, $f2 BY $a1, $a2
    CONSTRUCT
      <neighborpair>
        $f1
        $f2
      </>
  </>
```

```
<neighborsbypathway>
  <pw>...</pw>
  <neighborpair>
    <feature>...</feature>
    <feature>...</feature>
  </neighborpair>
  ...
  <neighborpair>
    <feature>...</feature>
    <feature>...</feature>
  </neighborpair>
</neighborsbypathway>
...
<neighborsbypathway>
  ...
</neighborsbypathway>
```

Figure 2: Example GQL query (left) and an outline of its results (right).

2 Queries on Genome Data

Let us now consider an example. It has long been hypothesized that there may be some kind of correspondence between gene transcription order and functional relationships of the encoded proteins. Dandekar *et al.* showed that some highly conserved gene pairs do indeed interact physically on the same metabolic pathway [7]. This kind of observation can be very useful in the following ways.

1. If the function of the product of one gene in a conserved gene pair is known, conservation of gene order may hint at the function of the product of the neighboring gene.
2. If the products of conserved gene pairs generally have basic cellular functions, this may shed some light on how the fundamental "building blocks" of the cell evolved.
3. Conserved gene transcription order hints at some kind of interdependence of the folding of proteins (co-translational folding).

Unfortunately, this correspondence is not a trivial one to identify using existing genome databases and tools. First of all, the analysis requires sequence, protein function, and reaction pathway data that are spread across several databases such as GenBank [4], SWISS-PROT [3], and KEGG [13]. Second, we must have some mechanism for picking up neighboring genes one pair at a time on each genome. Third, we have to run sequence similarity or functional search programs to select gene pairs that are highly conserved between genomes. Finally, we need to combine the results and interpret them. In short, many hundreds of thousands of database accesses, searches, and operations must be smoothly and efficiently coordinated – a daunting task for even the most computer-literate biologist!

The genome database system that we propose can readily cope with this kind of complex analysis. More concretely, say we want to show that some genes that are neighbors on a genome also encode for proteins that are consecutive components of a pathway: "Find all pairs of neighbor genes g_1 and g_2 on the *Escherichia coli* genome and a pathway *pw*, where g_1 and g_2 encode enzymes that are consecutive components of *pw*."

The query can be expressed in our query language, GQL, as shown at left in Fig. 2. The result of this succinct query is a list of pairs of neighboring genes that encode proteins that physically interact on the same pathway, grouped by pathway, as indicated in the right of the figure. In an actual experiment, we were able to confirm the results given in [7] (see Section 5). The ability to expose such tendencies among highly complex networks of pathways is one of the aims of our method.

3 Fundamentals

3.1 GXML: Genome-oriented eXtensible Markup Language

Genomes and the genetic information they contain have a generally hierarchical structure: DNA consists of two strands, each of which contains features, which in turn have an associated type and function, and so on. Furthermore, some biological reactions occurring within the genome also tend to be hierarchically organized. For example, the intermediary sugar metabolism contains a glycolysis pathway, which in turn includes an Embden-Meyerhof pathway [2, 11]. Since a hierarchical data model can closely represent such structures of the genome, describing genomes as hierarchical structured documents should assist in determining correspondences between genes and other genomic entities.

Several structured document languages have been introduced, including ASN.1, SGML, and XML (eXtensible Markup Language). Although ASN.1 and SGML are accepted and powerful methods of representing hierarchical data, they assume a fixed schema and require the definition of strict inter-object relationships. In contrast, XML has a large degree of flexibility in its structure; e.g., new types of data can be introduced without affecting the reachability of existing data, and without requiring reconstruction of the dataset. In the case of genome data, rapid progress in the field causes frequent changes in the relationships between biological entities. Therefore, XML seems the best candidate for forming the basis of a structured genome document language.

There are some genomic data structures that cannot be represented directly in structured documents. These include circular genomes (documents cannot be physically wrapped), overlapping genes (all data elements must nest correctly), and complementary strands (we need to associate genes across opposite strands, while also being able to discriminate by strand). To overcome these obstacles, we designed an XML-based language called GXML (Genome-oriented eXtensible Markup Language).

An example of a GXML document is shown in Fig. 3. Data is "marked up" under several major elements. `genome` is the root element for a single genome. `contig` contains the nucleotide sequence for a contiguous region of the genome. `feature` contains information on a gene, such as gene name, location, DNA sequence, amino-acid sequence for the encoded protein, and function. `pw` contains information on a specific pathway, such as a list of enzymes that constitute the pathway. `role` contains information on a specific enzyme, such as a list of genes that can encode the enzyme, and the substrates (reactants) and products of a reaction involving the enzyme.

As can be seen, the sequence, protein, function, and pathway information that constitute a genome are represented in a logical and self-descriptive package that is highly machine-readable.

3.2 GQL: Genome-oriented Query Language

Several methods have been proposed for querying XML documents. Some of the methods involve storing the documents in traditional object-oriented or relational database management systems, and using familiar query languages to execute queries on the data. However, in our case this is considered impractical for the following reasons.

- GXML documents are self-descriptive and their structures contain a great deal of useful information. We need to be able to query documents not only on content but also on structure. This is difficult in traditional database systems, since schemas are expected to be rigid and the query languages do not provide for, for example, pattern matching of element hierarchies.
- A major strength of XML is its flexibility to schema evolution. Since schema changes in genome data are particularly frequent, it defeats the purpose of using XML if we have to translate the data into stricter relational or object-oriented data formats.

Thus it is preferable to query GXML documents directly, using a query language that is appropriate to the underlying semi-structured data model. Since GXML data can be thought to represent graph-like networks of relationships between genomic entities, queries on GXML data should be capable of

```
<?xml version="1.0" ?>
<!DOCTYPE gxml SYSTEM "gxml.dtd" >

<gxml>
   <genome>
      <gid>Escherichia coli K-12...</gid>
      <whose>E. coli Genome Project</whose>
      <date>98Nov18</date>
      <contig>
         <cid>c000</cid>
         <dna>ATGCGAGTGTTGAAGTTCGGCGG...</dna>
      </contig>
      ...
      <feature type="orf">
         <fid>b1263</fid>
         <alias>trpD</alias>
         <location>
            <cid>c000</cid>
            <start>1317813</start>
            <end>1319408</end>
            <strand>-</strand>
         </location>
         <dna>ATGGCTGACATTCTGC...</dna>
         <prot>MADILLLDNIDSF...</prot>
      </feature>
      ...
      <pw>
         <pid>00401</pid>
         <pwname>tryptophan biosynthesis</pwname>
         <pwrole>
           <rid>4.1.3.27</rid>
           <fid>b1263</fid>
         </pwrole>
         ...
      </pw>
      ...
      <role>
         <rid>4.1.3.27</rid>
         <rdescription> ... </rdescription>
         <fid>b1263</fid>
         ...
         <substrate>Pyrophosphate</substrate>
         <product>Anthranilate</product>
         ...
      </role>
   </genome>
</gxml>
```

Figure 3: Example of a GXML structured genome document.

deriving other networks of relationships. An existing query language, XML-QL [9], is capable of such graph reconstruction, has a simple syntax, and features pattern-matching to take advantage of the self-descriptiveness of documents. Therefore, we chose XML-QL as the basis for our query language.

To make the relationships represented in GXML documents readily accessible to use as building blocks for higher level queries, we extended XML-QL with powerful functions that constitute a "view" of several important biological relationships (see Fig. 4). We also augmented the language with aggregate functionality that we considered necessary but that was not defined in the XML-QL specification [16]. The resultant query language is called GQL (Genome-oriented Query Language).

4 A Database System based on GXML and GQL

4.1 Overview

To implement GXML and GQL, we consider the architecture shown in Fig. 5. The system consists of four major components, briefly described below.

- **Translator**
  Since GXML files are not yet widely available, it is necessary to generate them by translating and merging data from existing genome databases such as GenBank, SWISS-PROT, and KEGG.
- **GXML Database**

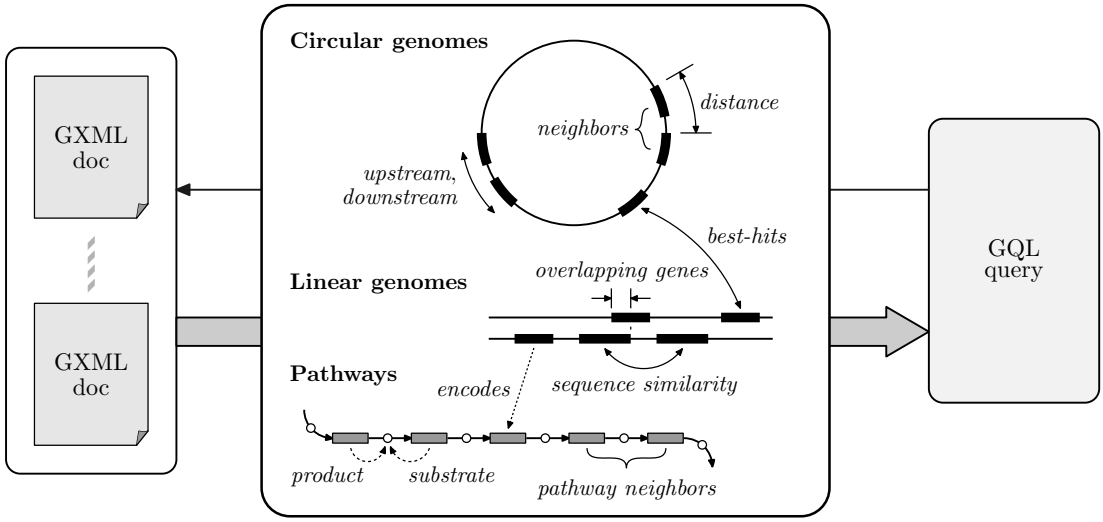

Figure 4: Genomic 'view' provided by GQL.

Since GQL queries act directly on GXML documents, we store the documents in a self-managed database, and use indices strategically to shorten the time required for data retrieval.

- **GQL Processor**
  The GQL processor is responsible for parsing an incoming query, determining an efficient plan of execution (not currently implemented; see Section 6), and then processing the query by executing a series of nested loops in which variables are bound and tested by pattern matching or external function calls. The plan of execution specifies the nesting of the loops and the indices used.

- **External Function Wrappers**
  This component consists of "wrappers" for coordinating execution of and communication with external functions and programs such as FASTA.

4.2 Indices

Since GXML documents are generally very large, it is inefficient to parse them for each data seek operation. Moreover, since GQL uses pattern-matching to query the document structure, a mechanism is needed for pinpointing elements at specific locations in the hierarchy without a complete traversal. We solve these problems by creating several kinds of indices for each GXML document stored.

Tree-structured index

This index consists of a number of fixed-length records linked together by pointers into a tree-like structure. Each record represents a single element within the document, and stores pointers to the element's parent, first child, and next sibling, as well as the physical locations of the element and its contents within the GXML document. Using this index, it is easy to determine containment relationships or access element content.

DNA position index

It is often necessary to identify genes based on their positional relationships on the genome. For example, we may want to find all pairs of neighboring genes. However, this is a difficult task because (a) residue-based distances may have to be calculated for several millions of gene pairs, and (b) the positional relationships sought may have been lost when circular genomes were 'split' for representation as documents. To solve these difficulties, we have developed a DNA position index, in which the genome sequence is laid out twice and spliced into a doubly-linked structure. Using the index, it is easy to locate neighboring genes on both linear and circular genomes.

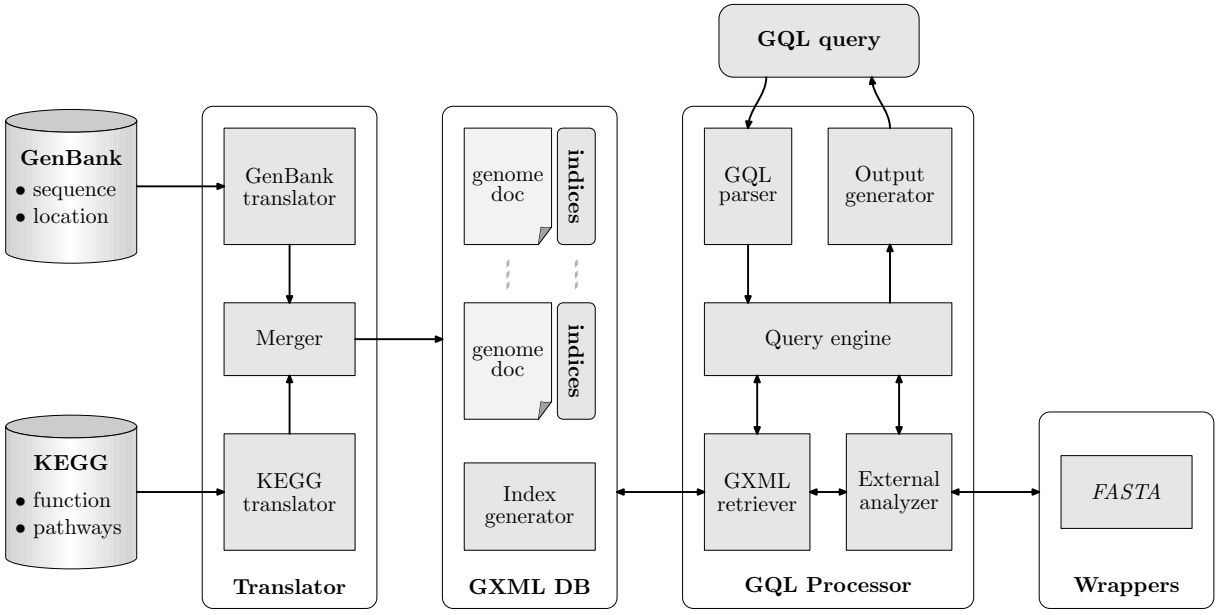

Figure 5: Architecture of the GXML/GQL database system.

We also introduce several kinds of supplementary hash indices for fast access to specific kinds of elements; e.g., gene name index, element name index, pathway name index, etc.

5 Experimental Results

To test the viability of our method, we have implemented a prototype system based on the proposed architecture. The translator component was created using the Perl language, while all other components were created using the C++ language. In the prototype system, an efficient plan of execution is specified by the user together with the query. The platform used for execution was a Gateway GP6-400 (Intel Pentium-II 400MHz processor).

We created GXML genome documents for two complete genomes, *Escherichia coli* and *Bacillus subtilis*, using data extracted from GenBank Release 110 and a February 21, 1999 download of the KEGG database. We then executed several queries, each of which is described below. We make particular note of the biological significance of the results obtained.

5.1 Transcription order vs. physical interactivity (revisited)

We first executed the query described in Section 2. Recall that the aim is to determine, per pathway, pairs of gene neighbors that have physical interactions on the pathway. We consider in particular the *tryptophan biosynthesis* pathway, for which the following gene pairs were isolated: (*trpC*,*trpA*), (*trpC*,*trpB*), (*trpC*,*trpC*), (*trpD*,*trpC*), (*trpD*,*trpD*), and (*trpE*,*trpD*).

These results reveal some interesting facts. Firstly, the pair (*trpC*,*trpC*) hints that *trpC* is multi-functional. The same can be said for *trpD*. Secondly, the fact that *trpA* and *trpB* are both seen to interact with *trpC* hints that they may be sub-unit genes, working together to encode a single protein.

A graphical interpretation of the results is shown in Fig. 6. A search of the biological literature [7] confirmed that each of the pairs we found are indeed consecutive members of the tryptophan biosynthesis pathway, and that the relationships shown in Fig. 6 are correct. It is also intriguing to note that the tryptophan biosynthesis pathway is one example of a pathway where gene order is conserved significantly between genomes. In future tests, we could extend our query to include multiple genomes in an attempt to expose this relationship.

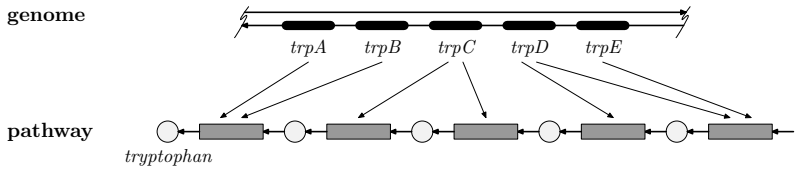

Figure 6: Results showing gene neighbors whose products interact physically (see Section 5.1).

```
WHERE
  <feature><alias>"sdhA"</></> AS $sdhA
  <feature><alias>"sdhB"</></> AS $sdhB
    IN "bsub.gxml",
  <feature></> AS $g1
  <feature></> AS $g2 IN "ecoli.gxml",
  neighbors($g1,$g2),
  bidirbesthit($sdhA,$bsub,$g1,$ecoli),
  bidirbesthit($sdhB,$bsub,$g2,$ecoli)
CONSTRUCT
  <pair>
    $g1
    $g2
  </>
```

```
WHERE
  <feature><alias>"phoP"</></> AS $phoP
  <feature><alias>"phoQ"</></> AS $phoQ
  <feature></> AS $p1
  <feature></> AS $p2 IN "ecoli.gxml",
  $p1 != $phoP, $p1 != $phoQ,
  $p2 != $phoP, $p2 != $phoQ,
  similarity($phoP,$p1,$s1),
  similarity($phoQ,$p2,$s2),
  $s1 >= 200, $s2 >= 200,
  neighbors($p1,$p2)
CONSTRUCT
  <pair>
    $p1
    $p2
  </>
```

Figure 7: GQL representation of queries described in Sections 5.2 (left) and 5.3 (right).

5.2 Identifying Homologous Gene Pairs across Genomes

Let us consider another example. Say we want to use similarity searches to find neighboring gene pairs that are homologous across genomes. This is particularly tedious because we must (a) consider every pair of genes on each genome, (b) exclude pairs of non-neighboring genes, and (c) verify homologous relationships between genes of one pair on one genome and genes of another pair on another genome. With GQL, it becomes possible to automate the process. In natural language, we might say: "Given two neighbor genes *sdhA* and *sdhB* on the *Bacillus subtilis* genome, retrieve all neighbor genes g_1 and g_2 on the *Escherichia coli* genome where g_1 and g_2 are orthologous to *sdhA* and *sdhB*, respectively." In GQL, this is expressed as shown at left in Fig. 7.

The result of the query was the single gene pair (*frdA*,*frdB*). The fact that we achieved a single, exact match across the genomes strongly suggests that the pairs (*sdhA*,*sdhB*) and (*frdA*,*frdB*) may be evolutionarily related, and therefore may exhibit common functionality. Indeed, we could confirm in the literature [6] that both pairs of genes participate in the citric acid (TCA) cycles of their respective genomes. Hence they play important roles in the energy metabolism in many organisms.

5.3 Exploring Functional Relationships on a Single Genome

Our final query example demonstrates how easily GQL can be used to explore functionally related genes on a single genome: "Given two neighbor genes *phoP* and *phoQ* on the *Escherichia coli* genome, retrieve all pairs of two neighbor genes p_1 and p_2 on the same genome where p_1 and p_2 are paralogous to *phoP* and *phoQ*, respectively." Biologically speaking, this query should be useful in exposing gene pairs that may have been derived from a common ancestor pair through duplication, and hence may share common functional interactions. The GQL form of the query is shown at right in Fig. 7.

Ten pairs of gene neighbors were isolated by this query: (*phoB*,*phoR*), (*baeR*,*baeS*), (*f227*,*f480*), (*ompR*,*envZ*), (*kdpE*,*kdpD*), (*cpxR*,*cpxA*), (*rstA*,*rstB*), (*basR*,*basS*), (*f239*,*f452*), and (*creB*,*creC*). Since these pairs are all paralogous to (*phoP*,*phoQ*), they may exhibit similar functionality to *phoP* and *phoQ*. This was indeed found to be the case. It is reported that the genes in all 11 pairs function together as cognate sensor/regulator pairs [12]. This result is illustrated graphically in Fig. 8.

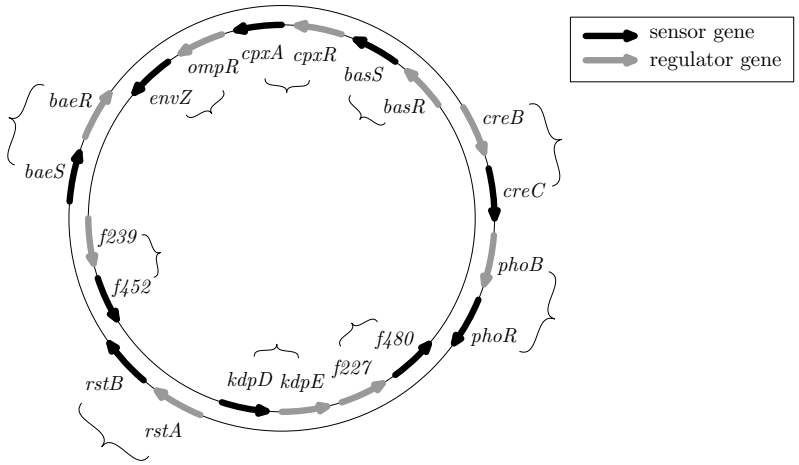

Figure 8: Pairs of neighboring genes with common functionality (see Section 5.3).

5.4 System Performance

The CPU time required to create and index the two GXML genome documents was 10.2 seconds. The document sizes were 11,566,197 bytes and 10,391,654 bytes for the *Escherichia coli* and *Bacillus subtilis* documents, respectively. Since the sets of complete genome data contained in GenBank and KEGG are updated relatively infrequently (no more than once daily), we can conclude that these translation times are highly acceptable. Execution times for the three queries were 12.8 seconds, 14.5 seconds, and 11.7 seconds, respectively. These are very reasonable results.

6 Remarks

We accomplished several things in the preceding sections. First, we found an existing, accepted technology – XML – that we could adapt to effectively represent huge quantities of diverse, ever-evolving genome data. Second, we extended XML-QL to define a query language with biologically meaningful functionality and constructs deemed essential from a data engineering viewpoint. Third, we proposed a method for constructing a complete structured document genome database system. Finally, we were able to confirm using a prototype that our method has the potential to be both powerful and practical. However, we have deliberately avoided the following sticky issues.

Translation into GXML

Since the data stored in KEGG and GenBank are updated infrequently in relation to the time required for translation into GXML, it might seem that the process could easily be automated. However, this is difficult for some of the same reasons that motivates us to create GXML: (1) complex database schemas and inconsistencies, (2) schema evolution: recognizing it and adapting to it, (3) heterogeneity between data sources, and (4) the lack of a version-controlled mechanism for associating data across multiple data sources.

Query Optimization

In the current implementation, GQL queries are optimized by the user by specifying the plan of execution. Ideally, the query processor should be able to automatically generate efficient plans of execution. Since the GXML DTD constrains the structure of the documents, and since a rough idea of the size of data sets can be gained from parameters such as genome size, it should be possible to select plans using cost-based heuristics.

We conclude that with further development in these areas, a GXML/GQL genome database should form an invaluable tool for helping researchers identify complex genome interactions on many levels.

Acknowledgements

This work was supported in part by CREST of JST (Japan Science and Technology), and a Grant-in-Aid "Genome Science" (08283103) for Scientific Research on Priority Areas from the Ministry of Education, Science, Sports and Culture in Japan. We would also like to thank Dr. Ross Overbeek of the WIT Project for his valuable suggestions and assistance in this research.

References

[1] Abiteboul, S., Querying semi-structured data, *Proceedings of the International Conference on Database Theory*, 1–18, 1997.

[2] Alberts, B. *et al.*, *Molecular Biology of The Cell*, Garland Publishing, New York, 3rd edition, 1994.

[3] Bairoch, A. and Apweiler, R., The SWISS-PROT protein sequence data bank and its supplement TrEMBL in 1999, *Nucleic Acids Research*, 27(1):49–54, 1999.

[4] Benson, D.A. *et al.*, GenBank, *Nucleic Acids Research*, 27(1):12–17, 1999.

[5] Bray, T., Paoli, J., and Sperberg-McQueen, C.M. (eds.), Extensible Markup Language (XML) 1.0, *W3C Recommendation 10-Feb-98*, available at `http://www.w3.org/TR/REC-xml`, 1998.

[6] Cronan, J.E.Jr. and Laporte, D., Tricarboxylic Acid Cycle and Glyoxylate Bypass, *Escherichia coli and Salmonella: Cellular and Molecular Biology*, ed. Neidhardt, F. C. *et al.*, ASM Press, Washington D.C., Chap. 16, 206–216, 1996.

[7] Dandekar, T., Snel, B. *et al.*, Conservation of gene order: a fingerprint of proteins that physically interact, *Trends in Biochemical Science*, 23:324–328, 1998.

[8] Davidson, S. B., Overton, C., Tannen, V., and Wong, L., BioKleisli: a digital library for biomedical researchers, *J. Digital Libraries*, 1(1):36–53, 1997.

[9] Deutsch, A., Fernandez, M., Florescu, D., Levy, A., and Suciu, D.: XML-QL: A Query Language for XML, *W3C Submission 19-August-1998*, available at `http://www.w3.org/TR/NOTE-xml-ql`, 1998.

[10] Davidson, S.B., Overton, C., and Buneman, P., Challenges in integrating biological data sources, *J. Computational Biology*, 2(4):557–572, 1995.

[11] Fraenkel, D.A., Glycolysis, *Escherichia coli and Salmonella: Cellular and Molecular Biology*, ed. Neidhardt, F. C. *et al.*, ASM Press, Washington D.C., Chap. 14, 189–198, 1996.

[12] Mizuno, T., Compilation of all genes encoding two-component phosphotransfer signal transducers in the genome of *Escherichia coli*, *DNA Research*, 4:161–168, 1997.

[13] Ogata, H., Goto, S., Sato, K. *et al.*, KEGG: kyoto encyclopedia of genes and genomes, *Nucleic Acids Research*, 27(1):29–34, 1999.

[14] Ritter, O., The Integrated Genomic Database, *Computational Methods in Genome Research* (Suhai, S. (ed.)), Plenum Press, New York, 57–73, 1994.

[15] Stokes, A.J., Matsuda, H., and Hashimoto, A., GXML: a novel method for exchanging and querying complete genomes by representing them as structured documents, *IPSJ Transactions on Database*, 40(3):66–78, 1999.

[16] Stokes, A.J., Matsuda, H., and Hashimoto, A., Introduction of aggregate functions to a language for querying structured genome documents, *IPSJ SIG Notes*, 99-DBS-61:237–242, 1999.

[17] Thierry-Mieg, J. and Durblin R., ACeDB – A *C. elegans* Database: Syntactic Definitions for the ACEDB Data Base Manager, 1992.