Regular Paper

GXML: A Novel Method for Exchanging and Querying Complete Genomes by Representing them as Structured Documents

AARON J. STOKES,† HIDEO MATSUDA†† and AKIHIRO HASHIMOTO††

Complete DNA sequences (complete genomes) for an increasing number of organisms are becoming available each year for use in biological research. However, genome project groups incorporate their own formats (or schemas) for representing the genome data accumulated by the projects. Such heterogeneity of their schemas prevents researchers from exchanging and comparing their data across genomes. In this paper, we present a new method for exchanging and querying information on complete genomes. Since genomes and the genetic information encoded on them have a hierarchical structure, they can be represented as a kind of structured document. We propose a document language called GXML for representing complete genomes. The document language, based on XML, can be used to exchange many kinds of genomic data, and offers a high degree of extensibility. We also define a query language called GQL to operate on the genome documents. Using this language, one can easily associate genes among different genomes and perform other biological analyses. We developed a prototype system based on the language. Using the system, we executed several test queries. The results were consistent with those published in biological literature. The processor and memory requirements of the prototype system were acceptable.

1. Introduction

Recent advances in biotechnology make it possible to determine the whole DNA nucleotide sequence (the complete genome) for an organism and to predict the entire set of putative genes on the genome. However, the functional roles of many predicted genes have not yet been revealed. In the area of Comparative Genomics, researchers attempt to determine the location and functionality of, and the interaction between, putative genes on complete genomes based on sequence comparison between genomes and on previously-established results.

Genome project groups make their data available to researchers directly via the Internet (by WWW, FTP, etc.), or through some curated biology databases. However, each project incorporates its own format (or schema) for representing the genome data accumulated by the project. Even if 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.

Moreover, most data stored in biology databases are not highly structured; in fact, much information (e.g., regarding gene functionality) is described in a free-text format. Thus, it is difficult for researchers to make comparisons between genomes based on such information.

Several methods have been proposed for querying various types of genome data[9]. 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[25] and ACeDB[21]. However, due to the rapid progress of biotechnology, not only data but also the relationships among the data are updated frequently. Thus some structured data model is required that exhibits significant flexibility to schema updates.

On the other hand, the transformation approach incorporates a powerful query language (such as BioKleisli[10]), 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, in this approach, users need to know some details about the original data formats to be transformed. As described in Sec-

† CREST, JST (Japan Science and Technology)
†† Graduate School of Engineering Science, Osaka University

tion 2.3, several inconsistencies in data representation exist, even within the same biology database. Thus it is troublesome to specify how to perform the transformation for each data source.

We focus not on *integration* (although this is a necessary step, of course), but on *representation*. We define a portable, plain text document format for representing complete genomes, that can be presented to humans with minimal processing and is also highly machine-readable. We then define a query language that acts on the genome documents to expose important biological relationships. Using our method, researchers can readily combine, analyze, and exchange genome information gathered from multiple sources.

Our method is currently in active use by the Japan *E. coli* genome DNA sequencing project[1),12),15),27)] and the WIT Project[22)] in the United States to exchange project-related data.

The rest of this paper is organized as follows. In the next section, we briefly introduce genomes and related terminology and discuss how genome data is currently stored and used. In Section 3, we describe how structured documents can be used to represent complete genomes. In Section 4, we define GXML, our genome-oriented structured document language. In Section 5 we describe a query language that is optimized for use with genome data. Section 6 shows some experimental results for querying information on complete genomes. We follow with an overall discussion in Section 7. The final section concludes the paper.

2. Genome Data

2.1 Genes and Genomes

It is now widely known that *DNA* (deoxyribonucleic acids) form the basis of the inheritance mechanism of organisms. A DNA molecule consists of a pair of two sequences (or *strands*) of four nucleotides (A, T, G and C). These two strands are held together by links acting between A and T, and between G and C nucleotides.

A *gene* is a functional unit of inheritance, mainly corresponding to a segment of the organism's DNA that codes for a protein. *Proteins* can be represented by sequences of 20 types of amino acids. Genes and other segments of DNA that are thought to code for functional entities are collectively called *features*.

Enzymes are highly specific protein catalysts that speed up certain chemical reactions within the cell. Chemical compounds in the cell move from enzyme to enzyme along specific *reaction pathways*, the sum of which determines the chemistry and behavior of the living cell.

The whole DNA sequence of an organism, together with the set of all of its features located on the sequence and associated information, is called its *genome*.

2.2 Genome Databases

Genome data (sequence, pathway, functionality information, etc.) are curated genome-related databases such as GenBank[5)], SWISS-PROT (Swiss Protein Database)[3)], PIR (Protein Information Resource)[4)], and KEGG (Kyoto Encyclopedia of Genes and Genomes)[14)].

Currently, a number of different *flat file* entry formats are in use for retrieving information about genomes (e.g., the GenBank, SWISS-PROT, and PIR formats). A flat file is a plain text file that has a straightforward, non-hierarchical tagged structure.

Figure 1 shows an example of an entry in the GenBank format. Each GenBank entry is composed of tagged fields, each of which is a tag-description pair. For example, the description for the tag `LOCUS` consists of such information as the entry ID and the number of nucleotides in the entry. Each entry contains DNA sequence data in the `ORIGIN` field, and a feature table describing genes that lie on the sequence. A field with the `CDS` (coding sequence) tag contains information about a gene (its position, gene name, etc.) specified by qualifiers such as `/gene`.

2.3 Problems with Data Formats

Unfortunately, even though genome data represented in the various flat file formats may be closely related or even contain cross-references to each other, the formats differ vastly in structure. Also, most of the formats are designed for human-readability, making it difficult to create and maintain tools that can correctly interpret the data contained in the files.

Another major difficulty is that there are inconsistencies, even within the same format. As an example, consider **Fig. 2**, which shows excerpts from the respective GenBank files for the genomes of *Mycoplasma genitalium* and *Aquifex aeolicus*, from Release 110 of the database.

Putative genes (or open reading frames, *ORFs*) are generally identified and assigned a

```
LOCUS       U00096    4639221 bp    DNA   circular  BCT       18-NOV-1998
DEFINITION  Escherichia coli K-12 MG1655 complete genome.
ACCESSION   U00096
KEYWORDS    .
SOURCE      Escherichia coli.
...
REFERENCE   1  (bases 1 to 4639221)
  AUTHORS   Blattner,F.R., Plunkett III,G., Bloch,C.A., ...
  TITLE     The complete genome sequence of Escherichia coli K-12
  JOURNAL   Science 277 (5331), 1453-1474 (1997)
...
FEATURES             Location/Qualifiers
     source          1..4639221
                     /organism="Escherichia coli"
                     /strain="K-12"
...
     CDS             190..255
                     /gene="thrL"
...
     CDS             complement(1317813..1319408)
                     /gene="trpD"
...
     CDS             4638511..4639197
                     /gene="lasT"
...
ORIGIN
        1 agcttttcat tctgactgca acgggcaata ...
       61 tgatagcagc ttctgaactg gttacctgcc ...
...
//
```

Fig. 1 Portions of a sample GenBank entry for a complete genome.

unique ORF ID when the DNA sequence is determined. Later, when the function of the putative gene has been determined through sequence comparison to known genes and/or biological experiments, a gene name is assigned. In the entries shown in Fig. 2, ORF IDs are shown in bold type, while gene names are underlined.

In the file for *Mycoplasma genitalium*, the `/gene` qualifier is used to store the ORF ID, for both putative and known genes, whereas the gene name of a known gene is placed inside the free-text data for the `/product` qualifier. On the other hand, in the file for *Aquifex aeolicus*, the `/gene` qualifier is used to store the ORF ID only for putative genes. For known genes, the ORF ID is moved to the `/note` qualifier, and the `/gene` qualifier is used for the gene name.

Since the GenBank file formats are updated frequently (for example, the qualifiers used for storing ORF IDs in Release 109 were different to those now used in Release 110; `/standard_name` for *Mycoplasma genitalium* and `/label` for *Escherichia coli*), it is difficult to design and maintain tools that overcome such inconsistencies.

3. Genomes as Structured Documents

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, biological reactions occurring within the genome also tend to be hierarchically organized. For example, the Embden-Meyerhof pathway is included in the glycolysis pathway, which is a component of the intermediary sugar metabolic pathway[2),11)].

Since a hierarchical data model can closely represent the structure of the genome, describing genomes as hierarchical structured documents should assist in determining the correspondence between genes. However, genomes and the genetic information they contain cannot be represented directly by structured documents for the following reasons.

- Many bacterial genomes form a circular structure such that their DNA nucleotide sequences are wrapped.
- Although the elements of a structured document cannot overlap, features may overlap if their reading frames are different (see **Fig. 3**) or if they are located on different strands.
- A genome consists of two strands that are complementary to each other. Since each feature (such as a gene) can be located on either strand, it must be possible to easily associate a feature on one strand with another on the complementary strand.

We approach the above problems by first defining a structured document language which can effectively express genome data. We then introduce a query language equipped with a "view" facility for querying genomes without

Mycoplasma genitalium

```
...
(Known gene)
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 (rplK)"
...
(Putative gene)
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

```
...
(Known gene)
gene            3665..4390
                /note="aq_009"
                /gene="rplC"
CDS             3665..4390
                /gene="rplC"
                /codon_start=1
                /product="ribosomal protein L03"
...
(Putative gene)
gene            complement(9657..10157)
                /gene="aq_022"
CDS             complement(9657..10157)
                /gene="aq_022"
                /codon_start=1
                /product="putative protein"
...
```

Fig. 2 Example of inconsistencies within GenBank files.

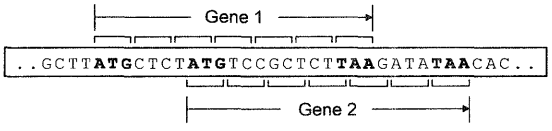


Fig. 3 Overlapping genes on a single strand.

regard to the underlying data representation.

4. GXML: Genome-oriented eXtensible Markup Language

4.1 The Case for XML

Several structured document languages have been proposed, including Abstract Syntax Notation One (ASN.1), SGML, and the eXtensible Markup Language (XML)[23]. Among these languages, ASN.1 has been used as an alternative to the flat file in GenBank. ASN.1 is a powerful language for incorporating relational and object-oriented schemas directly in documents. However, ASN.1 requires strict definitions for all aspects of the schema. It is difficult to modify the structure of ASN.1 documents without adversely affecting the operation of tools that operate on them.

Since relationships between biological entities are rapidly changing due to the progress of biotechnology, we focus on the ability to easily define, modify, and parse the structure and content of the document, rather than representation of strict inter-object relationships. In this case, XML seems a better candidate.

An XML file (see **Fig. 4**) consists primarily of text data surrounded (marked up) by matching start and end tags. Provided the tags nest correctly (i.e, the regions they cover do not overlap), the document is considered to be *well-formed* and can be parsed by any XML-compliant parser. A distinct advantage of XML over other data formats is that XML documents are *self-descriptive*; i.e., information about the structure of a well-formed document can be determined during the parse process, without the necessity for any definition of the document structure. This is highly useful for ad hoc representation of temporary data, as might be transferred between genome projects.

Where it is desired to maintain some degree of consistency between documents, XML allows the definition of a set of rules (known as a Document Type Definition, or *DTD*) that govern the structure of the document. Once a document conforms to a particular DTD, it is considered to be *valid*.

4.2 GXML and the GXML DTD

To cope with the issues described in Section 3, we have designed an XML-based document language called GXML (Genome-oriented eXtensible Markup Language) in consideration of the following points.

- Circular genomes: Since features can overlap, splitting the genome may also split a feature into two segments. We define the attributes `nostart` and `nostop` to indicate that the segments are contiguous across the split, as shown below.

```
<feature>
  <location nostop="true">
    ...
  </location>
  <location nostart="true">
    ...
  </location>
<feature>
```

- Overlapping features and Complementary strands: It must be possible to associate

```
Address-book ::= seq {
  entry {
    id "e001"
    name {
      firstname "John"
      lastname "Smith" }
    tel "0123-45-6789"
    ...
  }
  ...
  entry {
    id "e258"
    ...
  }
```

```
<?xml version="1.0" ?>
<addressbook>
  <entry id="e001">
    <name>
      <firstname>John</firstname>
      <lastname>Smith</lastname>
    </name>
    ...
    <tel>0123-45-6789</tel>
    ...
  </entry>
  ...
  <entry id="e258">
  ...
  </entry>
</addressbook>
```

Fig. 4 Extracts from sample files in ASN.1 (left) and XML format (right).

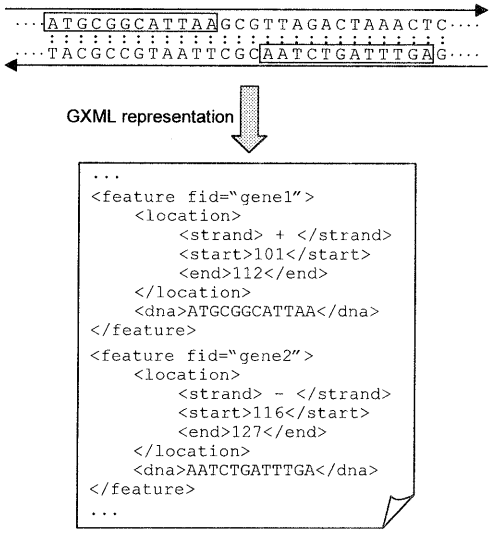


Fig. 5 Representing features on different strands.

features based on their locations regardless of whether or not the features are overlapped and whether or not they are on the same strand. Thus we do not mark up features according to strand; rather, we specify the strand within the location data (see **Fig. 5**).

We have defined a GXML DTD, a condensed form of which is shown in **Fig. 6**. `<!ELEMENT ...>` statements declare the structure and content of elements using a regular expression-like syntax. `<!ATTLIST ...>` statements declare the name, type, and default value for attributes associated with the elements.

In a GXML document, genome 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

```
<!-- An XML DTD for representing complete genomes     -->
<!-- gxml.dtd version 1.2 Jun 1999  Aaron J. Stokes  -->
<!ELEMENT    gxml            ( genome )>
<!ELEMENT    genome          ( gid, whose, date, contig*,
                               feature*, pw*, role* )>
<!ELEMENT    gid             ( #PCDATA )>
<!ELEMENT    whose           ( #PCDATA )>
<!ELEMENT    date            ( #PCDATA )>
<!ELEMENT    contig          ( cid, dna )>
<!ELEMENT    cid             ( #PCDATA )>
<!ELEMENT    dna             ( #PCDATA )>
<!ELEMENT    feature         ( fid, alias*, location+,
                               dna, prot?, function? )>
<!ATTLIST    feature   type  ( orf|rna|intron|intein )
                             #REQUIRED>
<!ELEMENT    fid             ( #PCDATA )>
<!ELEMENT    prot            ( #PCDATA )>
<!ELEMENT    alias           ( #PCDATA )>
<!ELEMENT    location        ( cid, start, end, strand )>
<!ATTLIST    location
             nostart         ( true|false ) "false">
             nostop          ( true|false ) "false">
<!ELEMENT    start           ( #PCDATA )>
<!ELEMENT    end             ( #PCDATA )>
<!ELEMENT    strand          ( #PCDATA )>
<!ELEMENT    function        ( fdescription, confidence?,
                               why? )>
<!ELEMENT    fdescription    ( #PCDATA )>
<!ELEMENT    confidence      ( #PCDATA )>
<!ELEMENT    why             ( #PCDATA )>
<!ELEMENT    pw              ( pid, pwname, rid* )>
<!ELEMENT    pid             ( #PCDATA )>
<!ELEMENT    pwname          ( #PCDATA )>
<!ELEMENT    role            ( rid, rdescription, fid*,
                               substrate*, product* )>
<!ELEMENT    rid             ( #PCDATA )>
<!ELEMENT    rdescription    ( #PCDATA )>
<!ELEMENT    substrate       ( #PCDATA )>
<!ELEMENT    product         ( #PCDATA )>
```

Fig. 6 Outline of the GXML DTD.

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.

An example of a GXML document is shown in **Fig. 7**. As can be seen, the entire DNA sequence of an organism, together with all of the genes and associated information that constitute the genome, are represented in a self-descriptive package of data that is logical and highly machine-readable.

```
<?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>
      <rid>4.1.3.27</rid>
      ...
    </pw>
    ...
    <role>
      <rid>4.1.3.27</rid>
      <rdescription> ... </rdescription>
      <fid>b1263</fid>
      ...
      <substrate>Pyrophosphate</substrate>
      <product>Anthranilate</product>
      ...
    </role>
  </genome>
</gxml>
```

Fig. 7 Example of a GXML genome document.

5. GQL: Genome-oriented Query Language

Several query languages have been proposed to query XML documents, such as XQL[26] and XML-QL[24]. Although these languages offer similar capability in terms of data extraction, XML-QL is considered to be most expressive because XML-QL queries can construct new XML data from the results of queries. Since this data can be used as input to further queries, results can be refined through successive application of queries.

We introduce a new query language, called Genome-oriented Query Language (GQL), that is based on XML-QL and is optimized for extracting and processing the genome data contained in GXML documents.

5.1 Simple Extraction using GQL

A typical GQL query consists of a `WHERE` clause, specifying the structural elements or content which are to be extracted or processed, and a `CONSTRUCT` clause, which specifies how the results will be formed.

For example, the following query obtains a list of the feature IDs for all features contained in `ecoli.gxml`.

```
WHERE
  <feature>
    <fid>$id</>
  </> IN "ecoli.gxml"
CONSTRUCT
  <fid>$id</>
```

Such a query would give the following output.

```
<fid>b0001</fid>
<fid>b0002</fid>
...
<fid>b4403</fid>
```

5.2 GQL Genomic View

GQL also presents an extended genomic 'view' of the underlying data, as illustrated in **Fig. 8**. The genomic view allows one to form queries using biologically meaningful constructs, without having to consider how the data is implemented in the underlying GXML document. We define the following major functions to constitute the view.

- `dist` describes the distance between two features, defined as the number of nucleotides separating the features, regardless of strand. On a circular genome, distance is measured on the path around the genome that returns the least number of nucleotides. If the features overlap, distance is measured between the closest ends and is negated.
- `neighbors` matches features that are neighbors on the genome. The user can specify the maximum distance, in nucleotides, between features for them to be considered as neighbors.
- `upstream`, `downstream` match features that are *upstream* or *downstream* of each other on the same strand of DNA, based on the ordering within the document and accounting for circular genomes.
- `similarity` describes the similarity between the DNA sequences of two features, as calculated by the FASTA[16] program (a string comparison algorithm that is widely used for sequence comparison).
- `besthit` matches *best-hits* across genomes. The best-hit on $genome_2$ of feature f_1 on $genome_1$ is f_2, if f_1 shows greater sequence similarity to f_2 than to any other features on $genome_2$. The notion of a best-hit has been shown to be effective in deducing functional and evolutionary relationships between features across genomes[19].
- `bidirbesthit` matches features that are identical (more exactly, *orthologous*) across

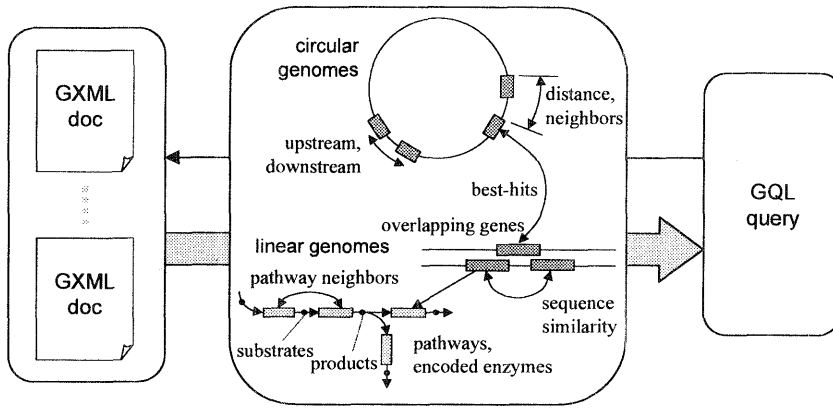


Fig. 8 Genomic 'view' provided by GQL.

genomes by applying `besthit` in both directions.

- `pwneighbors` matches features that encode for enzymes where a product of one enzyme is a substrate of the other enzyme; i.e., the enzymes may be consecutive components of a pathway.

6. Experimental Results

6.1 Prototype System

Based on GXML and our query language GQL, we have implemented a prototype system for querying complete genomes, as shown in **Fig. 9**. Using the Perl language, we created tools that extract data from the GenBank and KEGG databases and convert and merge the data into complete GXML genome documents. A position-based element index[17] is generated for each document.

We created a GQL query processor in the C++ language, which uses the element indices to match patterns and bind GQL variables to elements or content within the GXML documents. The GQL "view" functions are implemented in part as macros that are expanded to GQL conditions on execution, and in part using internal routines or external programs.

The system used in this experiment was a Gateway GP6-400 (Intel Pentium-II 400MHz processor).

6.2 Experiment

In our experiment, we first created GXML genome documents for two complete genomes, *Escherichia coli* and *Bacillus subtilis*, based on data extracted from GenBank Release 110 and a February 21, 1999 download of the KEGG database.

We then performed several test queries based on the GXML genome documents. Each of the queries is described in detail below, making particular note of the biological significance of the results obtained.

6.2.1 Exploring Functional Relationships on a Single Genome

The first test query demonstrates how our method can be used to explore functionally related genes on a single genome, using sequence similarity. In natural language, we can express our query in the following way.

Query 1:

"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."

Two genes are considered to be *paralogous* if they lie on the same genome and their sequences exhibit a certain degree of similarity to each other. Biologically, exposing this relationship can be useful in determining genes that were derived from a duplication event on the genome during the process of evolution. Such genes often share common functionality. In our test query, we consider two genes to be paralogous if they are located on the same genome and the FASTA similarity score is at least 200. This threshold could be adjusted easily if desired.

In our query language, the query can be expressed as follows.

```
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,
```

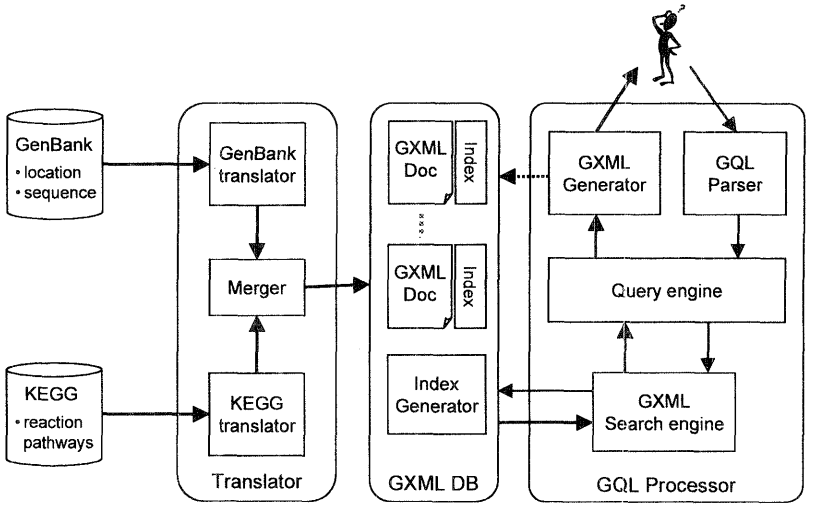


Fig. 9 The prototype system.

```
    neighbors($p1,$p2),
    similarity($phoP,$p1,$s1),
    similarity($phoQ,$p2,$s2),
    $s1 >= 200, $s2 >= 200
  CONSTRUCT
    <pair>
      $p1
      $p2
    </> TO "query1.gxml"
```

The first two patterns bind the variables `$phoP` and `$phoQ` to the `feature` elements corresponding to the genes *phoP* and *phoQ*, respectively. The following two patterns bind `$p1` and `$p2` to all `feature` elements in turn.

The next two lines of the query stipulate that neither `$p1` nor `$p2` can be bound to the same features to which `$phoP` and `$phoQ` are bound; i.e., we exclude the genes *phoP* and *phoQ* from the search.

The remaining four lines simply state that `$p1` and `$p2` must refer to neighboring features and that the similarity scores between *phoP* and the feature bound to `$p1`, and between *phoQ* and the feature bound to `$p2`, must not be less than 200.

The `CONSTRUCT` block creates a new file, `query1.gxml`, and then outputs a new `<pair></pair>` instance for every unique permutation of values for `$p1` and `$p2`.

A shown in **Fig. 10**, ten pairs of gene neighbors were isolated by the query. The fact that these pairs are all paralogous to the gene pair (*phoP*, *phoQ*) suggests that they may have similar functionality to *phoP* and *phoQ*.

This hypothesis was confirmed in biological literature[13], where Mizuno reports that the pair (*phoP*, *phoQ*) and the ten pairs we obtained are, indeed, functionally related. The genes function together as cognate sensor/regulator pairs. This is illustrated graphically in **Fig. 11**.

```
<pair> ... phoB ... phoR ... </pair>
<pair> ... baeR ... baeS ... </pair>
<pair> ... f227 ... f480 ... </pair>
<pair> ... ompR ... envZ ... </pair>
<pair> ... kdpE ... kdpD ... </pair>
<pair> ... cpxR ... cpxA ... </pair>
<pair> ... rstA ... rstB ... </pair>
<pair> ... basR ... basS ... </pair>
<pair> ... f239 ... f452 ... </pair>
<pair> ... creB ... creC ... </pair>
```

Fig. 10 Results for Query 1.

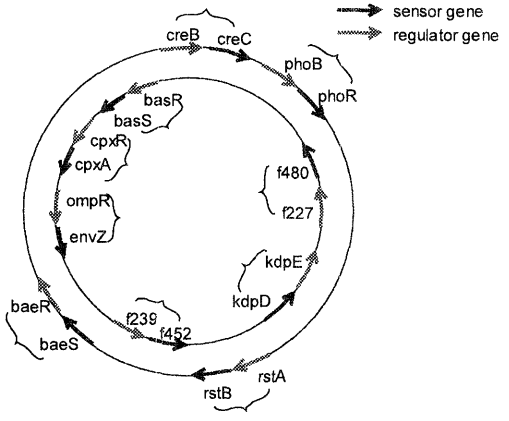


Fig. 11 Graphical depiction of results for Query 1.

6.2.2 Corresponding Genome Location and Pathway Functionality

The second test query shows how we can use GQL to elucidate a correspondence between the location of genes on the genome and the functionality of the encoded proteins on a pathway. More specifically, the objective is to show that some genes that are neighbors on a genome also encode for proteins that are consecutive components of a pathway.

Query 2:

"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 GQL in the following way.

```
WHERE
  <feature></> AS $f1
  <feature></> AS $f2 IN "ecoli.gxml",
  neighbors($f1,$f2),
  upstream($f1,$f2),
  pwneighbors($f1,$f2,$pw)
CONSTRUCT
  <pathway>
    $pw
    ( <pair>
        $f1
        $f2
      </> )
  </> TO "query2.gxml"
```

The first two lines of the query bind `$f1` and `$f2` to each pair of features on the genome in turn. The following three lines specify that the two features must be neighbors on the genome and that proteins encoded by the features must be neighbors on a pathway. Note that our definition of `neighbors` allows for a gene to be its own neighbor.

The `CONSTRUCT` block appears rather more complex than that of Query 1 because we want to group the resultant gene pairs according to the pathways for which they encode proteins. We achieve this in GQL by enclosing the `<pair>` pattern in parentheses.

A portion of the output is shown in **Fig. 12**. Several similar groups of gene pairs were obtained for other pathways, but without loss of accuracy we can restrict our discussion to the results shown. The figure shows six pairs of genes that were found to encode for the *tryptophan biosynthesis* pathway.

We investigated the accuracy of the result, and found that the biological literature[8] confirms that the pairs we found are indeed consecutive members of the tryptophan biosynthesis pathway. The relationships thus exposed are illustrated in **Fig. 13**.

```
...
<pathway>
    <pw> ... tryptophan biosynthesis ... </pw>
    <pair> ... trpD ... trpD ... </pair>
    <pair> ... trpE ... trpD ... </pair>
    <pair> ... trpD ... trpC ... </pair>
    <pair> ... trpC ... trpC ... </pair>
    <pair> ... trpC ... trpA ... </pair>
    <pair> ... trpC ... trpB ... </pair>
</pathway>
...
```

Fig. 12 Partial results for Query 2.

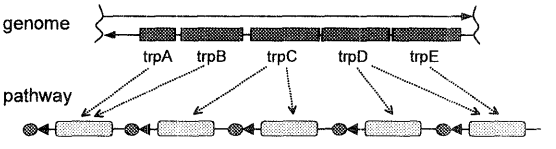


Fig. 13 Graphical depiction of results for Query 2.

Several interesting facts are revealed by the results of the query. Firstly, the fact that we obtained the two pairs (*trpC*, *trpC*) and (*trpD*, *trpD*) indicates that we can correctly identify genes which are *multifunctional*. Multifunctional genes encode for proteins with two or more different functions. Secondly, the results indicate that *trpA* and *trpB* may be *sub-units*. These are genes that work together to encode a single protein.

It is interesting to note that the tryptophan biosynthesis pathway is shown in 8) to be an example of a pathway where gene order is conserved significantly between different genomes. In future tests, we could extend our query to include multiple genomes in an attempt to expose this relationship.

6.2.3 Corresponding Genes across Genomes

The third test query illustrates how our query language can be used to determine the correspondence between genes on different genomes, and thus highlight possible functional relationships.

Specifically, we consider a pair of gene neighbors on one genome whose functionality is known, and search for neighbors on another genome that are orthologous to the reference pair. The query can be expressed in natural language as follows.

Query 3:

"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."

As mentioned in Section 5.2, two genes can be considered as orthologous (they correspond across genomes) if they are bi-directional best-hits. Hence we arrive at the following GQL representation of the query.

```
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,$ecoli),
  bidirbesthit($sdbB,$bsub,$g2,$ecoli)
CONSTRUCT
  <pair>
    $g1
    $g2
  </> TO "query3.gxml"
```

The first two lines of the query bind `$sdhA` and `$sdhB` to the `feature` elements that represent the genes *sdhA* and *sdhB*, respectively, on the genome of *Bacillus subtilis*. The next two lines bind `$g1` and `$g2` to all pairs of `feature` elements on the genome of *Escherichia coli*.

The remaining conditions add the restrictions that `$g1` and `$g2` must bind to neighboring genes, that `$g1` must bind to a gene that is a bi-directional best hit of *sdhA*, and that `$g2` must bind to a gene that is a bi-directional best hit of *sdhB*. We are not concerned with the dummy parameters `$bsub` and `$ecoli`; we implicitly specify the source genome when we specify the source files `ecoli.gxml` and `bsub.gxml`.

The result of the query is 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, it is confirmed in 7) that both pairs of genes participate in the citric acid cycles, also referred to as TCA cycles, of their respective genomes. The citric acid cycle is a well-known pathway that is central to the energy metabolism in many organisms.

It is important to note that our definitions of gene distance and gene neighbors allowed us to obtain a biologically meaningful result regardless of the fact that the genes *frdA* and *frdB* are overlapping on the genome.

6.3 System Performance

We first measured the user CPU time required to create and index two GXML genome documents, as shown in **Table 1**. Translation and index creation time totals were under 10 seconds for each genome. 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 the translation times achieved are highly acceptable.

We then performed each test query, measuring the times required to perform each phase of query execution. The results are shown in **Table 2**. It is evident that the bulk of the time required to execute each query was consumed in executing joins between bound variables. Pattern matching and FASTA calculation times were found to be acceptable.

Although each query required approximately 9 minutes to execute, this is reasonable when we consider the ease with which a complex analytical query can be written and performed on data for several complete genomes.

7. Discussion

7.1 Translation into GXML

The data stored in KEGG and GenBank is updated infrequently in relation to the time required for translation into GXML. From this it would seem that the process is efficient and could easily be automated. However, a fully automatic translation system is difficult to implement for the following reasons.

- Translators must be able to cope with complex database schemas and inconsistencies on both the syntactic and semantic levels. For example, a GenBank translator would require dedicated parsing routines for many genomes, because of the inconsistencies described in Section 2.3.
- Database schemas and flat file formats are updated frequently, and in some cases the changes may go undetected, causing erroneous results. For example, if a qualifier currently used to store one type of data is instead used to store a different type of data in a future release, the change may not be detected since the qualifier still exists and contains some form of data.
- It is difficult to find a complete and accurate method for corresponding the data trans-

Table 1 Performance of GXML translator and index generator.

organism	genes	pathways	enzymes	translation time	index creation time	GXML doc size
Escherichia coli	4405	81	548	6.76	2.78	11,566,197 bytes
Bacillus subtilis	4221	81	425	6.20	2.37	10,391,654 bytes

Table 2 Performance of GQL query processor.

query	pattern matching/binding	joining	FASTA calculation	function expansion/other	total
Query 1	5.48	429.49	15.82	11.60	462.39
Query 2	8.78	484.09	—	10.22	503.09
Query 3	4.91	462.72	55.34	9.88	532.85

lated from multiple data sources, due to the heterogeneity of their schemas. In the case of GenBank and KEGG, it is necessary to follow several transitive correspondences to collect data for a particular gene.

- Some form of automatic version control is required to cope with differences in the data contained in the databases. For example, if the contents of KEGG are updated less frequently than GenBank, the most recent version of GenBank data might not necessarily be the optimal one for integration with KEGG data.

The idea of a *trusted data set* may be of use in solving some of these problems. We could translate from a specific set of GenBank and KEGG flat files into a specific set of GXML documents, based on a set of translation and integration rules. When the flat file formats are updated, the rules would be revised automatically based on a comparison between the updated files and the trusted set, knowing the expected output. A final consistency check of the GXML output could then be performed based on rules derived from the trusted document set.

7.2 Query Optimization

It is evident from Table 2 that the execution of joins is more computationally intensive than any other process in the prototype GQL system, which incorporates a naive nested-loop join algorithm. Since GQL functions such as `neighbors`, `similarity` and `upstream` match pairs of features, this generally results in a double nested-loop iteration of over 4000 features per loop for any query incorporating these functions. This indicates why the join times were similar for each of the test queries.

We can use the constraints inherent in GQL functions to remove some of the nested iterations. For example, in Query 2, the naive implementation would have `$f1` and `$f2` forming a double nested-loop, testing the functions `neighbors`, `upstream`, and `pwneighbors` for each iteration. By noting that the `neighbors` function matches only pairs of neighbors on the same genome, we can remove the `$f2` loop. For each iteration of the remaining loop, we consider `$f1` as a key, for which `neighbors`, `upstream`, and `pwneighbors` return sets of candidate values for `$f2`. Since these sets are of constant size (and quite small), an intersection between the sets will result in values for `$f2` in constant time.

The same technique can be applied to Query 1, where we can use the constraint on `neighbors` to remove the loop formed by `$p2` and achieve a significant increase in efficiency. For Query 3, we can take advantage of the constraints that we expect only one feature given its gene name (`alias`), and that `bidirbesthit` returns at most only one match per genome. Pattern-matching would first bind `$sdhA` and `$sdhB`, and then `bidirbesthit` can be executed to find candidate sets for `$g1` and `$g2`. A test of whether `$g1` and `$g2` are neighbors or not is readily accomplished, and a check of whether `$g1` (or `$g2`) is in `ecoli.gxml` concludes the iteration.

7.3 Future Work

Our data model is limited in that it only supports implicit linking between elements within a single document. We are investigating the use of XLink and XPointer to allow for more efficient representation of relationships between GXML documents.

A highly useful addition to GQL functionality would be the capability of calculating transitive closures. This would allow us, for example, to achieve a meaningful sequence of genes rather than a set of gene pairs in Query 2: the resultant sequence would directly represent the gene ordering on the pathway. However, it is known to be difficult to optimize transitive closure queries since they are generally recursive. A number of recursion evaluation techniques have been designed, including the well-known *magic sets* transformation[6]. In our fu-

ture work, we will consider incorporating magic sets transformation in GQL to allow the processing of transitive closures.

8. Conclusion

We have developed a method for representing genomes and associated data as structured documents. Since genome data have complex structures, several extensions from extant structured document languages were required. We have also designed a query language equipped with biological constructs to act on the genome documents.

From the results of executing several test queries, our prototype system obtained several biologically meaningful results with acceptable performance. In practice, our method has proven useful to several genome projects as a method of exchanging genome data. The overall perspective of complete genomes that our method provides is a unique tool for genomic research.

Acknowledgments 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.

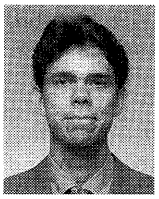
References

1) Aiba, H. et al.: A 570-kb DNA Sequence of the *Escherichia coli* K-12 Genome Corresponding to the 28.0-40.1 min Region on the Linkage Map, *DNA Research*, Vol.3, pp.363–377 (1996).
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*, Vol. 27, No. 1, pp. 49–54 (1999).
4) Barker, W. C., Garavelli, J. S., McGarvey, P. B. et al.: The PIR-International Protein Sequence Database, *Nucleic Acids Research*, Vol. 27, No. 1, pp. 39–43 (1999).
5) Benson, D. A. et al.: GenBank, *Nucleic Acids Research*, Vol. 27, No. 1, pp. 12–17 (1999).
6) Bancilhon, F., Maier, D. et al.: Magic sets and other strange ways to implement logic programs, *Proceedings of the ACM Symposium on Principles of Database Systems*, Cambridge, Massachusetts, pp. 1–15 (1986).
7) 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, pp. 206–216 (1996).
8) Dandekar, T., Snel, B. et al.: Conservation of Gene Order: a Fingerprint of Proteins that Physically Interact, *Trends in Biochemical Science*, Vol. 23, No. 9, pp. 324–328 (1998).
9) Davidson, S.B., Overton, C., and Buneman, P.: Challenges in Integrating Biological Data Sources, *J. Computational Biology*, Vol.2, No.4, pp. 557–572 (1995).
10) Davidson, S. B., Overton, C., Tannen, V., and Wong, L.: BioKleisli: A Digital Library for Biomedical Researchers, *J. Digital Libraries*, Vol. 1, No. 1 (1996).
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, pp. 189–198 (1996).
12) Itoh, T. et al.: A 460-kb DNA Sequence of the *Escherichia coli* K-12 Genome Corresponding to the 40.1-50.0 min Region on the Linkage Map, *DNA Research*, Vol.3, pp.379–392 (1996).
13) Mizuno, T.: Compilation of All Genes Encoding Two-component Phosphotransfer Signal Transducers in the Genome of *Escherichia coli*, *DNA Research*, Vol. 4, pp. 161–168 (1997).
14) Ogata, H., Goto, S., Sato, K., Fujibuchi, W. et al.: KEGG: Kyoto Encyclopedia of Genes and Genomes, *Nucleic Acids Research*, Vol. 27, No. 1, pp. 29–34 (1999).
15) Oshima, T. et al.: A 718-kb DNA Sequence of the *Escherichia coli* K-12 Genome Corresponding to the 12.7-28.0 min Region on the Linkage Map, *DNA Research*, Vol.3, pp.137–155 (1996).
16) Pearson, W. R. and Lipman, D. J.: Improved Tools for Biological Sequence Comparison, *Proc. Natl. Acad. Sci. USA*, Vol. 85, pp. 2444–2448 (1988).
17) Sacks-Davis, R., Dao, T. et al.: Indexing Documents for Queries on Structure, Content and Attributes, *Proc. International Symposium on Digital Media Information Base (DMIB'97)*, pp. 236–245 (1997).
18) Skupski, M. P., Booker, M., Farmer, A. et al.: The Genome Sequence DataBase: Towards an Integrated Functional Genomics Resource, *Nucleic Acids Research*, Vol. 27, No. 1, pp. 35–38 (1999).
19) Tatusov, R. L. et al.: Metabolism and Evolution of *Haemophilus influenzae* Deduced from a Whole-Genome Comparison with *Escherichia coli*, *Current Biology*, Vol. 6, No. 3, pp. 279–291 (1996).

20) Tamames, J., Casari, G., Ouzounis, C. and Valencia, A.: Conserved Clusters of Functionally Related Genes in Two Bacterial Genomes, *J. Molecular Evolution*, Vol.44, No.1, pp.66–73 (1997).
21) Thierry-Mieg, J. and Durblin R.: ACeDB – A *C. elegans* Database: Syntactic Definitions for the ACEDB Data Base Manager, 1992.
22) The WIT Project, available at `http://wit.mcs.anl.gov/WIT2/`.
23) Bray, T., Paoli, J. and Sperberg-McQueen, C.M. ed.: Extensible Markup Language (XML) 1.0, *W3C Recommendation 10-Feb-98*, available at `http://www.w3.org/TR/REC-xml`.
24) 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`.
25) Ritter, O.: The Integrated Genomic Database, *Computational Methods in Genome Research* (S. Suhai.(ed.)), Plenum Press, New York, pp. 57–73 (1994).
26) Robie, J., Lapp, J. and Schach, D.: XML Query Language (XQL), *W3C Proposal September 1998*, available at `http://www.w3.org/TandS/QL/QL98/pp/xql.html`.
27) Yamamoto, Y. et al.: Construction of a Contiguous 874 kb Sequence of the *Escherichia coli* K-12 Genome Corresponding to the 50.0-68.8 min Region on the Linkage Map and Analysis of its Sequence Features, *DNA Research*, Vol.3, pp. 91–113 (1997).

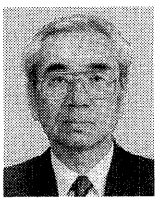
(Received March 20, 1999)
(Accepted June 27, 1999)

(Editor in Charge: *Kazumasa Yokota*)



Aaron J. Stokes was born in 1973. He received his B.Eng. from Osaka University in 1999, and is currently a research assistant at CREST of JST (Japan Science and Technology). His research interests include database applications in molecular biology.

Hideo Matsuda was born in 1959. He received his B.Sc., M.Eng. and Ph.D. degrees from Kobe University in 1982, 1984 and 1987 respectively. From 1984 to 1990, he was a research associate and, from 1990 to 1994 a lecturer of the Department of Systems Engineering at Kobe University. He was a visiting scholar of Mathematics and Computer Science Division at Argonne National Laboratory from 1991 to 1992. He is now an associate professor of Department of Informatics and Mathematical Science at Osaka University. His research interests include bioinformatics and molecular biology database. He is a member of IEICE, IEEE CS and ACM.



Akihiro Hashimoto was born in 1938. He received his B.Eng., M.Eng. and Dr.Eng. degrees from Osaka University in 1961, 1963 and 1966, respectively. He worked in NTT Laboratories from 1966 to 1989 and was engaged in research on fault diagnosis and design automation in computer systems and development of the DIPS system. He was a visiting assistant professor of University of Illinois from 1969 to 1971. He is now a professor of the Department of Informatics and Mathematical Science at Osaka University. His research interests include information processing technology in molecular biology. He is a member of IEICE, IEEE and ACM.