データベースシステム 119−40
（1999. 7. 22）

Introduction of Aggregate Functions to a Language for Querying Structured Genome Documents

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

In our previous work, we introduced a novel method for exchanging and querying complete genome data by representing them as structured documents. We defined an XML-based genome language called GXML, and a genome-oriented query language called GQL for making biologically meaningful queries on GXML genome documents. In this paper, we extend GQL with aggregate functions and sorting capability to further increase the usefulness of our method. We discuss the additional syntax and present several examples of biological queries that require aggregation. We are currently investigating algorithms for implementing the new constructs in a prototype system.

1. Introduction

In a previous paper[5], we introduced a query language called Genome-oriented Query Language (GQL), that is based on XML-QL[4] and acts on structured documents representing complete genomes. However, the language as defined lacks sorting and aggregation capability, which are necessary for performing many types of biologically meaningful queries. In this paper, we discuss the introduction to GQL of constructs relating to sorting, aggregation, query nesting, and duplicate removal.

This paper is organized as follows. The following section gives brief descriptions of GXML, our genome document format, and GQL in its present definition. In Section 3, we describe how we can extend GQL to enhance its usefulness. Section 4 shows through examples how we incorporate the new constructs into the GQL syntax. A discussion follows in Section 5. The final section concludes the paper.

2. GXML and GQL

2.1 GXML

GXML (Genome-oriented eXtensible Markup Language) is an application of the eXtensible Markup Language (XML)[2] that describes how to represent in a self-descriptive *genome document* the entire DNA sequence of an organism, together with all its genes and associated information that constitute its genome. The hierarchical data model provided by GXML closely represents the structure of the genome, including the following types of structure that are difficult to represent directly in structured documents.

- Circular genomes (wrapped DNA)
- Overlapping genes
- Complementary DNA strands

An excerpt from a sample GXML document is shown in **Fig. 1**.

2.2 GQL

Some methods have been proposed for querying XML documents by translating them into object-oriented or relational data formats, and interpreting the query using traditional database management systems. However, in the case of GXML this is impractical for the following reasons.

- A major strength of XML and other semi-structured document languages is their flexibility to schema evolution[1]. Unlike object-oriented or relational schemas, the structure of XML documents can be modified significantly without affecting the reachability of existing data and without requiring reconstruction of the dataset. Schema changes in genome data are particularly frequent, due to rapid progress in the field of molecular biology.
- The ability to query XML documents based not only on content but also on structure is important. 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.

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

```
<?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. 1 Example of a GXML genome document.

To overcome these difficulties, we introduced a query language called Genome-oriented Query Language (GQL), that is based on XML-QL and allows the user to query GXML documents directly. We also defined some functions in GQL that constitute a "view" of important biological relationships within and between genomes (see **Fig. 2**).

A GQL query is composed of a `WHERE` block, which consists of pattern expressions and conditions that describe how to bind variables to elements and content within documents, and a `CONSTRUCT` block, which specifies a template that describes how the bound values are to be presented in the output. The following query example demonstrates the use of pattern-matching and the GQL view function `neighbors` to find all genes that are neighbors of the *trpD* gene on the `Escherichia coli` genome.

```
WHERE
  <feature>
    <alias>"trpD"</>
  </> AS $trpD
  <feature></> AS $f
    IN "ecoli.gxml",
  neighbors($trpD,$f)
CONSTRUCT
  $f
```

3. Extending GQL

GQL, as defined in our earlier work, was sufficient for some specific queries, but lacked the following functionality that we consider vital to any complete query language.

- Ordering of output

There was no provision for ordering elements in the results of the query. Since the output is also an XML document, ordering is important not only for the sake of clarity to human viewers, but also because many GQL functions (`upstream`, `neighbors`, etc.) take advantage of ordering to greatly boost evaluation efficiency.

- Specification of output range

Since some of the relationships between biological entities such as genes cannot be expressed in binary terms, functions such as `besthit` that return either a single result or none may not be appropriate for some queries. Moreover, although it is possible to specify a score cut-off for the `similarity` function, it is difficult to decide on a value for the cut-off without first viewing some rough results. We desire some functionality for specifying range in GQL queries (e.g., 'Ouput only the top x matches ...').

- Nested queries

Although we can specify the hierarchy of the output file in the `CONSTRUCT` block, the entire hierarchy is repeated once for each set of values bound to the variables within the block. We need to be able to nest queries to allow for an arbitrary number of child elements under a particular element.

- Distinct values

In GQL, variables are bound based on all matches encountered for the specified patterns. As a result, the output may contain duplicate sets of bindings. Although this is useful in some queries, a method for removing duplicate sets of bindings is also desired.

4. Query Examples

4.1 Sorting and Range Specification

We first introduce the `ORDER BY` construct for sorting the results of queries.

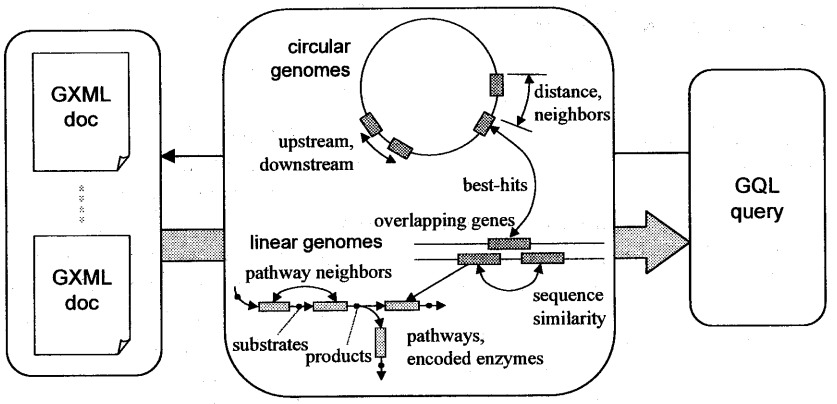

Fig. 2 Genomic 'view' provided by GQL.

```
<OrderBy-Clause> ::= "ORDER"
  <ObjectVars> "BY" <SortVars>
<ObjectVars> ::= <Var> |
  <ObjectVars> "," <Var>
<SortVars> ::= <SortVar>
  | <SortVars> "," <SortVar>
<SortVar> ::= <Var> ["DESC"|"ASC"]
```

Here `<Var>` refers to a GQL variable such as `$myvar`. `<ObjectVars>` explicitly specifies the list of variables whose bindings are to be sorted. `<SortVars>` specifies the variables whose values are to be used as sort keys.

We also introduce range specification syntax within the `CONSTRUCT` block:

```
<Construct> ::=
  "CONSTRUCT" [<RangeSpec>]
  ["DISTINCT"] <Template>
<RangeSpec> ::= "TOP" <PosInt> ["%"]
```

The following example illustrates the use of the new constructs (see Section 4.4 for an example using `DISTINCT`).

"Extract the 10 genes on the *Bacillus subtilis* genome that exhibit highest similarity to the *frdA* gene on the *Escherichia coli* genome. Output should be in descending order of similarity to *frdA*."

Query 1

```
1 WHERE
2   <feature>
3     <alias>"frdA"</>
4   </> AS $frdA IN "ecoli.gxml",
5   <feature></> AS $f2
6     IN "bsub.gxml",
7   similarity($frdA,$f2,$sim)
8 ORDER $f2 BY $sim DESC
9 CONSTRUCT TOP 10
10   $f2
```

4.2 Nested queries

A query can be specified in the `CONSTRUCT` block of another query. For every match found in the outer query for variables that are common between the two queries, the inner query is executed and the results of its `CONSTRUCT` block are incorporated into the output.

As an example, we reconsider Query 2 from our original paper[5]:

"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*. "

This query was found useful in confirming some interesting biological relationships[3], including those shown in **Fig. 3** for the *tryptophan biosynthesis* pathway. However, some degree of ad hoc post-processing was required because the matches were both unstructured, as depicted below, and also unsorted.

```
<result>
  <pw>...</pw>
  <feature>...</feature>
  <feature>...</feature>
</result>
...
<result>
  <pw>...</pw>
  <feature>...</feature>
  <feature>...</feature>
</result>
```

Now that we have equipped GQL with the ability to nest queries, it becomes possible to

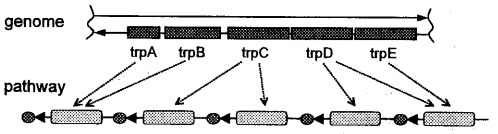

Fig. 3 Graphical depiction of partial results from Query 2.

extend the query for more meaningful output:

"... Group the features by pathway in the result, sort pathways by pathway name, and sort features by gene name."

In this case we might expect the following highly structured and sorted output, from which it should not be so difficult to visualize the relationships such as those in Fig. 3.

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

The final query could be written as follows.

Query 2

```
 1 WHERE
 2   <pw>
 3     <pwname>$pwname</>
 4   </> AS $pw IN "ecoli.gxml"
 5 ORDER $pw BY $pwname
 6 CONSTRUCT
 7   <neighborsbypathway>
 8     $pw
 9     WHERE
10       <feature><alias>$a1</></> AS $f1
11       <feature><alias>$a2</></> AS $f2
12         IN "ecoli.gxml",
13       neighbors($f1,$f2),
14       upstream($f1,$f2)
15       pwneighbors($f1,$f2,$pw)
16     ORDER $f1, $f2 BY $a1, $a2
17     CONSTRUCT
18       <neighborpair>
19         $f1
20         $f2
21       </>
22   </>
```

4.3 Aggregation

We now introduce the **GROUP BY** and **HAVING** clauses for aggregation. The syntax of the former is identical to that of **ORDER BY**.

```
<GroupBy-Clause> ::= "GROUP"
  <ObjectVars> "BY" <SortVars>
<Having-Clause> ::= ["HAVING"
  <ConditionList>]
<ConditionList> ::= <Condition> |
  <ConditionList> "," <Condition>
```

The **GROUP BY** clause is used in exactly the same way as the **ORDER BY** clause to specify how to order values. In this case, however, each unique combination of sorted values is assigned a *bin*, and aggregation is performed over each bin. Aggregate functions include **count**, **sum**, **min**, **max** and **avg**. The **HAVING** clause enforces conditions on the results of the aggregate functions.

Within the **CONSTRUCT** block, variables specified in **<SortVars>** may not appear as parameters of aggregate functions, while those specified in **<ObjectVars>** must.

The following example demonstrates the use of aggregation to check whether any features have been assigned the same gene name. Although there are some special cases, this kind of duplication is often the result of some kind of error.

"Are there any features on the *Escherichia coli* genome that have the same gene name? If so, produce a list of gene names and their respective features."

Query 3

```
 1 WHERE
 2   <feature>
 3     <alias>$a</>
 4   </> AS $f
 5     IN "ecoli.gxml"
 6 GROUP $f BY $a
 7 CONSTRUCT
 8   <dupgenes>
 9     <genename>$a</>
10     <featurelist>
11       WHERE
12         <feature>
13           <alias>$a</>
```

```
14          </> AS $f
15            IN "ecoli.gxml"
16        CONSTRUCT
17          $f
18      </>
19    </>
20 HAVING
21    count($f) > 1
```

4.4 Distinct Values

Finally, we introduce **DISTINCT** to avoid duplication of output.

"List IDs of features of *Escherichia coli* that are capable of encoding component enzymes of the *tryptophan biosynthesis* pathway."

In many pathways, the same enzyme (role) is found in multiple locations throughout the pathway. Consequently, a naive query would produce a list with many duplicate features. The following query uses **DISTINCT** to produce a more favorable result.

Query 4

```
 1 WHERE
 2   <pw>
 3     <pwname>"tryptophan"</>
 4     <rid>$rid</>
 5   </> as $pw
 6   <role>
 7     <rid>$rid</>
 8     <fid></> AS $fid
 9   </> IN "ecoli.gxml"
10 CONSTRUCT DISTINCT
11   $fid
```

5. Discussion

The XML-QL proposal[4] does in fact include syntax for sorting using an **ORDER-BY** clause. The example query given in the proposal is quoted below.

```
1 WHERE
2   <pub>$p</> IN ...
3   <title>$t</> IN $p
4   <year>$y</> IN $p
5   <month>$z</> IN $p
6 ORDER-BY $y, $z
7 CONSTRUCT $t
```

The **ORDER-BY** clause syntax given appears more straightforward than that which we propose. However, it turns out that there are several semantic problems with the above approach:

(1) If **$p** happens to contain multiple **year** elements (which is possible if the publication is reprinted, for example), which **$y** value should be used as a key for that publication when sorting?

(2) In the simple example shown, **$p** is bound to a publication, and **$t**, **$y**, and **$z** are explicitly bound to content within **$p** using the **IN** construct. One of the benefits of XML-QL being its pattern-matching capability, we might prefer to rewrite the query as shown below.

```
1 WHERE
2   <pub>
3     <title>$t</>
4     <year>$y</>
5     <month>$z</>
6   </> IN ...
7 ORDER-BY $y, $z
8 CONSTRUCT $t
```

Having done so, however, the sorting process becomes highly ambiguous: What exactly is subject to sorting here?

(3) Once data from multiple sources is brought into the **WHERE** block, the situation becomes even more ambiguous. The **ORDER-BY** clause, like in SQL, expects that some kind of relationship exists between **$y**, **$z**, and **$t** on which **$t** can be sorted. However, XML-QL is different to SQL in the sense that a variable is not necessarily associated with any particular data *source*; rather, it is associated with a *pattern*.

In consideration of these points, in GQL we have elected to specify within the **ORDER BY** and **GROUP BY** clauses which variables are subject to sorting (aggregation), and which are used as a basis for the operation. This explicit specification removes the ambiguities found in the XML-QL syntax.

6. Conclusion

We have proposed several extensions to our genome document query language GQL, that widen the variety of queries which can be formed and enhances the ease with which query results can be interpreted. Some of our extensions do not conform exactly to equivalent constructs that have been proposed for XML-QL, since we found several unsatisfactory points regarding the semantics of those constructs.

We are currently working on algorithms for

incorporating these extensions into our prototype GQL system.

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) Abiteboul, S.: Querying Semi-Structured Data, In *Proceedings of the International Conference on Database Theory*, pp. 1–18 (1997).

2) 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`.

3) 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).

4) 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`.

5) Stokes, A.J., Matsuda, H., and Hashimoto, A.: GXML: A Novel Method for Exchanging and Querying Complete Genomes by Representing them as Structured Documents, DEWS'99, 5B-4 (1999).