

Support for BioIndexing in BLASTgres

Ruey-Lung Hsiao, D. Stott Parker, and Hung-chih Yang

Computer Science Department, UCLA, CA 90095-1596, USA
{rlhsiao,stott,hcyang}@cs.ucla.edu

Abstract. The ability to perform genome-wide and cross-genome data analyses can dramatically reduce the time required for new biological discoveries. This raises important issues in bioinformatics database research involving data representations and data integration. Essential biological datatypes (such as sequence locations) and tools (such as the popular BLAST sequence alignment tools) are not supported in traditional database systems, which has forced researchers to represent biological knowledge counterintuitively, and implement codes for data operations. This paper introduces BioIndexing, a conceptual infrastructure for representing and managing biological information that permits this information to be queried within a modern database system. The paper also describes an implementation — BLASTGRES, an extension of the POSTGRESQL database system — that provides indexable bioinformatics datatypes and joinable BLAST alignment. The sequence location datatype in BLASTGRES is of specific interest, since it is indexable, essential to the sequence alignment information produced by BLAST, and pervasive in existing biological information.

1 Introduction

Database management systems are widely used in large-scale bioinformatics pipelines. They provide efficient and stable data repositories for effective biological knowledge discovery. With the exponential growth in data scale and the increasing complexity of problems that researchers are trying to solve, developing an efficient and effective database management system is becoming a central topic of bioinformatics.

Traditional database systems, however, were not designed with bioinformatics applications in mind. Their lack of support for essential biological datatypes and functionality forces bioinformatics researchers to represent biological knowledge in ad hoc ways and implement their own transformations on the data. Moreover, dynamic and complex relationships exist between biological data entities. These relationships between entities may be invalidated by new scientific discoveries, and new relationships may be added over time. The traditional fixed database schema is inappropriate for representing biological knowledge. Furthermore, a general ability to connect information in different formats is required.

Biological databases typically require support for several key capabilities. Figure 1 highlights common needs of databases that manage feature information

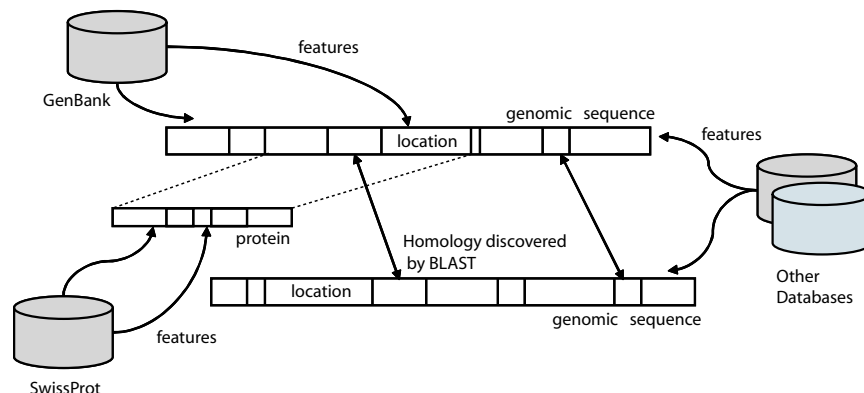


Fig. 1. Much biological knowledge is described in terms of the features associated with locations in sequences. BioIndexing is a conceptual infrastructure for representing and managing this knowledge in large-scale database systems. BLASTgres, the database system described in this paper, supports BioIndexing by providing a location datatype and “BLAST joins” between locations. These two capabilities permit efficient large-scale management of sequence-related feature knowledge.

(large-scale maps between sequence locations and annotational data) from diverse data sources. Three core capabilities required are support for: 1) datatypes implementing central biological abstractions, such as sequence, range, location, as well as the extensibility required by exploratory research involving these datatypes; 2) efficient indexing of biological datatypes; 3) sequence alignment primitives, including integration of popular tools such as the BLAST family [2].

These three capabilities intersect in the notion of indexing. The sequence map concept that is ubiquitous in bioinformatics is a kind of index, relating features to sequence locations, and sequence alignment is a related form of indexing. Indexing is a common emphasis for organizing and using biological information.

This paper argues that this emphasis on indexing is fundamental to biological information management, both in the way this information is represented and in the way it is used. Today biological information is vast, distributed, and connected with sequences. It is typically explored in a navigational manner, both through browser interfaces and automated traversal of maps. We can capitalize on this emphasis if we can develop database systems that address it directly.

BioIndexing is a conceptual infrastructure for representing and managing biological information with indexing constructs. BioIndexing is not just an attempt to model the way indexing is used in managing biological knowledge, but also a scheme for connecting and querying this information within modern database systems. This paper shows how databases can be developed that implement this indexing infrastructure.

This paper specifically concentrates on two aspects of BioIndexing, which are implemented in BLASTGRES: intrinsic indexing — indexable bioinformatics datatypes — and extrinsic indexing — the ability to join information from

external resources. A sequence location datatype can relate these two aspects. Sequence locations pervade existing biological information, and define a central datatype. They are essential to the sequence alignments produced by tools like BLAST, which today provides one of the primary indexing mechanisms for this information. Furthermore, these aspects of BioIndexing are naturally combined within a modern database system, giving a flexible infrastructure for connecting and managing biological information.

BLASTGRES is an implementation of BioIndexing that provides these capabilities. We elected to develop BLASTGRES as an extension of the PostgreSQL database system for several reasons, including in particular that PostgreSQL: 1) is an open source software package with high performance and stability; 2) facilitates the introduction of user-defined datatypes; 3) provides GiST (Generalized Search Tree) indexing [8]. GiST indexing is particularly important for the location datatype because BLASTGRES needs to support query predicates other than equality and range test.

We argue that the location datatype and integrated BLAST functionality provide basic indexing infrastructure for large-scale biological databases that can be implemented with modern database systems. In the following sections, we will focus on the significance of extrinsic and intrinsic BioIndexing mechanisms and their implementation in BLASTGRES.

2 Extrinsic Indexing — BLAST Support in BLASTGRES

Data integration has always been considered one of the most important issues in bioinformatics database systems [13], since an enormous variety of bioinformatics resources and information are available worldwide. These resources are highly heterogeneous, distributed and disseminated in different formats, such as relational tables, XML files, standard sequence format, proprietary formats, etc. In order to have a unified view of biological knowledge, data integration is essential in bioinformatics databases.

However, the real power of integrating diverse bioinformatics resources is the ability to uncover new, unexpected insights [26]. Data integration alone does not achieve this goal. In addition to data integration, the ability to connect information from different data sources provides the key mechanism to infer possible characteristics to unknown biological entities. Bioinformatics practitioners often take a navigational approach to new biological discoveries, that is, they infer possible characteristics of the subject in their study from the ones of similar and related subjects. The ability to connect information makes this automatic inference and knowledge discovery possible.

2.1 BLAST Joins

Even though some people might disagree [22, 23] with David Wake’s words that “homology is the central concept for ALL of biology” [24, 25], there is no doubt that similarity search is one of the most heavily-used tools in computational and

comparative biology [21]. Among available similarity searching tools, BLAST is currently most popular. It is a set of local alignment algorithms that identify local regions of maximal similarity between sequences, providing a way in relating features of known sequences to those of unknown ones.

BLAST uses a heuristic algorithm to detect sequence similarity and is optimized for speed. It is suitable for large-scale analyses. Another advantage of BLAST is its ability to compare a sequence in question with the most up-to-date databases available. Hence, it is an indispensable component in bioinformatics computing environment, and the ability to integrate BLAST results with other biological information is essential.

BLAST can be viewed as a kind of index; given a sequence of interest, BLAST finds related sequences. Many biological relationships can be represented and related by joining BLAST results. We call BLAST a form of extrinsic indexing because it is a function which serves as an index mechanism but does not require local storage for index structures.

2.2 Integrating BLAST with a Database

Today the application of BLAST results in relating biological sequences is often ad hoc. Special codes are needed to interact with a BLAST server, parse BLAST results and integrate the results with other biological data. In addition, BLAST provides only a limited set of controls over its result; advanced filtering and query mechanisms are sometimes required. For instance, currently BLAST results are sorted by E-values and result filtering is limited to a small set of attributes. Given that one of the most commonly performed tasks in sequence analysis is sequence similarity search, and BLAST is not only the most heavily-used tool for this purpose but also lacks query mechanisms, integration of BLAST with database systems is vital.

Several commercial database systems have been extended in a more principled way with support for BLAST. In *Oracle 10g* [11], similarity search can be performed against sequences stored in a local database through a series of BLAST functions. In IBM's *Information Integrator* [9], BLAST queries are sent to remote BLAST servers and BLAST results are imported through table functions. Advanced controls and features are needed to make sense of BLAST results, such as the annotational information for the sequences and clustering of the BLAST results.

Large-scale bioinformatics data analysis requires the integration of BLAST for automatic inferences of relationships between sequences. The current BLAST tool represents a highly-tuned heuristic index on biological information. A great deal of human intervention can be required to annotate the sequences and filter out unwanted results. The support of BLAST in database systems not only eases the process of data integration between different heterogeneous bioinformatics data sources, but also provides advanced query abilities that were not provided in traditional BLAST tools.

2.3 BLAST Integration in BLASTgres

In BLASTGRES, BLAST support is achieved by defining a set of user-defined functions that return BLAST results as a table. Datatypes are transformed automatically in this process. For example, location information (identifier, start and end point) for both the query and subject sequences are included in the results generated by BLAST. The `blast()` function in BLASTGRES automatically transforms this information into two corresponding location datatype values. Annotational information, such as species, and description for a particular sequences can also be added automatically through invoking the proper function calls.

Currently, only a limited number of filtering conditions are supported in BLAST results such as accepting only matches with certain range of alignment length or percent identity. With BLAST support within SQL queries, complex constraints on the results can be imposed via WHERE-clause conditions. BLAST results can also be viewed and analyzed by sorting or grouping on any attribute.

From a database user's perspective, there does not need to be any difference between the a BLAST hit table and a normal table, except that the former is treated as a parameterized table. In this way, BLAST results can be easily integrated with other biological information in the system. Besides, annotational data can be attached to BLAST results automatically in order to provide detailed knowledge about these sequences. Thus, since the BLAST parameterized table is a surrogate of the BLAST server, it accepts all parameters accepted by the BLAST server. Some typical examples are:

```
-- blast a given sequence with BLAST default parameters
SELECT * FROM blast('acttgatg');
-- blast the 2nd exon of the Mus musculus H2-DMB1 gene
SELECT * FROM blast('NM_010387.2[265..546]');
-- blast a given sequence from a sequence file
SELECT * FROM blast('lab_est.fasta');
-- blast all the sequences in a given table.
SELECT * FROM blast('lab_ESTs', 'seq');

-- number of hits from different species
SELECT COUNT(*), species FROM annotated_blast( 'AF101044', 'blastn' )
GROUP BY species;

-- display hits from the same species together
SELECT * FROM annotated_blast( 'AF101044', 'blastn' ) ORDER BY species;

-- retrieve sequences that code for SNRPN proteins
-- in species other than homo sapiens
SELECT * FROM annotated_blast( 'AF101044', 'blastn' )
WHERE descriptions LIKE '%SNRPN%' AND species <> 'Homo sapiens';
```

Since BLAST only tells us the relationship between pairs of sequence segments, the automation and integration of BLAST with other data sources provides more information for the investigations of the sequences of interest and inference of features of unknown sequences.

Moreover, advanced query mechanisms can also be provided through this type of integration. For instance, the following query retrieves all sequences similar to

the mRNA of the Homo sapiens SNRPN upstream reading frame protein with E-value less than 1E-5 and bit score greater than 800:

```
SELECT subject_location, length FROM blast('AF101044')
WHERE evalue < 1E-5 AND bitscore > 800;
```

Additional BLAST parameters can be specified by the parameter table. According to the BLAST specification, around 50 parameters can be specified. A parameter table is just a normal table with two fields: one for the parameter name and another for the parameter value. This can be very useful. For instance, it can be enlightening for BLAST beginners to experiment with changes in BLAST parameter values, and observe their effect on results without the trouble of going through a series of web interactions. The following example shows the SQL command for locating differences in BLAST results with different parameter settings:

```
-- create tables to hold parameter settings
CREATE TABLE parameter1 ( name TEXT, value TEXT );
CREATE TABLE parameter2 ( name TEXT, value TEXT );

-- change the default settings
INSERT INTO parameter1 VALUES ( 'WORD_SIZE', '7' );
INSERT INTO parameter1 VALUES ( 'GAPCOSTS', '5 2' );
INSERT INTO parameter2 VALUES ( 'WORD_SIZE', '13' );
INSERT INTO parameter2 VALUES ( 'GAPCOSTS', '3 1' );

CREATE TABLE result1 AS
SELECT subject_location, length FROM blast_p( 'AF101044', 'parameter1' );

-- retrieve the matches that are not found in both results
SELECT R1.*, R2.*
FROM result1 AS R1, blast_p( 'AF101044', 'parameter2' ) AS R2
WHERE (R1.subject_location <> R2.subject_location)
OR ((R1.subject_location = R2.subject_location) AND (R1.length <> R2.length))
```

3 Intrinsic Indexing — the Location Datatype in BLASTGRES

We call the location datatype and related access methods a form of *intrinsic indexing*, because it fits the canonical definition of index in traditional database systems.

3.1 Locations — an Indexable Bioinformatics Datatype

The notion of a sequence location has become one of the building blocks of biological information. For instance, BIOPERL [1] defines a hierarchy of location-related objects to support location operations. The notion of location and range is also present in NCBI data model and its programming toolkits.

The location concept has complex properties, and can be difficult for users to deal with without appropriate support. For instance, locations are sometimes developed relative to different reference coordinate systems, and several location objects might reference the same sequence segments with different coordinate

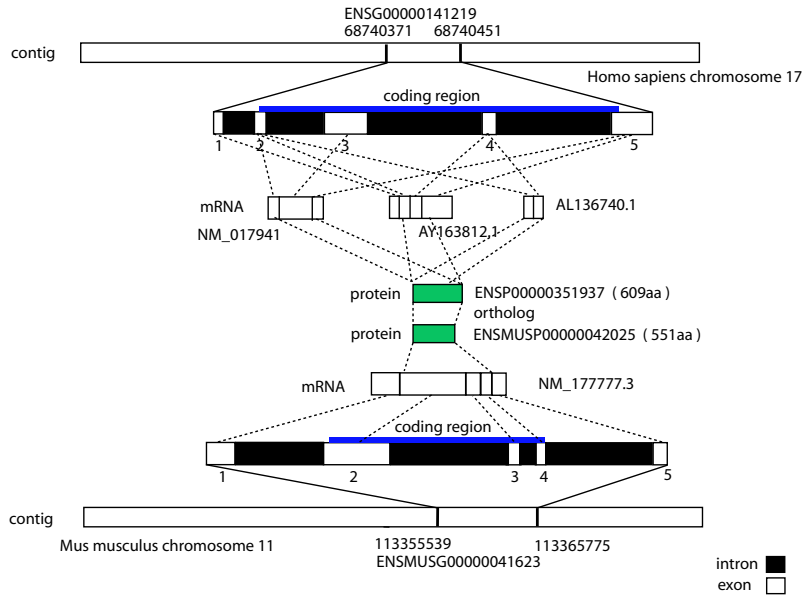


Fig. 2. Diverse relationships exist between biological information. These relationships could be maps (annotational data or features associated with a location), or relationships between locations (e.g., orthology or paralogy). This figure highlights the importance of the location datatype in bioinformatics.

systems. Without proper abstractions, users have to implement their own codes to handle these location operations and translate among them.

Figure 2 shows the complexity of various relationships among sequence segments and their annotation information. In this figure, we know that the HLC-8 (lung cancer-related protein 8) gene could produce several possible mRNA isoforms, which in turn code for this protein. HLC-8 is also orthologous to another gene (ensembl id: ENSMUSG00000041623) in *Mus musculus*. Annotational information, such as coding region, gene structure, exon components in various mRNA isoforms, orthologous relationships) are all attached to sequence segments. These sequence segments can be represented with the notion of a *location*. A location consists of a sequence identifier and an interval range. Identifiers are character strings or accession numbers used to denote a particular sequence, and the interval range consists of a pair of positive integers used to denote the sub-range within the given sequence.

3.2 Challenges in Implementing Locations

Relational databases are generally unequipped to support locations as an abstraction, and permit more powerful querying of locations. Ad hoc representations of locations result, and a variety of potential difficulties arise often in practice:

1. *Inconsistent location representations causes difficulties in data sharing and exchange.*

Without a unified representation for the location and range datatypes, different representation schemes might be employed by different people. This imposes difficulties in data exchange and sharing. For instance, it is common to denote a location interval by a pair of integers (a, b) , specifying the range of position $[a, \dots, b]$ in a sequence, but it is also common to use a pair of integers (a, b) to denote a starting location a and an interval length b . In addition, integers used to represent intervals can be 0-based or 1-based, and the integers might define an open interval or a closed interval. Inconsistencies in representations complicate automatic information exchange and can render the information unusable.

2. *High complexity in implementing location operations*

Without direct support from database systems, users are required to implement the location operations themselves, which is error-prone and often leads to inconsistencies in operational definition. SQL queries about locations expand into lengthy and complex WHERE clauses. An interesting example arises when researchers working on EST sequences need to group together consecutive overlapping EST fragments. Without direct support of a location datatype they might have to write SQL queries such as the following, which attempts to ‘coalesce’ any set of consecutive overlapping locations into a single location:

```
SELECT DISTINCT A.id, A.lower, B.upper
FROM ESTs AS A, ESTs AS B
WHERE A.unigene_clusterid = B.unigene_clusterid
AND A.lower < B.upper
AND NOT EXISTS
  (SELECT *
   FROM ESTs AS C
   WHERE C.unigene_clusterid = A.unigene_clusterid
   AND A.lower < C.lower AND C.lower < B.upper
   AND NOT EXISTS
     (SELECT * FROM ESTs AS D
      WHERE D.unigene_clusterid = A.unigene_clusterid
      AND D.lower < C.lower AND C.lower <= D.upper))
AND NOT EXISTS
  (SELECT *
   FROM ESTs AS E
   WHERE E.unigene_clusterid = A.unigene_clusterid
   AND ((E.lower < A.lower AND A.lower <= E.upper) OR
        (E.lower < B.upper AND B.upper < E.upper)))
```

This example demonstrates how complex it can be for users to implement location operations, and shows how abstraction for location is important in biological databases. (Similar problems arise in temporal database [12].) High-level algorithms are hard to express without the location datatype and its associated functions.

3. *Performance penalties*

Query processors are sometimes incapable of generating efficient execution plans for complex conjunctive (or disjunctive) normal forms and handling in-

equality relationship between inter-dependent attributes. For instance, Typical query optimizers in relational database systems do not gracefully handle the kinds of `WHERE` clauses needed, which include clauses like

```
subject.start > query.start AND subject.start < query.end
```

This example seeks to locate subject intervals that are entirely contained in the query interval. Traditional database join algorithms have concentrated on single-attribute equijoins, while most predicates for interval join must deal with inequality on two interdependent attributes. Without explicit support for the location datatype and location indexing, location joins can incur significant performance penalties.

A typical and straightforward representation of a location would be a sequence identifier as a character string and the location interval as a $(start, end)$ pair of integers, but there are many other possible representations, such as using integer codes for sequence identifiers and/or a $(start, length)$ interval representation.

Internally, a location is represented as an identifier for the sequence, an integer interval $[lower, upper]$, and the strand in which this sequence resides (only for DNA sequences and is typically denoted as '+' or '-'). The interval range is represented as a closed interval with positions starting at 1, following the convention of most biological databases.

Sequence identifiers are typically implemented as character strings. Often no restrictions are imposed on these strings than that they uniquely specify a particular sequence. Different identifier formats have been adopted by major databases, such as NCBI and Ensembl, and arbitrary strings are often used in local database for identification purposes.

3.3 The Location Datatype in BLASTgres

Coordinate transformation and slicing operations are also supported. New reference coordinates can be specified by defining aliases to existing locations and slicing can be specified to denote a sub-range within a location. The following SQL session demonstrates these features:

```
CREATE TABLE features ( location loc, description text);

-- The Prader-Willi/Angelman syndrome region on chromosome 15
INSERT INTO features VALUES ( 'NG_002690[1..755217]', 'Prader-Willi/Angelman syndrome region' );
INSERT INTO features VALUES ( 'NG_002690[1..174707]', 'AC090602.16' );
INSERT INTO features VALUES ( 'NG_002690[174707..324834]', 'AC124312.5' );
INSERT INTO features VALUES ( 'NG_002690[324835..478258]', 'AC124303.5' );
INSERT INTO features VALUES ( 'NG_002690[478259..606120]', 'AC100774.2' );
INSERT INTO features VALUES ( 'NG_002690[606121..755217]', 'AC124997.4' );

-- create an alias for denoting SNRPN gene
-- slicing operation is used for specifying the sub range.
SELECT alias( 'SNRPN', 'NG_002690[1..755217][42737..638553]' );

-- annotate the exon feature to the location SNRPN[113115..113261]
-- which is equivalent to NG_002690[155891..155997]
INSERT INTO features VALUES ( 'SNRPN[113115..113261]', '10th exon' );
```

Essential operations and functions for the location datatype are also supported. A major proportion of these functions are related to interval operations. More than 30 interval operations are defined, including Allen's interval logic [15] (which includes after, before, contains, during, equals, overlaps, overlapped_by, finishes, finished_by, meets, met_by, starts and started_by). Optimization information (such as regarding ordering, commutativity or negation) is also provided to permit optimization of important operations like merge-join, hash-join or general theta-join.

Below is a simple example to demonstrate the power of location datatype support. This example shows a session that (painfully) attempted to locate alternatively spliced exon intervals which intersect with known homology intervals and associate them with known protein features from the Pfam and Swissprot databases.

```
CREATE TABLE alt_splice_homology_map AS
SELECT o.*, d.swiss_id, d.query_start, d.query_end,
       d.hit_start+(o.seq_start-d.query_start)/3,
       d.hit_start+(o.seq_end-d.query_start)/3,
FROM alt_splice_exon_obs o, alt_splice_homology d
WHERE   o.ug_id = d.ug_id
        AND o.seq_start > d.query_start
        AND o.seq_start < d.query_end
        AND d.e_value < 0.01
GROUP BY o.ug_id, o.seq_start;

SELECT o.*, f.type, f.start, f.end
FROM alt_splice_homology_map o, swiss_feature f
WHERE   o.swiss_id=f.swiss_id
        AND o.hit_end >= f.start
        AND o.hit_end <= f.end;
```

The same queries can be expressed in a more intuitive way with the location datatype:

```
CREATE TABLE alt_splice_homology_map AS
SELECT o.*, d.location,
       range_start(d.query)+(o.location-range_start(d.hit))/3
FROM alt_splice_exon_obs o, alt_splice_homology d
WHERE   o.location @ d.location -- contained
        AND d.e_value < 0.01
GROUP BY o

SELECT o.*, f.type, f.location
FROM alt_splice_homology_map o, swiss_feature f
WHERE   o.location &< f.location -- left overlap
```

When a new sequence identifier is encountered by BLASTgres, it is assigned a unique number. These identifiers are stored in a name table (system-wide symbol table) to avoid expensive string comparisons in index lookup and efficient storage management. A benefit of the name table is that three numbers can encode any location — potentially saving significant amounts of storage — and comparisons between two locations (including index tree traversals) can be implemented by integer comparisons, replacing more expensive string comparisons. Another advantage of using the name table is to provide mappings between different namespaces. Since different identifiers are assigned to the same genomic sequence by different databases, the name table can incorporate the external

knowledge of namespace mappings and assign the same number to sequence identifiers that denote the same sequence.

3.4 Location Indexing in BLASTgres

The introduction of location datatype not only provides a natural and intuitive way to represent biological information, but also boosts system performance. Additional performance increase could be achieved by supporting the location index scheme. Supports for indexing schemes in traditional relational database systems are very limited and inflexible. They are only limited to a few well-known index structures, such as B+-tree, Hash and R-tree and could be used for a limited set of native datatypes for (in)equality and range queries.

Supporting location indexing implies support for interval indexing as well. Interval indexing is not supported in traditional database systems and standard join operations could not handle intervals efficiently; therefore, interval indexing has been the focus of several research studies. Previous research spans several decades, from Edelsbrunner's dynamic rectangle intersection searching [14], to the RI (relational interval) tree [7] recently developed to support efficient interval indexing and join operations.

On the other hand, GiST provides an elegant way to solve this problem. GiST is extensible in both datatypes that it can index and query predicates that it supports. It is a balanced search tree in which search keys are maintained in a hierarchical manner. The search keys may in fact be any arbitrary predicate; by convention, this predicate must hold true for each datum below the key. With a given search key, GiST search will in principle traverse the entire tree in a depth-first search manner. If the query predicate is consistent with the search key (holds true), GiST search continues with the subtree below this key. Otherwise, the subtree is ignored in the traversal process.

We implemented our location indexing under GiST architecture. Each search key contains two pairs of integers: $[id_lower, id_upper]$ represents the range which the identifier integers in the keys of the subtrees fall into and $[lower, upper]$ the range which the intervals in the keys of the subtrees fall into. In other words, they are the minimal bounding intervals that cover the range of identifier integers and intervals in the subtrees. Common interval predicates, such as Left, Right, OverLeft, Overlaps, OverRight, Contains, Contained, and Equal, are supported in our GiST index implementation. Figure 3 shows an exemplary GiST index tree for location datatype. The shaded nodes are nodes that GiST must traverse in order to answer the query.

Six methods are required to implement the semantics of the GiST index (we only include 4 methods here since the other two methods involve in compression and we do not use such a scheme). Their semantics and implementation are as follows (we take the Contains predicate as an example):

1. *Consistent(E, q)*: For any given query predicate q and an index key entry $E = (p, ptr)$ where p is the predicate and ptr is the pointer to its subtree, determine whether q is guaranteed to be satisfied by p or not. This determines

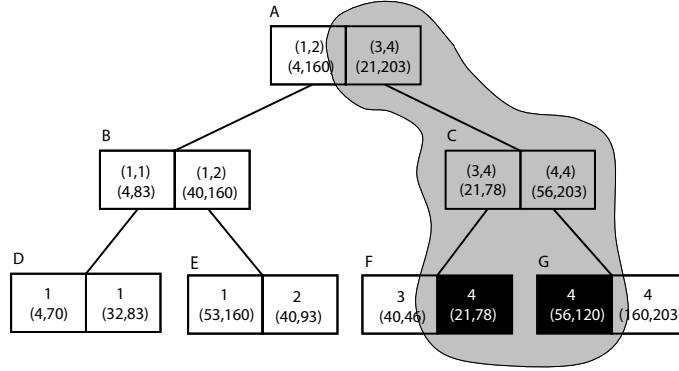


Fig. 3. An exemplary GiST index tree. Search keys are represented as two bounding intervals (for leaf nodes, the lower bound is equal to upper bound for the first bounding interval). The bounding interval of an internal node contains those subnodes. To search for intervals that have identifier number equal to 4 and contain the interval $[60, 70]$, the search algorithm would explore the shaded subindex, reaching the second entry in F and the first entry in G.

whether the search process should continue to traverse the subtree indicated by ptr or not. For the Contains predicate:

$$\begin{aligned} & ((q.identifier \geq p.id_lower) \text{ AND } (q.identifier \leq p.id_upper)) \\ \text{AND } & ((q.upper \geq p.upper) \text{ and } (q.lower \leq p.upper)) \text{ OR} \\ & (q.upper \leq p.upper) \text{ and } (q.lower \geq p.upper) \end{aligned}$$

If the index entry is an external node, we have to make sure that the identifier integer for p and q are the same and their intervals actually overlap.

$$\begin{aligned} & (q.identifier == p.id_lower) \text{ AND} \\ & (q.upper \leq p.upper) \text{ AND } (q.lower \geq p.lower) \end{aligned}$$

2. *Union(P)*: For a given set P of index entries $(p_1, ptr_1), \dots, (p_n, ptr_n)$, construct a new index node that holds for all index entries in the set. In our implementation, the minimal bounding intervals for identifiers and interval ranges are maintained by

$$\begin{aligned} new\ id_lower &= \min(p_1.id_lower, \dots, p_n.id_lower) \\ new\ id_upper &= \max(p_1.id_upper, \dots, p_n.id_upper) \\ new\ lower &= \min(p_1.lower, \dots, p_n.lower) \\ new\ upper &= \max(p_1.upper, \dots, p_n.upper) \end{aligned}$$

3. *Penalty(E_1, E_2)*: Calculate a domain-specific penalty for inserting E_2 into the subtree of E_1 . GiST always inserts an entry into the subtree with least penalty. In our implementation, the penalty metric is determined by the increase in size of minimal bounding intervals from expanding E_1 to $Union(E_1, E_2)$.
4. *PickSplit(P)*: Whenever a node in the GiST index structure is full, this method is invoked to split the set P of entries into two sets of entries. In our system, we sort the entries by the order of identifier interval and range interval and evenly divide the set into two halves.

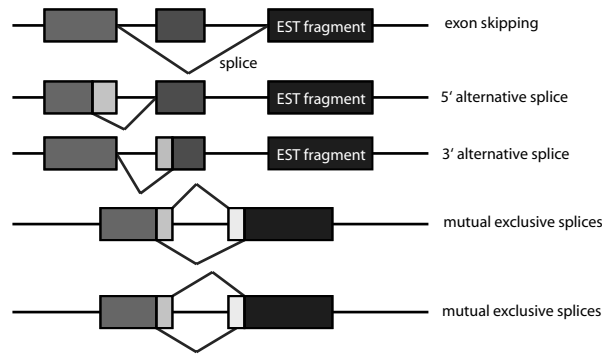


Fig. 4. Different types of alternative splicing events

4 Sample Application in Alternative Splicing

Alternative splicing has become one of the central foci in bioinformatics research since it contributes a great deal to the variety in transcriptomes and proteomes. It is variously estimated that between 20% to 80% of human genes are subject to alternative splicing [17, 19]. In this section, we illustrate the use of BLASTGRES to detect and catalog alternative splicing events. We adapted the dataset used in the HASDB database [16], which captures transcription and splicing relationships between ESTs and human genes. This dataset was created by mapping human cDNA and EST sequences onto human contigs in order to study alternative splicing. The following table definition was adapted from HASDB for BLASTGRES:

```
CREATE TABLE splice (
  splice_no      INTEGER UNIQUE NOT NULL,
  splice        loc,
  five_prime_site CHAR(20),
  three_prime_site CHAR(20));
```

The splice table contains non-redundant set of splices; a splice is a segment of genomic sequence which is flanked by locations observed in the EST evidence. By locating genomic fragments that cannot be aligned with ESTs, intron (splice) structures can be inferred, and these in turn can be used to infer exon structures.

Figure 4 shows different types of alternative splicing events. For illustrative alternative splicing queries, it is sufficient to consider the splice table. The following queries detect different alternative splicing events.

```
-- look for alternative 5' splice
SELECT A.splice_no, B.splice_no FROM splice A, splice B
WHERE A.splice_no <> B.splice_no AND loc_finishes(B.splice, A.splice);
-- intervals with the same end-point

-- look for alternative 3' splice
SELECT A.splice_no, B.splice_no FROM splice A, splice B
WHERE A.splice_no <> B.splice_no AND range_starts(A.splice, B.splice);
```

```

-- intervals with the same starting point

-- look for all pairs of splices for potential exon skip events
CREATE TABLE potential_exon_skip AS
SELECT A.splice_no AS splice1, B.splice_no AS splice2, C.splice_no AS splice3,
       C.splice AS splice
FROM splice A, splice B, splice C
WHERE A.splice_no <> C.splice_no AND B.splice_no <> C.splice_no
      AND NOT ( A.splice && B.splice )
      AND C.splice = ( A.splice | B.splice );
-- C's interval is the union of A's and B's interval

-- find all pairs in potential_exon_skip that have no intervening splice
CREATE TABLE exon_skip AS
SELECT DISTINCT A.splice1, A.splice2, A.splice3, A.splice
FROM potential_exon_skip A
WHERE NOT EXISTS (SELECT * FROM splice B
                  WHERE B.splice_no <> A.splice1
                    AND B.splice_no <> A.splice2
                    AND B.splice ^ A.splice -- B contained in the big range
                  );

-- look for splice pairs that overlap but have different starting and ending points
CREATE TABLE potential_mutually_exclusive AS
SELECT A.splice_no AS splice_no1, A.splice AS splice1,
       B.splice_no AS splice_no2, B.splice AS splice2,
       loc_minmax(A.splice,B.splice) as union_loc
FROM splice A, splice B
WHERE A.splice_no < B.splice_no AND A.splice && B.splice;

-- look for all the splice pairs in potential_mutually_exclusive which do not contain
-- another splice within the interval range they span. This formulation assumes
-- that the difference in length of two mutually exclusive splices is less than 100.
CREATE TABLE mutually_exclusive AS
SELECT DISTINCT A.splice1, A.splice2
FROM potential_mutually_exclusive A
WHERE ABS(loc_size(A.splice1)-loc_size(A.splice2))<100
      AND NOT EXISTS
      (SELECT * FROM splice B
       WHERE B.splice_no <> A.splice_no1
         AND B.splice_no <> A.splice_no2 AND loc_inside(B.splice, A.union_loc));
-- B's interval is completely inside A.union_loc's interval

```

5 Conclusion

In this paper, we introduced BioIndexing, a conceptual infrastructure connecting and querying biological information. This information is difficult to represent and manage, and its scale can present significant performance challenges. BioIndexing gives a practical model that modern database systems can follow for large-scale biological databases.

The word ‘index’ has been used in diverse ways. However, the term carries a common emphasis on providing a mapping between information entities. For instance, in the database field, an index is an auxiliary structure which speeds up the data retrieval by providing a mapping between a record key and the physical disk address of the records containing the key. BioIndexing provides similar functionality as a database index but also facilitates data integration.

The location concept is fundamental to BioIndexing, just as it is to bioinformatics. Biological features are generally attached to locations and locations

are also the bases for maps (associations of features with a sequence segment), alignments (relationships between two genomic sequence segments) and other complex relationships. Naive representation of the location concept, without formal development of a datatype, is error-prone and easily leads to inconsistencies in operational definition. Moreover, query efficiency also can be compromised due to the complexity of query criteria and inter-dependency between related attributes. Because of the vast scale of biological information, effective indexing schemes for the location datatype are also required.

BioIndexing includes intrinsic and extrinsic indexing. Intrinsic indexing concerns the indexability of bioinformatics datatypes, permitting both the representation and management of biological mappings. Extrinsic indexing concerns the functions and algorithms used to access and connect this information, even when it is not stored locally. Sequence alignment tools like BLAST give a popular form of indexing for sequences, and comprise a basic scheme for extrinsic indexing.

We also presented BLASTGRES, an extension to PostgreSQL database system that provides infrastructure in support of BioIndexing. Specifically, BLASTGRES provides a sequence location datatype and ‘BLAST joins’ — integration with the sequence indexing provided by BLAST. This combination permits powerful database query mechanisms to be utilized for BLAST automation and integration with other annotational data. With the integration of BLAST, proper abstraction of biological data, and fast indexing schemes, BLASTGRES is an illustration of the usefulness of the BioIndexing concept.

6 Acknowledgement

This work was funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 RR021813. Information on the National Centers for Biomedical Computing can be obtained from <http://nihroadmap.nih.gov/bioinformatics>.

References

1. Stajich, J. E., Block, D., Boulez, K., Brenner, S.E., Chertiz, S. A., Dagdigian, C., Fuellen, G., Gilbert, J.G.R., Korf, I., Lapp, H., Lehvaslaiho, H., Matsalla, C., Mungall, C. J., Osborne, B. I., Pocock, M. R., Schattner, P., Senger, M., Stein, L.D., Stupka, E.D., Wilkinson, M., Birney, E. The Bioperl Toolkit: Perl Modules for the Life Sciences. *Genome Research*, 1611–1618, 2002.
2. Altschul, S. F., Gish W., Miller W., Myers E. W., Lipman D. J. Basic local alignment search tool. *Journal of Molecular Biology*, 215, 403–410, 1990
3. Altschul, S.F., Madden, T.L., Schffer, A. A., Zhang, J., Zhang, E., Miller, W. Gapped BLAST and PSI-BLAST: a New Generation of Protein Database Search Program. *Nucleic Acids Research*, 3389–3402, 1997. <http://www.ncbi.nlm.nih.gov/BLAST/>.
4. Korf, I., Yandell, M., Bedell, J. BLAST, O’Reilly Publishers, 2003.

5. Stonebraker, M., Kemnitz, G. The Postgres Next-Generation Database Management System. *Communications of the ACM* **34:10**. 78–92, 1991. <http://www.postgresql.org/>
6. Carey, M., Hass, L. M., Schwarz, P. M., Arya, M., Cody, W. F., Fagin, R., Flickner, M., Luniewski, A. W., Niblack, W., Petkovic, D., Thomas, J., Williams, J. H., Wimmers, E. L. Towards Heterogeneous Multimedia Information Systems: The Garlic Approach. *Proceedings of the 5th International Workshop on Research Issues in Data Engineering - Distributed Object Management*. 124–131, 1995.
7. Enderle, J., Hampel, M., Seidl, T. Joining Interval Data in Relational Databases *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, 683 – 694, 2004.
8. Hellerstein, J. M., Naughton, J. F., Pfeffer A. Generalized Search Trees for Database Systems. *Proceedings of 21th International Conference on Very Large Data Bases*, 562–573, 1995.
9. Eckman, B., Prete, D. D. Efficient Access to BLAST using IBM DB2 Information Integrator, *IBM Healthcare and Life Sciences*, 2004.
10. NCBI Taxonomy BLAST help. <http://www.ncbi.nlm.nih.gov/blast/taxblasthelp.shtml>
11. Stephens, S. M., Chen, J. Y., Davidson, M. G., Thomas, S., Trute, B. M. Oracle Database 10g: a platform for BLAST search and Regular Expression pattern matching in life sciences. *Nucleic Acids Research*, **33**, 675–679, 2005.
12. Zaniolo, C., Ceri, S. Faloutsos, C., Snodgrass, R. T., Subrahmanian, V. S., Zicari, R. *Advanced Database Systems*, Chapter 5, Morgan Kaufmann Publishers, Inc, 1997.
13. Davidson, S.B., Overton, C., Buneman, P. Challenges in Integrating Biological Data Sources. *Journal of Computational Biology*, **4**, 557–572, 1995.
14. H. Edelsbrunner Dynamic Rectangle Intersection Searching. *Institute for Information Processing*, Report 47, Technical University of Graz, Austria, 1980.
15. Allen, J. F. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM* **26:11** 832–843, 1983.
16. Lee, C., Atanelov, L., Modrek, B., Xing, Y. ASAP: The Alternative Splicing Annotation Project. *Nucleic Acids Research*, **31**, 101–105, 2003.
17. Modrek, B., Lee, C. A Genomic View of Alternative Splicing. *Nature Genetics* **30**: 13–19. 2002
18. Sorek, R., Shamir, R., Ast, G. How Prevalent Is Functional Alternative Splicing in Human Genome? *TRENDS in Genetics*, **20**, 2, 68–71, 2004.
19. Lopez, A.J. Alternative splicing of pre-mRNA: developmental consequences and mechanisms of regulation. *Annual Review of Genetics*, **32**, 279–305, 1998
20. Thanaraj, T. A., Stamm, S., Clark, F., Riethoven, J. J. M., Texier, V. L. ASD: the Alternative Splicing Database, *Nucleic Acids Research*, **32**, 64–69, 2004.
21. Pertsemidis, A., Fondon J. W. III Having a BLAST with bioinformatics (and avoiding BLASTphemy). *Genome Biology*, **2(10):reviews2002.1-2002.10**, 2001
22. Wake, D. B. Comparative termininlogy. *Science*, **265**, 268–269, 1994
23. Wake, D. B. Homoplasy, homology and the problem of sameness in biology. *Novartis Found Symposium*, **222**, 24–33, 1999
24. Wells, J., Nelson, P. Homology: A concept in crisis. *Origins and Design*, **18(2)**. pg 15, 1997.
25. Laubichler, M. D. Homology in Development and the Development of the Homology Concept *American Zoologist*, **40(5)**, 777–788, 2000.
26. Lee, C., Irizarry, K. The GeneMine System for Genome/proteome Annotation and Collaborative Data Mining, *IBM Systems Journal*, **40(2)**, 2001.