An Extension to SQL92 for Biological Databases

Wenguo Liu, Daniel P. Miranker
{liuwg, miranker}@cs.utexas.edu

Abstract

MoBIoS is a project that aims at inventing a new generation database management system (DBMS) for molecular biological data. This DBMS has a set of features including storage of sequences, retrieval of sequences based on similarity metric, and a query language (mSQL) embodying the semantics of Genomics and Proteomics and allowing for concise expression of Bioinformatics studies.
This paper focuses on designing and implementing mSQL. mSQL is based on the standard relational operators (σ, π, *join*, *union*), with extensions in operators to enable splitting sequences into fragments, merging overlapped fragments, grouping fragments that can be merged, interior fragment select, searching, and joining optimized with metric space index. Query optimization rules are given based on these operator extensions. The concept of Sequence View is introduced, which has all the merits of standard database View, and provides for the explicit fragmentation of a sequence into overlapping substrings of a fixed length and the concurrent construction of a metric-space index on the fragments in order to accelerate matching of those fragments. Three sample biological queries (Whole Genome Join, Conserved Primer Pair Discovery, and Electronic PCR) are expressed in mSQL, query plan trees based on mSQL biological operator extensions and Sequence View are discussed. Biological applications show that mSQL is a suitable and efficient declarative querying language for querying primary structure of biological data sets with the potential to be extended for querying secondary structure.

Outline

1. Introduction

2. Related Work

3. mSQL
    - 3.1 Predefined types extensions
    - 3.2 Predefined biological related functions
    - 3.3 Extensions to the standard relational operators (σ, π, *join*, *union*)
        - 3.3.1 *CreateFragments*(Δ)
        - 3.3.2 Sequence View
        - 3.3.3 Merge (∇)
        - 3.3.4 GroupFragments (γ)
        - 3.3.5 Metric Match (τ)

3.3.6 Metric Join (∞)
3.3.7 Interior Fragment Select (σ^F)
3.4 Properties of mSQL operators
3.4.1 Identity
3.4.2 High-dimensional operators
3.4.3 Equivalence rules
3.4.4 Cost model

4. mSQL Applications
4.1 Whole Genome Join
4.1.1 Problem Definition
4.1.2 Query Plan Tree
4.2 Conserved Primer Pair Discovery
4.2.1 Problem Definition
4.2.2 Query Plan Tree
4.3 Electronic PCR
4.3.1 Problem Definition
4.3.2 Query Plan Tree

5. Conclusions and Future Work

An Extension to SQL92 for Biological Databases

1. Introduction

It is well known that biological data sets are growing at exponential rate, some nucleotide and protein sequences data sets are doubling about every 16 months [3, 18]. Therefore, it is a very difficult task to manage and query this huge set of sequence data sets. Gene sequence database is a potential method to tackle this problem. There are two logic perspectives in gene sequence databases. One is the sequences-level perspective that whole gene sequences are considered as fields of tuples for identification and storage, and the other is the fragment-level perspective that index has to be built on fragments of part of the sequences of tables. Although relational databases gain their successes in commercial data management in the past several decades, when applied to biological data sets management, relational database has obvious limits. Specifically, a traditional index is built on one or more columns of a table thus cannot simultaneously represent these two perspectives of the sequences efficiently.

An important part of a gene sequence database is a query language embodying the semantics of Genomics and Proteomics and allowing for concise expression of Bioinformatics studies. SQL92 might be a good choice for expressing sequence queries with its powerful functionalities and mature query optimization techniques, however, all of the operators of standard SQL92 (for example, group operator, select operator, etc.) are based on exact match, i.e., equivalence or nonequivalence between different database elements. In gene sequence databases, similarity search is the most important searches, and indexing mechanism based on similarity between different objects is necessary for accelerating matching of objects. Therefore, standard SQL92 is not suitable for the query language of a gene sequence database.

Biological data is not random but exhibits interesting structure with respect to clustering [1]. Actually, clustering is a primary method underlying bioinformatic discovery. Therefore, it is very likely that biological data can be indexed using metric-space indexing [19]. Researches in MoBIoS group [1, 2] have verified this assumption, and metric-space indexing has been developed as an efficient indexing mechanism for accelerating the matching of gene sequences.

Definition 1: A **Metric Space** is a data object space with a total distance function *d* with the following properties [15, 16]:

(i). $d(O_x, O_y) = d(O_y, O_x)$ (symmetry)

(ii). $0 < d(O_x, O_y)$, $O_x ? O_y$, and $d(O_x, O_x) = 0$ (non-negativity)

(iii). $d(O_x, O_y) = d(O_x, O_z) + d(O_z, O_y)$ (triangle inequality)

MoBIoS [1, 2] is a project that aims at inventing a new generation database management system (DBMS) for molecular biological data. This DBMS is developed over Mckoi [23], an open source Java RDBMS, and has a set of features including storage of sequences, retrieval of sequences based on similarity metric, and a query language (mSQL) embodying the semantics of Genomics and Proteomics and allowing for concise expression of Bioinformatics studies.

This paper focuses on designing and implementing mSQL. mSQL is based on the standard relational operators ($\sigma, \pi, join, union$), with extensions to enable splitting sequences into fragments, merging overlapped fragments, grouping fragments that can be merged, searching, and joining optimized with metric space index. mSQL with sequence views concept (an extension to standard SQL view concept) provides a programmatic way to capture two different logical perspectives (sequences-level and fragment-level) of sequences.
The paper is organized as follows: In section 2, we describe related techniques and languages for querying gene sequences. In Sections 3, we study the MoBIoS query language (mSQL), the central part of MoBIoS system in detail. Several biological applications of mSQL are discussed in Section 4. Section 5 concludes this paper and points to directions for future work.

2. Related Work

Metric-space index is an important technique for improving the performance of searching on gene sequences. The key difference between answering a range query in Metric Space and in a traditional data type space is that there exists no total linear order of data objects in Metric Space that preserves the relative similarity [20]. Through the use of the triangle property of the distance function, expensive distance calculations in Metric Space can be partly saved [15]. It is a key point for a gene sequence database to have metric space indexing mechanism and have an optimization engine that can utilize underlying metric space indexes.
Hammer and Schneider [4, 5] suggest developing a specific algebra for querying biological data sets, however, this algebra is still under construction. PiQA [3] is a protein query algebra, which provides a set of algebraic operations on both the primary and secondary structure proteins. In PiQA a match operator applies regular expressions to a long sequence, each accept state is recorded as a row in table, match extension operators are used to merge rows in the query result table. However, PiQA does not provide high-dimensional operators (join, group based on multiple columns, etc.). This is understandable since PiQA aims at competing the express capability of BLAST [22], which does not provide join operations.
Ordered database can be applied in many fields, querying ordered data arises naturally in applications of finance, molecular biology, and network management. Actually, gene sequences have obvious order properties. Seshadri et al. [6, 7, 8] proposes techniques for querying sequence databases. However, the focus of these techniques is on aggregate-based analysis of sequences. Approximate matching operators are not provided, which are necessary for querying biological sequences. Lerner and Shasha [9] introduce a query language and algebra, called AQuery, that supports order by considering columns as vectors and introducing ordered query operators. AQuery with its optimization techniques, can achieve better performance in querying speed compared with SQL99 implementations. However, similar to Seshadri's sequence data model, AQuery cannot be directly applied to biological database.
Biological database spends a large amount of time in string manipulation. Grahne and Hakli [10, 11, 12, 13] have done a lot of research in string database. A declarative database query language for manipulating character strings is provided. The declarative expression are evaluated by first performing a compilation transforming them to

nondeterministic finite automata and then by simulating these automata using a depth-first search engine. It should be noted that these string manipulation techniques can only be applied to primary sequence matching without approximations. Also, the scalability of the performance of string databases using string matching techniques has not been proved, the size of the automata states could be uncontrollable.

3. mSQL

MoBIoS SQL (mSQL) exploits standard SQL object-relational syntax for querying biological datatypes. The syntax to embody the metric-space extensions is largely covered by borrowing standard spatial database extensions to SQL [14]. For example, mSQL has a similar function *distance* to compute the similarity between different objects, however, this function is always related to a metric, i.e., in mSQL, distance between objects is defined only in a metric space, it is just like in spatial database, the distance is defined only in a spatial space.
In the following sections, first we will describe the predefined biological data types, then we will describe biological related functions defined on the biological data types, new operator extensions to standard relational operations ($\sigma, \pi, join, union$) are given in the last part of this section.

3.1 Predefined types extensions

In addition to the standard SQL92 type (Numeric, Characters, etc.), some biological data types are necessary to represent the basic biological data. These types are: *Metric*, *Sequence*, and *SubSequence*.

Strictly speaking, *Metric* is not necessarily a biological term. However, it is a feasible way to relate *Metric* to the similarities between biological objects. In mSQL, in addition to built-in metrics to support sequence homology and protein identification, users may add their own metrics.

Basically, there are two important types of similarity queries in Metric Space, i.e., range query and *k*-nearest neighbor query.

Definition 2: Range query range(Q,r): find all data objects O such that $d(Q, O) = r$; ***k*-Nearest Neighbor query $NN_k(Q)$:** find k data objects such that they are the k closest ones to Q [15,16].

It should be noted that *k*-nearest neighbor query can be expressed with range query.
Sequence is a biological data type to represent the basic biological data, which consists of a series of characters that embody biological properties. For example, DNA data and protein data can be represented by Sequence.
SubSequence is a biological data type included in mSQL to represent a fragment of the Sequence. In *SubSequence*, three members (offset, length, and parent sequence pointer) with their obvious meanings are used to uniquely identify a fragment of a Sequence. *SubSequence* is also called *Fragment*.

3.2 Predefined biological related functions

mSQL needs some functions to operate on predefined biological data types. These functions fall into two groups: the first group is used to compute the similarity between different data objects, and the second group is used for the operations on the SubSequence, and connecting a SubSequence with its parent Sequence.

distance is a function used to compute the similarity between different data objects. The syntax of *distance* is *double distance*(*String metric_name, Sequence* s_1, *Sequence* s_2), where *metric_name* is used to uniquely specify a *Metric* for this distance computation on two data objects s_1 and s_2.

The second group of functions include *FRAGACCID*, *FRAGOFFSET*, *FRAGLENGTH*, *FRAGCONTENT*. *FRAGACCID* is used to get the parent sequence identifier for a SubSequence, thus linking a SubSequence with its parent. *FRAGOFFSET* returns the offset of a SubSequence in its parent sequence. *FRAGLENGTH* gets the length of a SubSequence. *FRAGCONTENT* is used to return the string content of a SubSequence.

3.3 Extensions to the standard relational operators (σ, π, *join*, *union*)

3.3.1 CreateFragments (Δ)

Δ: String × Set<Sequence> × Integer × Integer → Set<Fragment>

CreateFragments (Δ) is an operator used to break sequences into a set of fixed length fragments, also known as q-grams.

Definition 3: A **q-gram** with parameters k and m is a substring of a sequence s of the form $s_k, s_{k+1}, s_{k+2}, \ldots, s_m$.

CreateFragments operator is defined with parameters: sequences accession id, sequences, fragment length, and fragment shift size. The result of this operation is a set of fragments, each consisting of the parent sequence accession id, the position of this fragment in its parent sequence, and the length of this fragment. A sample *CreateFragments* operation is given in Figure 1.

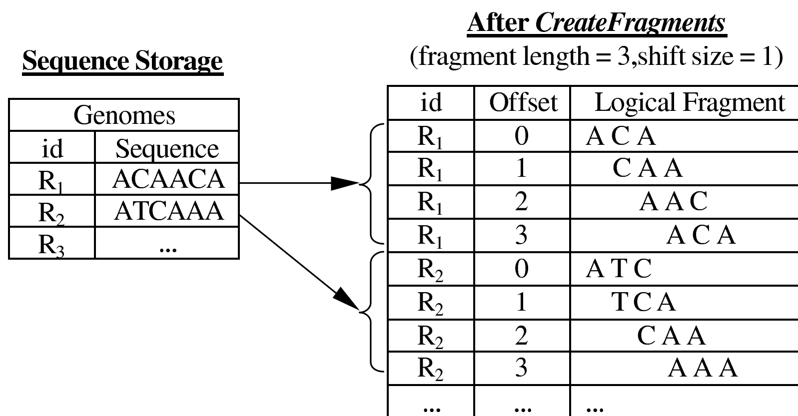


Figure 1: *CreateFragments* operation

We can see from Figure 1 that after *CreateFragments* operation with fragment length 3 and shift size 1, the sequence "ACAACA" is broken into 4 fragments, "ACA", "CAA", "AAC", and "ACA". A metric space index can be built on these fragments with a definition of sequence view, which will be introduced in the next section.

3.3.2 Sequence View

Sequence view is of course not a query operator, however, we explain the concept in this section to make it easier for the explanation of operators in the following sections.

A sequence view is a standard view definition in database except that a metric space index is built when the view is created. Sequence View has all the merits of standard database View (for example, operations on a part of a whole table is allowed with View), and also provides for the explicit fragmentation of a sequence into overlapping substrings of a fixed length and the concurrent construction of a metric-space index on the fragments in order to accelerate matching of those fragments. Figure 2 is a sample sequence view definition in mSQL.

```
CREATE SEQUENCEVIEW rice_sview(fragment) AS
        SELECT CREATEFRAGMENTS(Accid, DNA_Sequence, 3, 1)
        FROM genomes
        WHERE Organism = 'rice'
USING metric_name;
```

Figure 2: Definition of a sequence view in mSQL

A sequence view is built in three major steps:

1. Select sequences that are to be included in this sequence view through standard SELECT operation of SQL.
2. *CreateFragments* for those sequences selected in the first step.
3. Build metric space index for the fragments created from step 2.

The mechanism of sequence view is illustrated in Figure 3.

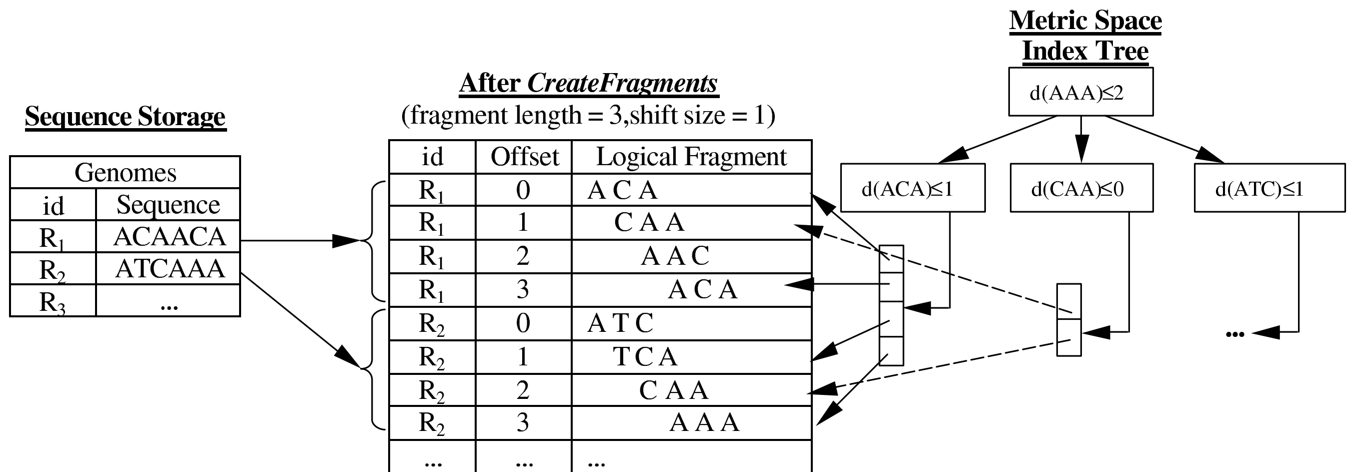


Figure 3: Sequence view mechanism

We can see from Figure 3 that a metric space index tree is built after the *CreateFragments* step. This index tree can accelerate matching of fragments. Please refer to [1, 2] for detailed information on metric space index creation, and how metric space index works.

3.3.3 Merge (∇)

∇^1: Set<Fragment> → Fragment

∇^2: Set<Fragment> × Set<Fragment> → Set<Fragment> × Set<Fragment>

Merge operator is used to merge a set of fragments into larger pieces of fragment if the fragments are *mergeable*. The definition of *mergeable* is given in definition 4.

Definition 4: A q-gram with parameters k and m is a substring of a sequence of the form $s_k, s_{k+1}, s_{k+2}, \ldots, s_m$. Two q-grams q_1 (with parameters k_1 and m_1) and q_2 (with parameters k_2 and m_2) are **mergeable** and can be merged into one longer q-gram q_3 (with parameters k_3 and m_3) iff

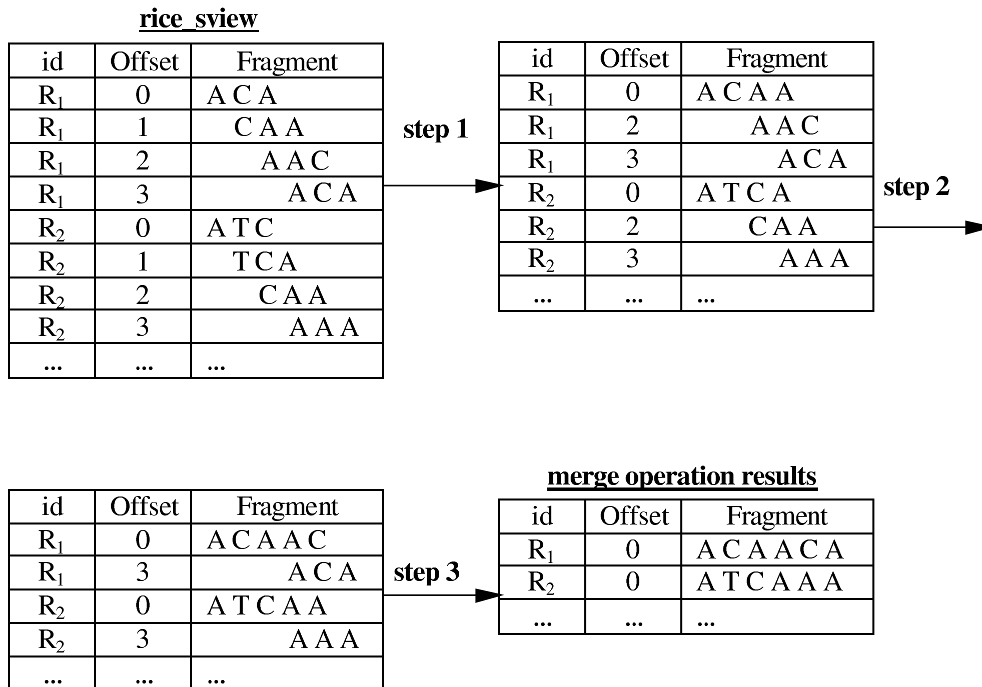
- q_1 and q_2 have the same parent sequence,
- $(k_1 \geq k_2$ and $k_1 \leq m_2)$ or $(k_2 \geq k_1$ and $k_2 \leq m_1)$,
- $k_3 = \min(k_1, k_2)$,
- $m_3 = \max(m_1, m_2)$.

The merge of two mergeable q-grams q_1 and q_2 can be expressed as $q_3 = \text{merge}(q_1, q_2)$.
A sample use of merge operator in mSQL is

```
SELECT    merge(R.fragment)
FROM      rice_sview R
;
```

Figure 4: mSQL using one-dimensional merge operator

In the above sample, merge operator combines together overlapped fragments into larger pieces of fragments. If there is only one sequence in the definition of *rice_sview* sequence view, the exectution of the above mSQL command will result in the original sequence before *CreateFragments*. The merge operation is illustrated step by step in Figure 5.



rice_sview

id	Offset	Fragment
R_1	0	A C A
R_1	1	C A A
R_1	2	A A C
R_1	3	A C A
R_2	0	A T C
R_2	1	T C A
R_2	2	C A A
R_2	3	A A A
...	...	...

step 1

id	Offset	Fragment
R_1	0	A C A A
R_1	2	A A C
R_1	3	A C A
R_2	0	A T C A
R_2	2	C A A
R_2	3	A A A
...	...	...

step 2

id	Offset	Fragment
R_1	0	A C A A C
R_1	3	A C A
R_2	0	A T C A A
R_2	3	A A A
...	...	...

step 3

merge operation results

id	Offset	Fragment
R_1	0	A C A A C A
R_2	0	A T C A A A
...	...	...

Figure 5: One-dimensional merge operation step by step

Merge operations is also applicable in two-dimensional case, although a little bit more complicated.

Definition 5: Two pair of q-grams (q_1, q_2) and (q_3, q_4) are **mergeable** and can be merged into one longer q-gram pair (q_5, q_6) iff (assume q-gram q_i has parameters k_i and m_i)

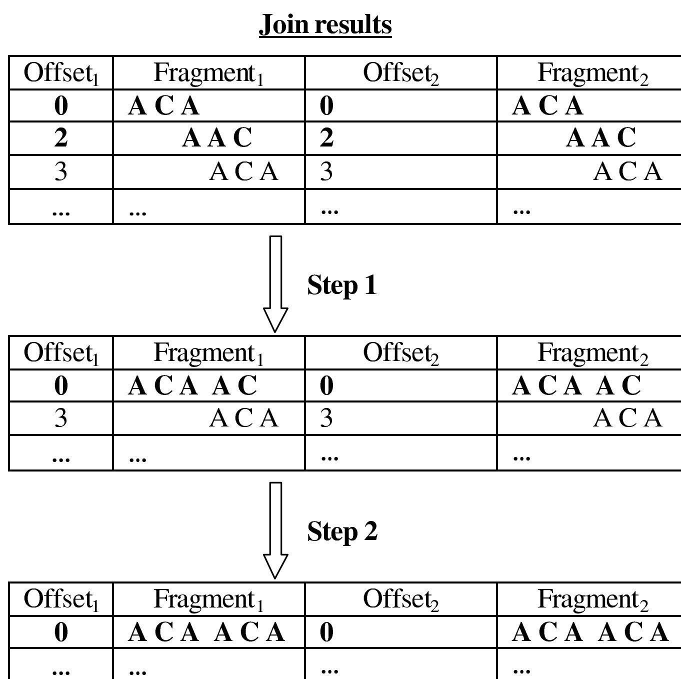
- q_1 and q_2 are of same length, q_3 and q_4 are of same length,
- q_1 and q_3 are mergeable,
- q_2 and q_4 are mergeable,
- $k_3 - k_1 = k_4 - k_2$,
- $q_5 = merge(q_1, q_3)$,
- $q_6 = merge(q_2, q_4)$,
- q_5 and q_6 are of same length.

A sample use of two-dimensional merge in mSQL is

```
SELECT      merge(R.fragment, A.fragment)
FROM        rice_sview R, arab_sview A
WHERE       distance('base_pair_mismatch', R.fragment, A.fragment) <= 0.0
;
```

Figure 6: mSQL using two-dimensional merge operator

For example, if we want to merge the results from some join operation, the merge operation actually happens in each separate sequence, but needs to take into account the offset difference. Only when the offset differences are the same can the fragments be merged. In Figure 7, for the first row and second row of the join results, the offset difference for the fragments of sequence 1 is 2, the offset difference for the fragments of sequence 2 is also 2, thus the first two fragments can be merged into one fragment for sequence 1 and sequence 2, respectively.



Join results

$Offset_1$	$Fragment_1$	$Offset_2$	$Fragment_2$
0	**A C A**	**0**	**A C A**
2	**A A C**	**2**	**A A C**
3	A C A	3	A C A
...	...	...	...

Step 1

$Offset_1$	$Fragment_1$	$Offset_2$	$Fragment_2$
0	**A C A A C**	**0**	**A C A A C**
3	A C A	3	A C A
...	...	...	...

Step 2

$Offset_1$	$Fragment_1$	$Offset_2$	$Fragment_2$
0	**A C A A C A**	**0**	**A C A A C A**
...	...	...	...

Figure 7: Two-dimensional merge step by steps

It is obvious that through merge operations, the query result size can be reduced without the loss of information. In Figure 7, the join result table changes from 3 rows into 1 row. In some sense, merge operator is a compression operator.

3.3.4 GroupFragments (γ)

γ^1: **Set<Fragment> $\rightarrow$ Set<Set<Fragment>>**

γ^2: **Set<Fragment> $\times$ Set<Fragment> $\rightarrow$Set<Set<Fragment> $\times$ Set<Fragment>>**

GroupFragments operator is used to group a set of fragments into groups, each group can be merged into exactly one larger piece of fragment. From the description of merge operator in the above section, we know that if a group of fragments can be merged into exactly one larger piece of fragment, these fragments must satisfy:

- All the fragments are from the same parent sequence.
- For any fragment in this group, there exists at least one fragment (other than this fragment) that is overlapped with this fragment.

Defintion 6: A set S of q-grams $\{q_1, q_2, ..., q_m\}$ is a **Fragments Group** iff

- S is of size 1, or
- $\forall q_i \in S, \exists q_k \in S$ $(i \neq k)$, q_i and q_k are mergeable.

A sample use of two-dimensional merge in mSQL is

```
SELECT      merge(R.fragment)
FROM        rice_sview R
WHERE       distance(metric_name, 'ACA', R.fragment) <= 0.0
GROUP BY    R.fragment
;
```

Figure 8: mSQL using one-dimensional *GroupFragments* operator

SQL92 *GROUP BY* keyword is used instead of inventing a new keyword. The query optimizer is responsible for recognizing whether it is a standard *GROUP* operation, or *GroupFragments* operation, based on the data type of the name after *GROUP BY*.

Figure 9 illustrates a one-dimensional group operation.

Query results

id	Offset	Fragment	
R_1	0	A C A	**Group 1**
R_1	1	C A A	
R_1	2	A A C	
R_1	3	A C A	
R_1	10	C A A	**Group 2**
R_1	11	A A C	
R_2	0	A T C	**Group 3**
R_2	1	T C A	
R_2	2	C A A	
R_2	3	A A A	
...	...	...	

Figure 9: One-dimensional GroupFragments operation

It is convenient to extend this one-dimensional group operator into two-dimensional or higher-dimensional operator. For example, for two-dimensional group operator, there is one requirement in addition to the requirements for one-dimensional group operator.

- The fragments offset differences must be the same for the fragments from different sequences.

This requirement is not strange at all because that is necessary for fragments to be merged. Higher dimensional group operators also need to enforce this requirement on the fragments in a group.

GroupFragments operator is closely related to *merge* operator, and they are often used together.

3.3.5 Metric Match (τ)

τ: **Set<Fragment>** × **(Fragment | String)** × **Metric** × **Double** → **Set<Fragment>**

The *Metric Match* operator searches the set of fragments to find fragments that approximately match the query object (which can be a fragment or a constant string object) based on the distance function specified by a Metric. The result of this operation is a set of fragments that match the query object. Metric match is often used to search the primary structures of DNA or protein sequences.

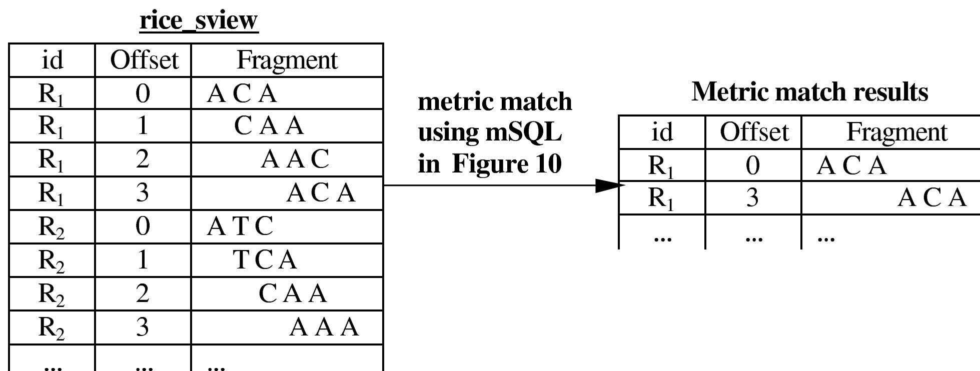
A sample use of *Metric Match* operator in mSQL is given in Figure 10.

```
SELECT      R.fragment
FROM        rice_sview R
WHERE       distance('base_pair_mismatch', 'ACA', R.fragment) <= 0.0
;
```

Figure 10: mSQL using metric match operator

The query optimizer will recognize the "distance" operator to be a form of metric match, thus can use the underlying metric space index to accelerate the searching speed if such an index exists, otherwise, an exhaustive search is executed.

Figure 11 gives an example of metric match on a predefined sequence view rice_sview.



rice_sview

id	Offset	Fragment
R_1	0	A C A
R_1	1	C A A
R_1	2	A A C
R_1	3	A C A
R_2	0	A T C
R_2	1	T C A
R_2	2	C A A
R_2	3	A A A
...	...	...

Metric match results

id	Offset	Fragment
R_1	0	A C A
R_1	3	A C A
...	...	...

Figure 11: Metric match operation

A radius of 0.0 is specified in mSQL of Figure 10, therefore, only exact match will appear in the result of the metric match.

3.3.6 Metric Join (∞)

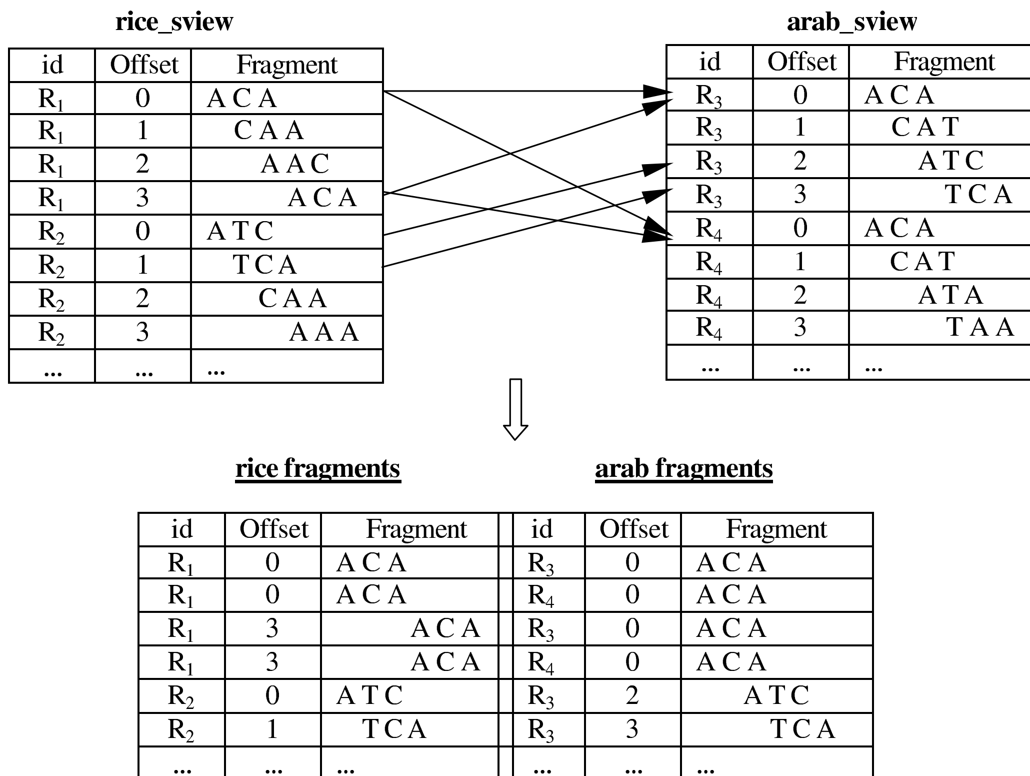
∞: Set<Fragment> × Set<Fragment> × Metric × Double → Set<Fragment> × Set<Fragment>

Metric join is a sort of join based on the distance function specified by a Metric. Metric join is similar to the extension of relational joins to spatial joins developed for spatial databases [14]. Metric-join operators modeled after a relation merge-join will tend to $O(n)$ execution time, where n is the length of the sequence.
A sample use of *Metric Join* operator in mSQL is given in Figure 12.

```
SELECT     R.fragment, A.fragment
FROM       rice_sview R, arab_sview A
WHERE      distance('base_pair_mismatch', R.fragment, A.fragment) <= 0.0
;
```

Figure 12: mSQL metric join operator

The query optimizer will recognize the "distance" operator to be a form of metric join, thus can use the underlying metric space index to accelerate the searching speed if such an index exists, otherwise, an exhaustive join is executed.
Figure 13 gives an example of metric join on two predefined sequence view rice_sview and arab_sview.



rice_sview

id	Offset	Fragment
R_1	0	A C A
R_1	1	C A A
R_1	2	A A C
R_1	3	A C A
R_2	0	A T C
R_2	1	T C A
R_2	2	C A A
R_2	3	A A A
...	...	...

arab_sview

id	Offset	Fragment
R_3	0	A C A
R_3	1	C A T
R_3	2	A T C
R_3	3	T C A
R_4	0	A C A
R_4	1	C A T
R_4	2	A T A
R_4	3	T A A
...	...	...

rice fragments / **arab fragments**

id	Offset	Fragment	id	Offset	Fragment
R_1	0	A C A	R_3	0	A C A
R_1	0	A C A	R_4	0	A C A
R_1	3	A C A	R_3	0	A C A
R_1	3	A C A	R_4	0	A C A
R_2	0	A T C	R_3	2	A T C
R_2	1	T C A	R_3	3	T C A
...	...	...	...	...	...

Figure 13: Metric join operation

A radius of 0.0 is specified in mSQL of Figure 12, therefore, only exact matched fragments appear in the result of the metric join.

3.3.7 Interior Fragment Select (σ^F)

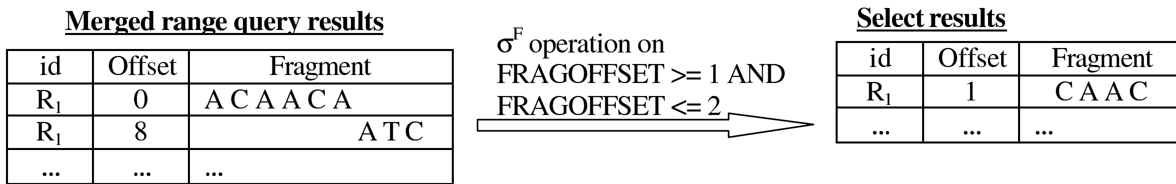
σ^F: **Relation** $\times$ C $\rightarrow$ **Relation**

where the input relation is an relation contains fragment attribute, C represents select conditions containing fragment offset properties, and the result relation is not necessarily a subset of the input relation's tuples, which is different from standard σ operator, because σ operator always produces a new relation with a subset of input relation's tuples. σ^F operator enables the select inside a fragment, i.e., a fragment is considered as a list of characters, thus position information of these characters is exposed. mSQL query optimizer is responsible for detecting whether σ^F or σ operator should be used. For example, a mSQL query in Figure 14 will result in a query tree containing σ^F operation following a range query and a merge operations.

```
SELECT      merge(R.fragment)
FROM        rice_sview R
WHERE       distance('base_pair_mismatch', 'AAC', R.fragment) <= 0.0 AND
            FRAGOFFSET(R.fragment) <= 100
GROUP BY    R.fragment
;
```

Figure 14: mSQL using interior fragment select operator

Figure 15 illustrates the interior fragment select operation.



Merged range query results

id	Offset	Fragment
R_1	0	A C A A C A
R_1	8	A T C
...	...	...

Select results

id	Offset	Fragment
R_1	1	C A A C
...	...	...

Figure 15: Interior fragment select operation

In figure 15, σ^F operation can search into the first tuple to find out the fragments satisfy the offset constraints. σ^F operator is able to find the fragment length and fragment shift size information before the merge operation.

3.4 Properties of mSQL operators

3.4.1 Identity

There is an identity property with mSQL operators *CreateFragments* and *Merge*, i.e., $\nabla^1(\Delta(s)) = s$, where s is a set of sequences.

3.4.2 High-dimensional operators

Several operators can be extended to high-dimension cases. For example, *Merge* and *GroupFragments* operators both have biological meanings. *Metric join* operator is

actually a two-dimensional case for *Metric match* operator, however, it is not clear whether higher-dimensional *Metric join* operators have corresponding biological meanings, and whether they are deserved to be introduced into mSQL.

3.4.3 Equivalence rules

An equivalence rule says that expressions of two forms are equivalent. Query optimizer can use the equivalence rules to replace an expression of the first form by an expression of the second form, or vice versa, thus in hope of achieving logically equivalent expressions, while at the same time reducing the querying cost.

For standard relational algebra, the equivalence rules include commutative selection rule, commutative theta-join rule, associative natural-join rule, etc [24, 25]. For example, commutative selection rule can be expressed as $\sigma_{\theta 1}(\sigma_{\theta 2}(T)) = \sigma_{\theta 2}(\sigma_{\theta 1}(T))$, where T is a table, θ_1 and θ_2 are selection criteria.

The extension to relational algebra as described in the prior sections introduces new equivalence rules that can benefit the query optimizer.

Rule 1: Metric join operator (∞) can be expressed using Cartesian products (×) and selections (σ).

$$T_1 \infty_{a,\, b,\, M\, ,r}\, T_2 = \sigma_{distance(Mname,\, a,\, b) \le r}\, (T_1 \times T_2),$$

where a is an attribute in table T_1, b is an attribute in table T_2, r is the metric join radius, and *Mname* is the metric name for Metric M.

Rule 2: Metric join operator (∞) is commutative.

$$T_1 \infty_{a,\, b,\, M\, ,r}\, T_2 = T_2 \infty_{b,\, a,\, M\, ,r}\, T_1,$$

where a is an attribute in table T_1, b is an attribute in table T_2, r is the metric join radius, and M is the specified Metric. Strictly speaking, the left-hand side and right-hand side are not equivalent because of the order of attributes. However, projection operations can always be used to reorder the attributes. The functionalities of both sides are actually the same. This rule is valuable if nested loop based algorithms are used to solve metric join problem, because we can always place the table with fewer tuples on the outer loop.

Rule 3: Metric join operator (∞) is associative.

$$(T_1 \infty_{a,\, b,\, M,\, r1}\, T_2) \infty_{b,\, c,\, M,\, r2}\, T_3 = T_1 \infty_{a,\, b,\, M,\, r1}\, (T_2 \infty_{b,\, c,\, M,\, r2}\, T_3),$$

where r_1 and r_2 are metric join radii, a, b, and c are attributes in table T_1, T_2, and T_3, respectively, and M is the specified Metric. This rule can be used to select the order of multiple metric join operations.

Rule 4: Metric match operator (τ) is commutative.

$$\tau_{a,\, Q1,\, M,\, r1}\, (\, \tau_{b,\, Q2,\, M,\, r2}\, (T)\,) = \tau_{b,\, Q2,\, M,\, r2}\, (\, \tau_{a,\, Q1,\, M\, ,\, r1}\, (T)),$$

where r_1 and r_2 are metric match radii, Q_1 and Q_2 are query objects, a and b are attributes in table T, and M is the specified Metric. This rule is useful in selecting the order of multiple metric match operations.

Rule 5: Metric match operator (τ) distributes over metric join operator (∞).

(1) If attribute a occurs in T_1 but not in T_2

$$\tau_{a,\, Q,\, M,\, r1}\, (T_1 \infty_{b,\, c,\, M,\, r2}\, T_2) = \tau_{a,\, Q,\, M,\, r1}\, (T_1) \infty_{b,\, c,\, M,\, r2}\, T_2$$

(2) If attribute a occurs in T_2 but not in T_1

$$\tau_{a, Q, M, r1} (T_1 \infty_{b, c, M, r2} T_2) = T_1 \infty_{b, c, M, r2} \tau_{a, Q, M, r1} (T_2)$$

(3) If attribute a occurs in both T_1 and T_2

$$\tau_{a, Q, M, r1} (T_1 \infty_{b, c, M, r2} T_2) = \tau_{a, Q, M, r1} (T_1) \infty_{b, c, M, r2} \tau_{a, Q, M, r1} (T_2)$$

Through pushing down metric select operations, query optimizer can use rule 5 to reduce the temporary table sizes, which are parameters into the expensive metric join operation.

Rule 6: Interior fragment select operator (σ^F) is select-transformed from select operator (σ).

Definition 7: $\sigma_\theta(T)$ can be **select-transform**ed into $\sigma^F_\theta(T)$ iff

- Table T contains fragment attribute,
- Select condition θ contains fragment offset constraints.

It is obvious that rule 6 is not an equivalence rule since only one direction (from σ to σ^F) holds. Rule 6 enables the query processing to deal with a larger fragment instead of many small fragments in it.

Rule 7: Merge (∇) and groupfragments (γ) operators are commutative with Interior fragment select operator (σ^F).

$$\nabla^1_a \gamma^1_a (\sigma^F_\theta(T)) = \sigma^F_\theta(\nabla^1_a \gamma^1_a (T))$$

$$\nabla^2_{a, b} \gamma^2_{a, b} (\sigma^F_\theta(T)) = \sigma^F_\theta(\nabla^2_{a, b} \gamma^2_{a, b} (T))$$

where θ is the select condition that contains fragment offset constraints, a, b are fragment attributes in table T. Rule 7 enables the query optimizer to select the order of merge/groupfragments operation and interior fragment select operation. It should be noted that for high-dimensional merge and groupfragments operators, such commutative property still exists.

3.4.4 Cost Model

Besides optimization rules, the query optimizer also needs to know the cost associated with new operators, thus can decide the most economic query plans. The cost model is listed in table 1, where m represents the number of input fragments, n represents the length of input sequences.

Operator	Cost	Description
createfragments (Δ)	$O(n)$	A linear scan
1-D Merge (∇^1)	$O(m)$	A linear scan
2-D Merge (∇^2)	$O(m)$	A linear scan
1-D groupfragments (γ^1)	$O(m\log m)$	Sort operation followed by linear scan
2-D groupfragments (γ^2)	$O(m\log m)$	Sort operation followed by linear scan
Metric match (τ)	$O(\log m)$	Very small coefficient before log$m$
Metric join (∞)	$O(m)$	Modeled after a relation merge-join
Interior fragment select (σ^F)	$O(m*Avg(fragment\ length))$	The average of fragment length in the table is a factor in the cost model

Table 1: Operators cost model

It is noteworthy that when creating a sequence view, a metric space index is also built. We currently construct the indices in $O(m \log m)$ [2]. We anticipate $O(m)$ execution time of building indices operation with the help of stream-based clustering algorithms.

4. mSQL Applications

4.1 Whole Genome Join

4.1.1 Problem Definition

Given two genomes G_1 and G_2, find mappings of substrings of the entire first genome to the substrings of the second genome.

4.1.2 Query Plan Tree

Whole genome join problem can be solved in two steps:

- Create sequence views G_1*_sview* and G_2*_sview* on sequences G_1 and G_2, respectively.
- Metric Join G_1*_sview* and G_2*_sview*.

Thus, a whole genome join can be expressed in mSQL as figure 16.

```
SELECT      R.fragment, A.fragment
FROM        G1_sview R, G2_sview A
WHERE       distance(metric_name, R.fragment, A.fragment) <= r
;
```

Figure 16: mSQL whole genome join application

where *r* is the join radius. The logical query plan tree for this SQL is illustrated in Figure 17, where *M* is used to specify the Metric corresponds to the metric name *metric_name*.

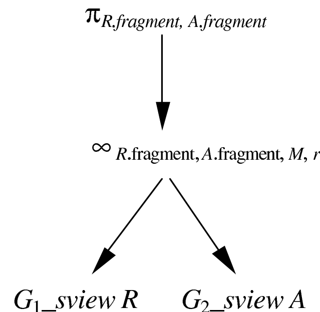


Figure 17: Query plan tree for whole genome join

Major optimization is done at metric join step with the help of metric space index defined on two sequence views G_1*_sview* and G_2*_sview*.

4.2 Conserved Primer Pair Discovery

4.2.1 Problem Definition

The problem of finding conserved primer pair can be quite clearly expressed in Figure 18.

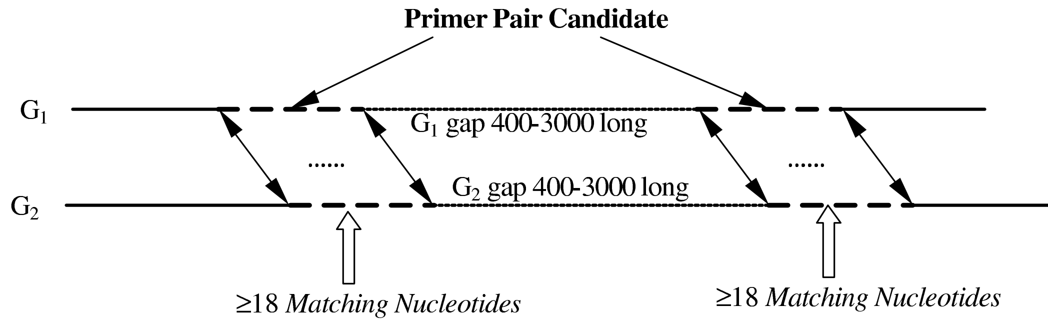


Figure 18: Find Conserved Primer Pair

For two sequences G_1 and G_2, this problem is to find the exact matched fragment pairs of these two sequences within the offset range 400 and 3000.

4.2.2 Query Plan Tree

Conserved primer pair discovery problem can be solved in four steps:

- Create sequence views G_1*_sview* and G_2*_sview* on sequences G_1 and G_2, respectively, with fragment length 18 and shift size 1.
- Utilize the metric space index to metric join G_1*_sview* and G_2*_sview*.
- To reduce the result size of metric join, a merge operation is performed on the metric join result.
- A filtering operation is used to get the fragment pairs within the offset range 300 and 4000.

Thus, a conserved primer pair discovery problem can be expressed in mSQL as figure 19.

```
SELECT *
FROM
        G1_sview R1, G1_sview R2, G2_sview A1, G2_sview A2
WHERE
        (FRAGOFFSET(R2.fragment)-FRAGOFFSET(R1.fragment)) >= 400   AND
        (FRAGOFFSET(R2.fragment)-FRAGOFFSET(R1.fragment)) <= 3000 AND
        (FRAGOFFSET(A2.fragment)-FRAGOFFSET(A1.fragment)) >= 400   AND
        (FRAGOFFSET(A2.fragment)-FRAGOFFSET(A1.fragment)) <= 3000 AND
        distance('base_pair_mismatch', R1.fragment, A1.fragment) <= 0.0       AND
        distance('base_pair_mismatch', R2.fragment, A2.fragment) <= 0.0
;
```

Figure 19: mSQL conserved primer pair discovery

The logical query plan tree for this SQL is illustrated in Figure 20, where *M* is used to specify the Metric corresponding to the metric name *base_pair_mismatch*.

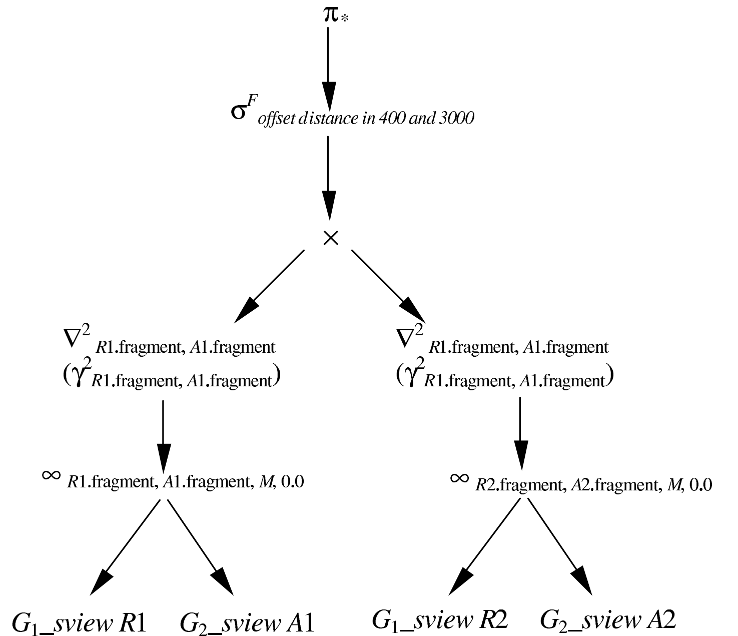


Figure 20: Query plan tree for conserved primer pair discovery

Major optimization is done at metric join step with the help of metric space index defined on two sequence views G_1*_sview* and G_2*_sview*. The query optimizer is able to find common subexpressions, thus recognize that two metric join in the logical query plan tree is actually doing the same thing, one metric join can be saved. Merge and groupfragments operations are used to reduce the temporary table size of the metric join results. A theta join is used to enforce the offset gap between 400 and 3000.

4.3 Electronic PCR

4.3.1 Problem Definition

Electronic PCR looks for sequence-tagged sites (STSs) in DNA sequences by searching for subsequences that closely match the PCR primers and have the correct order, orientation, and spacing that they could plausibly prime the amplification of a PCR product of the correct molecular weight [17].

4.3.2 Query Plan Tree

Electronic PCR problem can be solved in three steps:

- Create Electronic PCR primers table.
- Create sequence view *G_sview* on a DNA sequence *G*.
- Utilize the metric space index to accelerate the search of PCR primers in DNA sequence *G*.

Thus, an Electronic PCR problem can be expressed in mSQL as Figure 21 and Figure 22.

```
CREATE TABLE EPCRpair (
    start JAVA_OBJECT(edu.utexas.mobios.type.Sequence),
    end JAVA_OBJECT(edu.utexas.mobios.type.Sequence)
);
```

Figure 21: Create Electronic PCR primers table

```
SELECT        P.start, P.end
FROM          EPCRpair P, G_sview R1, G_sview R2
WHERE
distance('base_pair_mismatch', P.start, R1.fragment) <= 0.0   AND
distance('base_pair_mismatch', P.end, R2.fragment) <= 0.0 AND
(FRAGOFFSET(R2.fragment)-FRAGOFFSET(R1.fragment)) BETWEEN 0 AND 1000
;
```

Figure 22: Find PCR primers exist in a DNA sequence within the offset range 0 and 1000

The logical query plan tree for this SQL is illustrated in Figure 23, where *M* is used to specify the Metric corresponding to the metric name *base_pair_mismatch*.

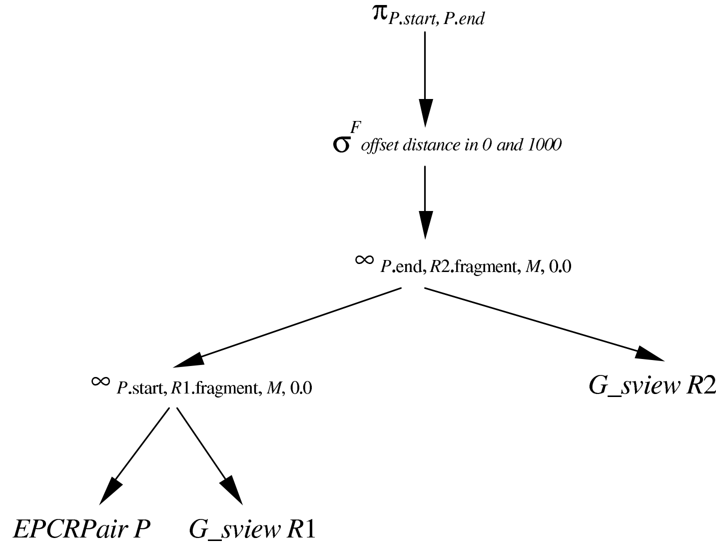


Figure 23: Query plan tree for Electronic PCR

Major optimization is done at metric join step with the help of metric space index defined on the sequence view *G_sview*. Also, the selection next to the top of this plan tree can be pushed down to the leaf of this tree, thus can further save the running time of metric join.

5. Conclusions and Future Work

mSQL is based on the standard relational operators (σ, π, *join*, *union*), with extensions in operators to enable splitting sequences into fragments (Δ), merging overlapped fragments (∇), grouping fragments that can be merged (γ), interior fragment select (σ^F), searching (τ), and joining (∞) optimized with metric space index. The sequence views concept (an extension to standard SQL view) provides a programmatic way to capture two different logical perspectives (sequences-level and fragment-level) of sequences. Biological applications (Whole Genome Join, Conserved Primary Pair Discovery, Electronic PCR) verify that mSQL has the necessary express capabilities for sequence identification and manipulation, thus is suitable for concise expression of Bioinformatics studies.

Currently sequence view works by splitting sequences into fixed length fragments, and then building metric space index on these fragments. The length of the fragments is a parameter specified by the user. Merge operations could change the length of fragments, thus the metric space index based on fixed length fragments cannot be used any more. Therefore, it would be beneficial if mSQL could be improved by introducing a new

mechanism that can build indexes based on sequences, not on fixed length fragments, or at least such indexes can still be useful when the fragment length changes.

Regular expression is a powerful tool for expressing complex queries, especially in querying secondary structures of protein data [3, 4]. Mckoi [23] provides the functionality of regular expression query. However, regular expression based queries can only consider sequences in sequences-level perspective instead of in fragment-level perspective. How to combine these two perspectives in regular expression based queries is an interesting research problem. In the future, we will incorporate regular expression query into the searching on biological sequence data, and at the same time, utilize the underlying metric space indexes.

Chaining is an important biological problem. The definition for one-dimensional chaining is given in definition 8.

Definition 8 [21]**:** Consider a set of r (possibly) overlapping intervals drawn on the line R, where each interval j has some associated value $v(j)$. **Chaining** is to select a subset of nonoverlapping intervals whose values sum to as large a number as possible.

Chaining is intrinsically an optimization problem. It might seem not proper to include one or several operators into SQL to solve such optimization problem because SQL is aiming at querying. However, if we can introduce efficient manipulation method of complex data types (like tree and graph) into mSQL, chaining can be trivially expressed as a shortest-path or longest-path problem of a graph.

Some gene sequences (for example, proteins), have four levels of structural organizations: primary, secondary, tertiary, and quaternary structures [3]. Currently, mSQL can only express queries on primary structure of gene sequences. However, this is sometimes not sufficient, because secondary structure can also provide important insights into the function of a gene sequence. The long-term goal of mSQL is to incorporate the capability to express complex queries on both primary and secondary structures of gene sequences.

References

1. Mao, R., Miranker, D. P., Sarvela, J N. and Xu, W.. Clustering Sequences in a Metric Space-the MoBIoS project. Poster of the 10th International Conference on Intelligent Systems for Molecular Biology, August 3-7, 2002, Edmonton, Canada.
2. Mao, R., Xu, W., Singh, N. and Miranker, D. P.. An Assessment of a Metric Space Database Index to Support Sequence Homology. In the proceeding of the 3rd IEEE Symposium on Bioinformatics and Bioengineering, March 10-12, 2003, Washington D.C
3. Sandeep Tata, Jignesh M. Patel. PiQA: an Algebra for Querying Protein Data Sets.
4. J. Hammel, M. Schneider. Genomics Algebra: A New, Integrating Data Model, Language, and Tool for Processing and Querying Genomic Information. CIDR'02, Asilomar, California, USA, 2002, 176-187.
5. L. Hammel, J. M. Patel. Searching on the Secondary Structure of Protein Sequences. VLDB'02, Hong Kong, China, 2002, 634-645.
6. P. Seshadri, M. Livny, and R. Ramakrishnan. Sequence Query Processing. SIGMOD'94, Minneapolis, Minnesota, USA, 1994, 430-441.

7. P. Seshadri, M. Livny, and R. Ramakrishnan. SEQ: A Model for Sequence Databases. ICDE'95, Taipei, Taiwan, 1995, 232-239.
8. P. Seshadri. Management of Sequence Data. PhD Thesis, University of Wisconsin - Madison, 1996.
9. Alberto Lerner, Dennis Shasha. Aquery: Query Language for Ordered Data, Optimization Techniques, and Experiments.
10. Gosta Grahne, Raul Hakli, Matti Nykanen, Esko Ukkonen. AQL: an Alignment Based Language for Querying String Databases.
11. Gösta Grahne, Raul Hakli, Matti Nykänen, Hellis Tamm and Esko Ukkonen: "Design and Implementation of a String Database Query Language". To appear as *Information Systems* **28**(4), pages 311-337, 2003. (Special issue on bioinformatics and biological data management.)
12. Raul Hakli, Matti Nykänen and Hellis Tamm: "Adding String Processing Capabilities to Data Management Systems". *Proceedings of the Seventh International Symposium on String Processing and Information Retrieval (SPIRE 2000)*, pages 122-131, 2000.
13. R. Hakli, M. Nykanen, H. Tamm, and E. Ukkonen. Implementing a Declarative String Query Language with String Restructuring. PADL'99, San Antonio, Texas, USA, 1999.
14. OGIS(1999). Open GIS consortium: Open GIS simple features specification for SQL (Revision 1.1). In URL: *htt://www.opengis.org/techno/specs.htm*.
15. B. Chazelle. Computational geometry: a retrospective. In Proc. ACM STOC'94, pages 75--94, 1994.
16. Ciaccia, P., Patella, M. and Zezula, P. 1997. M-tree: an efficient access method for similarity search in metric spaces. Proc. 23rd Int. Conf. Very Large Databases (VLDB).
17. Schuler GD. Sequence mapping by electronic PCR. Genome Res. 1997 May, 7(5): 541-550.
18. Growth of GenBank. http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html: NCBI (National Center for Biotechnology Information), Feb 2002.
19. Brin, S. 1995. Near Neighbor Search in Large Metric Spaces. In Proc. 21st. Int. Conf. Very Large Data Bases (VLDB), pp. 574-584.
20. V. Gaede and O. Gunther. Multidimensional Access Methods. ACM Computing Surveys, 1997.
21. D. Gusfield. Algorithms on strings, trees, and sequences: computer science and computational biology. Cambridge University Press, 1997.
22. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic Local Alignment Search Tool. Journal of Molecular Biology, 215(3), 403-410, 1990.
23. www.mckoi.com
24. A. Silberschatz, H. F. Korth, and S. Sudarshan. Database System Concepts, 4th edition. Published by McGraw-Hill, 2002.
25. S. Abiteboul, R. Hull, and V. Vianu. Foundations of Databases. Published by Addison-Wesley, 1995.