SOME THOUGHTS ON THE APPLICATION OF SPATIAL REPRESENTATION TECHNIQUES TO RELATIONAL DATABASES

Frans Coenen
Department of Computer Science,
The University of Liverpool,
Liverpool, L69 ZF, UK.
email: frans,@csc.liv.ac.uk
tel: +44 (0)151 794 3698,
fax: +44 (0)151 794 3715

1 INTRODUCTION

Below are presented some ideas on how a quantitative spatial representation may usefully be employed to: (1) represent relational database tables in a compressed format, and (2) how the proposed representation may support DB querying using this format (i.e. without necessitating a return to the parent representation). The latter is facilitated by an "arithmetic" that can be superimposed over the representation. The representation used is founded on the *tesseral* representation incorporated into the SPARTA (SPAtial Reasoning using Tesseral Addressing) system ([2]) which in turn was developed as part of the dGKBIS (dynamic Geographic Knowledge Based Information Systems) projects ([1, 3]).

This document is organised as follows. Section **??** gives an overview of the SPARTA tesseral representation together with some brief notes on tesseral representations in general, data storage and tesseral arithmetic. In Section **??** an example relational DB table is presented which is then "encoded" using the spatial representation. The mechanisms required to interact with this spatially represented DB table are then described in Section **??**. Finally, in Section **??**, some discussions are presented together with a number of questions and issues to be resolved.

2 THE SPARTA TESSERAL REPRESENTATION

The SPARTA representation assumes that any N-dimensional space comprises a set of N-dimensional isohedral (same shape) *cells* each of which is labelled with a unique *address*. Further it is assumed that such spaces are limited by a finite number of dimensions $\{D_1, D_2, \ldots, D_n\}$ each of which in turn is defined by a set of discrete coordinates. The particular configuration of a given space (number of dimensions and range of values for each dimension) is then reflected in the addressing system. This is a well established view and widely adopted, especially using labels founded on the Cartesian coordinate system (or some variation of it, e.g. latitude and longitude, or eastings and northings).

Thus the universe of discourse can be described as the set U of all possible labels such that:

$$U = P \sqcup N$$

(U is equivalent to the disjoint union of P and N) where:

$$\begin{aligned} P &= \{p \mid p \in \textit{the set of all cells representing positive space}\} \\ N &= \{n \mid n \in \textit{the set of all cells representing negative space}\} \end{aligned}$$

Any specific space will then consist of some domain of discourse (U_D) comprising a subset of U. Further each dimension (D) in such a space will be defined in terms of the set of its possible discrete values as follows:

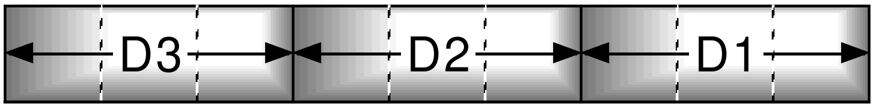

Figure 1: Bit Allocation

$$D_n = \{v \mid min_n \leq v \leq max_n \ \cdot \ min_n, max_n \in Integer \ \cdot \ max_n = min_n \times -1\}$$

(where $Integer$ is the set of all integers). Note that, for convenience, individual dimensions are numbered sequentially commencing with the number 1, and that given any space U_D the cell where all the coordinate values equate to 0 is referred to as the *global origin* of U_D.

Thus any space U_D will comprise positive and negative subspaces:

$$U_D = P_D \sqcup N_D \cdot P_D \subseteq P \cdot P_N \subseteq N$$

The most obvious labelling system to adopt is some variation on the Cartesian system. Although this system is well understood and consequently widely accepted, it is not consistent over dimensions, e.g. a 2-D point requires two coordinates while a 4-D point requires four coordinates. In addition the standard Cartesian representation becomes very unwieldy when attempting to manipulate addresses in more than two dimensions. The desire to overcome these disadvantages was the initial motivation for the SPARTA tesseral representation (although the representation is more than just a compression technique). The addressing system operates as follows. Given a set of N dimensions each of which has a range of values $min_n..max_n$, bit groupings are allocated within a signed integer sufficient to encapsulate the full range of values for each dimension. For example given three dimensions each of which has an value range of $-3..+3$, then a 9-bit signed integer can be used to encapsulate the entire space. A suitable allocation of bits is presented in Figure 1. Note that this 9-bit representation is used here for illustrative purposes only (the SPARTA system is founded on a 64-bit representation).

Using representations of this form any coordinate N-tuple can be converted into an *address* (label) using the following function:

$$\begin{aligned}
Dom(f_{cart2tess}) &= \{< v_1..v_n >\mid v_1, v_n \in D_n\} \\
Cod(f_{cart2tess}) &= \{t \mid t \in U\} \\
Gr(f_{cart2tess}) &= \{< v_1..v_n, t >\mid t = \sum_{i=1}^{n} v_i \times base_i\}
\end{aligned}$$

where Dom, Cod and Gr are the domain, codomain and graph of the function respectively, and $base_n$ is calculated using the following identity:

$$base_n = 2^{\sum_{i=0}^{n-1} b_i}$$

in which b_i equals the number of bits allocated to dimension i. In the case of the above example, the bases will be:

$$\begin{aligned}
base_1 &= 2^0 = 1 \\
base_2 &= 2^3 = 8 \\
base_3 &= 2^{3+3} = 64
\end{aligned}$$

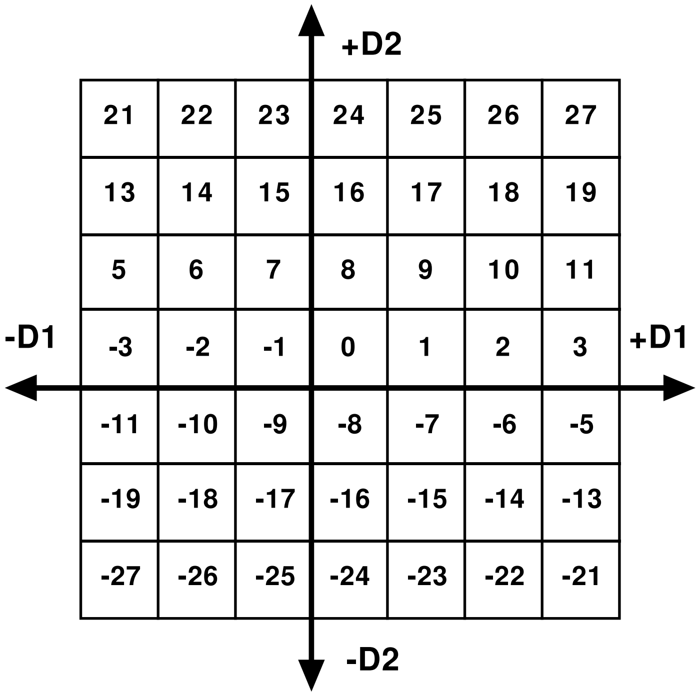

Figure 2: Section of example space in plane $D_1 - D_2$ (value for $D_3 = 0$)

thus:

$$\begin{aligned} f_{cart2tess}(v_1, v_2, v_3) &= v_1 \times base_1 + v_2 \times base_2 + v_3 \times base_3 \\ &= v_1 \times 1 + v_2 \times 8 + v_3 \times 64 \end{aligned}$$

Using this function all the addresses in the given 3-D space can be calculated. The set of addresses in the plane of $D_1 - D_2$ (where the value for D_3 is 0) is given in Figure 2. The above identity can be verified with respect to this figure by substituting appropriate coordinate values for $v_1..v_3$.

We can define a similar function, $f_{tess2cart}$, to convert back from a tesseral address to a Cartesian tuple. This is defined in terms of the $f_{tess2cart}$ function as follows:

$$\begin{aligned} Dom(f_{tess2cart}) &= \{t \mid t \in U\} \\ Cod(f_{tess2cart}) &= \{< v_1..v_n >\mid v_1 \in D_1..v_n \in D_n\} \\ Gr(f_{cart2tess}) &= \{< t, v_1..v_n >\mid f_{tess2cart}(v_1..v_n) = t\} \end{aligned}$$

(Note that the function can be implemented using appropriate left and right shift operations to isolate bit groupings).

The advantages of the representation (with respect to Cartesian and other "tesseral" representations) are as follows:

1. It results in a "left-to-right" linearisation of space which in turn offers advantages of:
   - General applicability to any number of dimensions without requiring alteration of the code or recourse to procedures and functions dedicated to a particular number of dimensions (as in the case of systems founded on Cartesian approaches).
   - Efficient data storage using ideas founded on "run-line" encoding strategies (see note in Sub-section 2.2 for further details).

- Computationally effective comparison and translation of sets of addresses.
- Predictability, i.e. given a location we can always predict the addresses of physically adjacent cells (this is not the case with other tesseral approaches).

2. Supports computationally efficient translation through the space using straightforward integer addition and subtraction, unlike other tesseral systems which required recourse to "look-up" tables or specialised tesseral processors (see note in Sub-section 2.3 for further details).
3. Provides for the rotation of objects (should this be desired).
4. Where necessary, addresses can be converted to and from Cartesian tuples in an extremely effective manner as described above (not the case with standard tesseral approaches).
5. The addressing system is intuitive.

As a consequence of these advantages many of the complexity problems associated with Cartesian approaches to spatial representation are circumvented.

2.1 Note on tesseral representations

Tesseral representations are spatial representations which are derived by repeatedly dividing a space into isohedral sub-spaces until some predefined resolution is achieved (see [5]). The resulting sub-spaces are then allocated (usually numeric) "addresses" in such a way as to reflect the hierarchical decomposition. The SPARTA representation described above can be considered to be a tesseral representation in the sense that it is a mechanism for addressing spaces decomposed into isohedral subspaces at some predefined resolution, however, there is no explicit inclusion of the concept of hierarchical decomposition and consequently this is not reflected in the addressing system. This last point is what sets the SPARTA tesseral representation apart from other tesseral approaches.

2.2 Note on data storage

Using the representation groups of addresses can be presented as a set. Further, wherever a group of addresses forms a N-dimensional cube this group of addresses can be defined in terms of two "corner" coordinates separated by an operator. In this paper the operator .. will be used to indicate such "N-cubes" as follows:

$$operand1..operand2$$

where $operand1$ represents the "corner" geometrically nearest to the global origin, and $opernand2$ the "corner" furthest away. For example the entire space in Figure 2 could be defined by the set:

$$\{0..8809\}$$

2.3 Note on tesseral arithmetic

A further significant advantage of the representation is that we can move (translate) through any space, referenced using the SPARTA approach, using simple integer arithmetic. For example, to move one point to the right (given the orientation in Figure 2) from a given reference we simply add 1 ($Base_1$), to move in the opposite direction we subtract 1 and so on. (The need to facilitate this arithmetic dictates the need for both negative and positive address — otherwise we would not be able to translate in "negative" directions.

Premiership results 20/9/98			
Home team	**Home goals**	**Away team**	**Away goals**
Coventry City	1	Newcastle	5
Derby County	2	Leicester City	0
Leeds	0	Aston Villa	0
Liverpool	3	Charlton	3
Middlesbrough	2	Everton	2
Nottingham Forest	0	West Ham	0
Southampton	1	Tottenham Hotspur	1
Wimbledon	2	Sheffield Wednesday	1

Table 1: Example database table 1

3 AN EXAMPLE DATABASE

Consider the example relational database table presented in Table **??** which comprises four columns and eight records (rows). The schema is defined as follows;

Column 1: Field = Home team, values (string[20] enumeration) = Aston Villa (0), Arsenal (1), Blackburn (2), Charlton (3) , Chelsea(4), Coventry City (5), Derby County (6), Everton (7), Leeds (8), Leicester (9), Liverpool (10), Manchester United (11), Middlesbrough (12), Newcastle (13), Nottingham Forest (14), Sheffield Wednesday (15), Southampton (16), Tottenham Hotspur (17), West Ham (18), Wimbledon (19).

Column 2: Field = Home goals, values (short Integer range) = 0 to 7.

Column 3: Field = Away team, values (string[20] enumeration) = Aston Villa (0), Arsenal (1), Blackburn (2), Charlton (3) , Chelsea(4), Coventry City (5), Derby County (6), Everton (7), Leeds (8), Leicester (9), Liverpool (10), Manchester United (11), Middlesbrough (12), Newcastle (13), Nottingham Forest (14), Sheffield Wednesday (15), Southampton (16), Tottenham Hotspur (17), West Ham (18), Wimbledon (19).

Column 4: Field = Away goals, values (short integer range) = 0 to 7.

Assuming each string character and short integer requires 16 bits then each row requires $42 \times 16 = 6720$ bits of storage and the entire table $8 \times 6720 = 53760$ bits of storage. If we consider each field to be a dimension then we have a 4-D system as follows:

$$\begin{aligned} D_1 &= \{range - 19 \ to \ +19\} \\ D_2 &= \{range - 7 \ to \ +7\} \\ D_3 &= \{range - 19 \ to \ +19\} \\ D_4 &= \{range - 7 \ to \ +7\} \end{aligned}$$

For convenience, in the following text we will indicate the maximum value for each dimension using the following notation max_{D1}, max_{D2}, max_{D3} and max_{D4} respectively. Similar notation will be used to indicate minimum values.

Thus 6 bits must be allocated to D_1 and D_3, and 4 to D_2 and D_4. The relevant bases are then:

$$\begin{aligned} base_1 &= 2^0 = 1 \\ base_2 &= 2^6 = 64 \end{aligned}$$

$$
\begin{aligned}
base_3 &= 2^{6+4} = 1024 \\
base_3 &= 2^{6+4+6} = 65536
\end{aligned}
$$

Consequently, with respect to the above table, the $f_{cart2tess}$ function is defined as follows:

$$
\begin{aligned}
Dom(f_{cart2tess}) &= \{< v_1..v_4 >\mid v_1 \in D_1..v_4 \in D_4\} \\
Cod(f_{cart2tess}) &= \{t \mid t \in U_D\} \\
Gr(f_{cart2tess}) &= \{< v_1..v_4, t >\mid t = (v_1 \times 1) + (v_2 \times 64) + (v_3 \times 1024) + (v_4 \times 65536)\}
\end{aligned}
$$

Thus (in this case):

$$U_D = \{-490975..490975\}$$

where the limits (maximum and minimum address) of U_D are calculated as follows:

$$
\begin{aligned}
& f_{cart2tess}(< min_{D1}, min_{D2}, min_{D3}, min_{D4} >) \\
& \quad (-19 \times 1) + (-7 \times 64) + (-19 \times 1024) + (-7 \times 65536) = \\
& \quad = -490975 \\
& f_{cart2tess}(< max_{D1}, max_{D2}, max_{D3}, max_{D4} >) \\
& \quad (19 \times 1) + (7 \times 64) + (19 \times 1024) + (7 \times 65536) = \\
& \quad 490975
\end{aligned}
$$

where the max_{D1} etc. and min_{D1} etc. identifiers indicate the maximum and minimum dimensions (coordinates) for U_D. These "max" and "min" identifiers will be used in the rest of this paper.

The database table is then contained in P_D, the positive part of the space U_D, i.e. the table is located in a 4-D N-cube measuring $20 \times 8 \times 20 \times 8$. More precisely the table in question is represented by a subset of P_D calculated as follows:

$$
\begin{aligned}
f_{cart2tess}(< 5, 1, 13, 5 >) &= (5 \times 1) + (1 \times 64) + (13 \times 1024) + (5 \times 65536) = 341765 \\
f_{cart2tess}(< 6, 2, 9, 0 >) &= (6 \times 1) + (2 \times 64) + (9 \times 1024) + (0 \times 65536) = 9350 \\
f_{cart2tess}(< 8, 0, 0, 0 >) &= (8 \times 1) + (0 \times 64) + (0 \times 1024) + (0 \times 65536) = 8 \\
f_{cart2tess}(< 10, 3, 3, 3 >) &= (10 \times 1) + (3 \times 64) + (3 \times 1024) + (3 \times 65536) = 199882 \\
f_{cart2tess}(< 12, 2, 7, 2 >) &= (12 \times 1) + (2 \times 64) + (7 \times 1024) + (2 \times 65536) = 138380 \\
f_{cart2tess}(< 14, 0, 18, 0 >) &= (14 \times 1) + (0 \times 64) + (18 \times 1024) + (0 \times 65536) = 18446 \\
f_{cart2tess}(< 16, 1, 17, 1 >) &= (16 \times 1) + (1 \times 64) + (17 \times 1024) + (1 \times 65536) = 83024 \\
f_{cart2tess}(< 19, 2, 15, 1 >) &= (19 \times 1) + (2 \times 64) + (15 \times 1024) + (1 \times 65536) = 81043
\end{aligned}
$$

Thus:

$$\textit{Premiership results 20/9/98}(T_1) = \{8, 9350, 18446, 81043, 83024, 138380, 199882, 341765\}$$

In the rest of this paper the identifier T_1 will be used to indicate this set. Note that each record is stored as a single integer.

Assuming that addresses are stored as 64 bit integers (as is the case for the SPARTA system) then the table will require $64 \times 8 = 512$ bits of storage. This compares very favorably with the 53760 bits calculated above.

$$\frac{512}{53760} = \frac{1}{105}$$

4 INTERACTION/DATA RETRIEVAL

Interaction with data encapsulated in the form described in the previous Section can still be achieved using a SQL style "front end". Thus the resolution of a query of the form:

```
* SELECT * FROM Premiership results 20/9/98;
```

is a simple matter of decoding each element of the set T_1 through the repeated application of the $f_{tess2cart}$ function (defined above). Similarly a query of the form:

```
* SELECT Home team FROM Premiership results 20/9/98;
```

can be addressed using a similar functions to the $f_{tess2cart}$ function to decode the output for selected columns such as a function $f_{tess2cartD1}$ to return the co-ordinate value associated with dimension D_1. Queries such as:

```
* SELECT Home team, Home goals FROM Premiership results 20/9/98;

* SELECT Home team, Home goals FROM Premiership results 20/9/98
* BY Name ASC;
```

can also be addressed in a similar manner.

In the case where we wish to limit a retrieval to a specified row in the table such as:

```
* SELECT Home team, Home goals, Away team, Away goals FROM
* Premiership results 20/9/98 WHERE
* Home team = 'Liverpool';
```

this can be implemented as:

$$R = \{10..478666\} \cap R$$

Where R is the subset of entries we are interested in. The dimensions of the N-cube 10..478666 are calculated as follows (Liverpool has the value 10 in the enumeration):

$$\begin{aligned}
&f_{cart2tess}(< 10, 0, 0, 0 >) = \\
&\qquad (10 \times 1) + (0 \times 64) + (0 times 1024) + (0 \times 65536) = 10 \\
&f_{cart2tess}(< 10, max_{D2}, max_{D3}, max_{D4} >) = \\
&\qquad (10 \times 1) + (7 \times 64) + (19 \times 1024) + (7 \times 65536) = 478666
\end{aligned}$$

Queries which combine multiple predicates by means of the logical operators AND, OR and NOT such as:

```
* SELECT Home team, Home goals, Away team, Away goals FROM
* Premiership results 20/9/98 WHERE
* Home team = 'Liverpool' OR Away team = 'Liverpool';

* SELECT Home team, Away team FROM Premiership results 20/9/98 WHERE
* NOT Home goals = 0 AND Away goals = 0;
```

can be implemented as follows:

$$R = \{10..478666, 10240..469459\} \cap R$$

$$R = \{0..478227\} \cap (\{0..19923\} \cap R)$$

where the N-cubes 10240..469459, 0..478227 and 0..19923 are calculated as follows:

$$\begin{array}{l}
f_{cart2tess}(<0,0,10,0>) = \\
\qquad (0 \times 1) + (0 \times 64) + (10 times 1024) + (0 \times 65536) = 10240 \\
f_{cart2tess}(<max_{D1}, max_{D2}, 10, max_{D4}>) = \\
\qquad (19 \times 1) + (7 \times 64) + (10 \times 1024) + (7 \times 56636) = 469459 \\
f_{cart2tess}(<0,0,0,0>) = \\
\qquad (0 \times 1) + (0 \times 64) + (0 times 1024) + (0 \times 65536) = 0 \\
f_{cart2tess}(<max_{D1}, 0, max_{D3}, max_{D4}>) = \\
\qquad (19 \times 1) + (0 \times 64) + (19 \times 1024) + (7 \times 65536) = 478227 \\
f_{cart2tess}(<0,0,0,0>) = \\
\qquad (0 \times 1) + (0 \times 64) + (0 times 1024) + (0 \times 65536) = 0 \\
f_{cart2tess}(<max_{D1}, max_{D2}, max_{D3}, 0>) = \\
\qquad (19 \times 1) + (7 \times 64) + (19 \times 1024) + (0 \times 65536) = 19923
\end{array}$$

The above only serves to give a flavor of the standard SQL predicates that can be implemented, however the examples considered indicate that the approach will directly support most standard forms of SQL query. Some queries, for example those that involve pattern matching using wild card characters:

```
* SELECT Home team, Away team FROM Premiership results 20/9/98 WHERE
* Away team LIKE '%ham%';
```

may be harder to address. Some further, more advanced SQL functions are considered in the following sub-sections.

4.1 Set Functions

Set functions in SQL allow users to perform operations on sets of data. These operations can provide valuable statistics on all or some values in a column. For example, you can count the number of rows in a table, sum the values in a column, and calculate the average of the values in a column, to name a few. Thus to count the number of entries in the above example database table would require an SQL query of the the form:

```
* SELECT NUMBER = COUNT(Home team) FROM Premiership results 20/9/98;
```

or if we wish to limit the rows returned:

```
* SELECT NUMBER = COUNT(Home team) FROM Premiership results 20/9/98
* WHERE Home goals = 0;
```

In terms of the above tesseraly represented database table such queries would be addressed by determining the cardinality of the set T_1 or R as appropriate using a function $f_{cardinality}(Location)$ which returns the "size" of its argument which must be a set of addresses (expressed as N-cubes or otherwise).

League Table 20/9/98			
Team	**Games played**	**Goal Diff.**	**Points**
Aston Villa	6	6	14
Derby County	6	4	12
Liverpool	6	5	11
Wimbledon	6	2	11
Leeds	6	4	10
Middlesbrough	6	2	9
West Ham	6	1	9
Manchester United	4	5	8
Newcastle	6	4	8
Arsenal	5	1	7
Nottingham Forest	6	-2	7
Tottenham Hotspur	6	-6	7
Sheffield Wednesday	6	2	6
Charlton	6	1	6
Everton	6	-1	6
Chelsea	4	0	5
Leicester City	6	-2	5
Blackburn	5	-4	4
Coventry City	6	-8	4
Southampton	6	-14	1

Table 2: Example database table 2

4.2 Multiple Table Retrieval (Joins)

SQL also enables the user to retrieve data from several tables with a single query. This is achieved using joins and nested (or sub) queries. Consider the addition to the above database of the table given in Table 2. The schema for this table is as follows:

Column 1: Field = Team, values (string[20] enumeration) = Aston Villa (0), Arsenal (1), Blackburn (2), Charlton (3) , Chelsea(4), Coventry City (5), Derby County (6), Everton (7), Leeds (8), Leicester City(9), Liverpool (10), Manchester United (11), Middlesbrough (12), Newcastle (13), Nottingham Forest (14), Sheffield Wednesday (15), Southampton (16), Tottenham Hotspur (17), West Ham (18), Wimbledon (19).

Column 2: Field = Games played, values (short Integer range) = 0 to 38.

Column 3: Field = Goal Diff., values (short Integer range) = -31 to 31.

Column 4: Field = Points, values (short integer range) = 0 to 114.

This schema will then result in the following dimensions:

$$
\begin{aligned}
D_1 &= \{range - 19to\ +19\} \\
D_2 &= \{range - 38to\ +38\} \\
D_3 &= \{range - 63to\ +63\} \\
D_4 &= \{range - 114to\ +114\}
\end{aligned}
$$

Thus 6 bits must be allocated to D_1, 7 bits for D_2 and D_3, and 8 for D_4. Note that to ensure that the resulting encapsulation is a subset of the set P_U we must add a factor of 31 to the "goal difference" to obtain the correct co-ordinate. The relevant bases are then:

$$
\begin{aligned}
base_1 &= 2^0 = 1 \\
base_2 &= 2^6 = 64 \\
base_3 &= 2^{6+7} = 8192 \\
base_3 &= 2^{6+7+7} = 1048576
\end{aligned}
$$

The data in Table 2 is then comprised of the following elements:

$$
\begin{aligned}
f_{cart2tess}(<0,6,6,14>) &= (0 \times 1) + (6 \times 64) + (6 \times 8192) + (14 \times 1048576) = 14983552 \\
f_{cart2tess}(<6,6,4,12>) &= (6 \times 1) + (6 \times 64) + (4 \times 8192) + (12 \times 1048576) = 12870022 \\
f_{cart2tess}(<10,6,5,11>) &= (10 \times 1) + (6 \times 64) + (5 \times 8192) + (11 \times 1048576) = 11829642 \\
f_{cart2tess}(<19,6,2,11>) &= (19 \times 1) + (6 \times 64) + (2 \times 8192) + (11 \times 1048576) = 11805075 \\
f_{cart2tess}(<8,6,4,10>) &= (8 \times 1) + (6 \times 64) + (4 \times 8192) + (10 \times 1048576) = 10772872 \\
f_{cart2tess}(<12,6,2,9>) &= (12 \times 1) + (6 \times 64) + (2 \times 8192) + (9 \times 1048576) = 9707916 \\
f_{cart2tess}(<18,6,1,9>) &= (18 \times 1) + (6 \times 64) + (1 \times 8192) + (9 \times 1048576) = 9699730 \\
f_{cart2tess}(<11,4,5,8>) &= (11 \times 1) + (4 \times 64) + (5 \times 8192) + (8 \times 1048576) = 8683787 \\
f_{cart2tess}(<13,6,4,8>) &= (13 \times 1) + (6 \times 64) + (4 \times 8192) + (8 \times 1048576) = 8675725 \\
f_{cart2tess}(<1,5,1,7>) &= (1 \times 1) + (5 \times 64) + (1 \times 8192) + (7 \times 1048576) = 7602497 \\
f_{cart2tess}(<14,6,-2,7>) &= (14 \times 1) + (6 \times 64) + (-2 \times 8192) + (7 \times 1048576) = 7577998 \\
f_{cart2tess}(<17,6,-6,7>) &= (17 \times 1) + (6 \times 64) + (-6 \times 8192) + (7 \times 1048576) = 7545233 \\
f_{cart2tess}(<15,6,2,6>) &= (15 \times 1) + (6 \times 64) + (2 \times 8192) + (6 \times 1048576) = 6562191 \\
f_{cart2tess}(<3,6,1,6>) &= (3 \times 1) + (6 \times 64) + (1 \times 8192) + (6 \times 1048576) = 6553987 \\
f_{cart2tess}(<7,6,-1,6>) &= (7 \times 1) + (6 \times 64) + (-1 \times 8192) + (6 \times 1048576) = 6537607 \\
f_{cart2tess}(<4,4,0,5>) &= (4 \times 1) + (4 \times 64) + (0 \times 8192) + (5 \times 1048576) = 5497092 \\
f_{cart2tess}(<9,6,-2,5>) &= (9 \times 1) + (6 \times 64) + (-2 \times 8192) + (5 \times 1048576) = 5480841 \\
f_{cart2tess}(<2,5,-4,4>) &= (2 \times 1) + (5 \times 64) + (-4 \times 8192) + (4 \times 1048576) = 4415810 \\
f_{cart2tess}(<5,6,-8,4>) &= (5 \times 1) + (6 \times 64) + (-8 \times 8192) + (4 \times 1048576) = 4383109 \\
f_{cart2tess}(<16,6,-14,1>) &= (16 \times 1) + (6 \times 64) + (-14 \times 8192) + (1 \times 1048576) = 1188240
\end{aligned}
$$

Therefore:

$$
\begin{aligned}
\textit{League table 20/9/98} = \{&1188240, 4383109, 4415810, 5480841, 5497092, 6537607, 6553987, \\
&6562191, 7545233, 7577998, 7602497, 8675725, 8683787, 9699730, \\
&9707916, 10772872, 11805075, 11829642, 12870022, 14983552\}
\end{aligned}
$$

In the rest of this paper the above set will be indicated using the identifier T_2.

A SQL query involving a join between the $Premiership\ results$ 20/9/98 and $League\ table$ 20/9/98 tables might then be expressed as follows:

```
* SELECT p.Home team, p.Home goals, p.Away team, p,Away goals
* FROM Premiership~results~20/9/98 p, League~table~20/9/98 l
* WHERE p.Home team = l.Team AND l.points >= 9;
```

So that the DBMS can identify the table to which a column belongs (in SQL) the column name is proceeded by a full-stop ".". Note also that the join is actually established in the $WHERE$

clause. To cut down on the amount of typing required correlation names are used such as p and l in the above example. The above can be defined, using the tesseral representation in SQL, as follows:

$$R = R_1 \cap T_1$$

such that:

$$\begin{array}{rcl} R_1 & = & \{x \mid \forall y \in (\{9691136..120310153\} \cap T_2) \cdot \\ & & x = \{f_{cart2tess}(v_1, 0, 0, 0)..f_{cart2tess}(v_{D1}, max_{D2}, max_{D3}, max_{D4})\} \cdot \end{array}$$

where the "max" identifiers indicate maximum dimension values for T_2, and the N-cube $\{9691136..120310153\}$ is calculated as follows:

$$\begin{array}{rcl} f_{cart2tess}(< 0, 0, 0, 9 >) & = & 9691136 \\ f_{cart2tess}(< 19, 38, 63, 114 >) & = & 120310153 \end{array}$$

The N-cube thus calculated defines the sub-space of P_D where the number of points is 9 or greater.

Note that in the above case there is a direct correlation between the coordinate values for D_1 in T_1 and T_2. If this were not the case a correction would have to be applied to each of the coordinates v_1.

4.3 Nested queries

SQL queries can be nested to any level. For example:

```
* SELECT Team, Games played, Goal Diff, points
* FROM League table 20/9/98
* WHERE Team in
*    (SELECT Home team
*    FROM Premiership results 20/9/98
*    WHERE Home goals > 1);
```

(Note that instead of the "in" predicate we can also use "exists" or "any-or-all").

Nested queries of the above form can be processed in a similar manner to that described to achieve a "join" in the foregoing sub-section:

$$\begin{array}{rcl} R & = & R_1 \cap T2 \cdot \\ R_1 & = & \{x \mid \forall y \in (\{128..47867\} \cap T_1) \cdot \\ & & x = \{f_{cart2tess}(v_1, 0, 0, 0)..f_{cart2tess}(v_1, max_{D2}, max_{D3}, max_{D4})\} cdot \\ & & cart2tess(y) \rightarrow < v_1, \ldots\} \end{array}$$

where the "max" identifiers refer to the maximum dimensions for table T_2, and:

$$\begin{array}{rcl} 128 & = & f_{cart2tess}(< 0, 2, 0, 0 >) \\ 478675 & = & f_{cart2tess}(< 19, 7, 19 >) \end{array}$$

SQL also provides a union operator to derive the union of several queries provide that the number of select fields are the same. This can also be supported using the proposed tesseral representation in a similar manner as described above.

5 DISCUSSION

In this paper we have considered the application of the SPARTA/dGKBIS representation to the domain of relational databases and data retrieval from such databases. The paper clearly establishes that tables represented using the tesseral representation require less storage space. But:

Question 1: Is storage space a current issue in the relational database community?

The paper also demonstrates how standard SQL queries can be interpreted in the context of the representation. But:

Question 2: Does the interpretation of queries in terms of the tesseral representation offer any computational advantages? If so is this significant?

The only set operator used in the above examples is the intersection operator ($\cap$). Given that the tables, using the proposed representation, comprise sets of integers other set operators can be applied, i.e. union ($\cup$), complement ($\setminus$), subset and proper-subset ($\subseteq$ and $\subset$), and superset and proper-superset ($\supseteq$ and $\supset$). But:

Question 3: What meaning could be attached to the result of applying such operators to subsets of elements contained in tesseraly represented database tables?

further:

Question 4: By combining sequences of set operations applied to sets of table elements, could some sort of reasoning be performed on the data? and if so would this be useful?

In the above no explicit use is made of the arithmetic that the representation supports. We do not translate through the space for example. Thus:

Question 5: Under what circumstances might we wish to translate through a database table?

A possible application of the arithmetic is to "expand" queries. For example given:

```
* SELECT Home team
* FROM Premiership results 20/9/98
* WHERE Home goals = 4;
```

This would be implemented as follows:

$$R = \{256..478483\} \cap T1$$

($245 = f_{cart2tess}(0, 4, 0, 0)$ and $478483 = f_{cart2tess}(max_{D1}, 4, max_{D3}, max_{D4})$). This would return the empty set $\emptyset$. In this case, so as to retrieve some data, we might want to "expand" the query by looking for "near" matches (misses?), i.e. `WHERE Home goals >= 3 AND Home goals <= 5`. It would be possible to automatically revise the query be resizing the N-cube $\{256..478483\}$ by adding the set $\{-64..64\}$. This will have the effect of expanding the given N-cube by one cell in both directions along the $Home\ goals$ dimension:

$$\{256..478483\} + \{-64..64\} = \{(256 - 64)..(478483 + 64)\} = \{192..478547\}$$

However:

Question 6: Is this approach to producing "near" answers in anyway beneficial?

A further possible application of the arithmetic and available set operators is to determine shapes that enclose groups of addresses (records) that share some common feature, e.g. (in the context of the above example) "score-draws". But:

Question 7: Under what circumstances might a user wish to identify such shapes?

Alternatively:

Question 8: Is this a possible approach to providing some sort of data mining facility?

If no useful function can be found for the arithmetic then there is no need to include addresses defining negative space (negative addresses are only required to facilitate translation in all directions). If this is the case the addressing system can be revised so that only positive space is described.

The answers to the above questions are all the subject of further work. It is not until suitable answers are arrived at that the real value (if any) of the proposed approach can be evaluated. It should also be noted that the content of this paper is all theoretical, only some very rudimentary implementation has been undertaken.

References

[1] Coenen, F.P., Diaz, B.M., Shave. M.J.R., Beattie, B. and Bench-Capon, T.J.M. (1997). N-Dimensional Reasoning Using a Tesseral Representation. In Hunt, J. and Miles, R. (Eds.), Research and Development in Expert Systems XIV, proceedings of ES'97, SGES publications, ISBN 1 899621 20 2, pp97-108.

[2] Coenen, F.P., Beattie, B., Shave, M.J.R, Bench-Capon, T.J.M. and Diaz, B.M. (1998). Spatial Reasoning Using The Quad-Tesseral Representation. To appear in AI Review.

[3] Coenen, F.P., Beattie, B., Bench-Capon, T.J.M., Diaz, B.M. and Shave. M.J.R. (1998). Spatio-Temporal Reasoning Using A Multi-Dimensional Tesseral Representation. Proceedings ECAI'98, John Wiley & Sons, pp140-144.

[4] Coenen, F.P., Beattie, B., Bench-Capon, T.J.M., Shave, M.J.R and Diaz, B.M. (1996). An Ontology for Linear Spatial Reasoning. In Wagner, R.R. and Thoma, H. (Eds), Database and Expert Systems Applications, (Proceedings DEXA'96), Lecture Notes in Computer Science 1134, Springer Verlag, pp718-727.

[5] Diaz, B.M. and Bell, S.B.M. (1986). Spatial data processing using tesseral methods, *Natural Environment Research Council publication*, Swindon, England.