SPATIO-TEMPORAL REASONING USING A MULTI-DIMENSIONAL TESSERAL REPRESENTATION

F.P. Coenen, B. Beattie, T.J.M. Bench-Capon, B.M. Diaz and **M.J.R. Shave** [1]

Abstract. A versatile and universally applicable quantitative multi-dimensional reasoning mechanism founded on a unique linear tesseral representation of space is described. The reasoning mechanism is based on a constraint satisfaction mechanism supported by a heuristically guided constraint selection strategy. The mechanism has been incorporated into a spatio-temporal reasoning system, the SPARTA (SPAtial Reasoning using Tesseral Addressing) system, which has been applied successfully to a diverse number of spatial applications.

1 INTRODUCTION

In this paper we describe a quantitative spatio-temporal reasoning system founded on a tesseral representation of Euclidean space which is universally applicable to N-dimensional spatial data stored or presented using a raster style encoding. The universal applicability offered by the reasoning system is a direct consequence of the tesseral representation on which it is founded. This representation may be viewed as either an intermediate representation which links the reasoning mechanism with some primary representation (raster and by extension vector), or as a primary representation in its own right.

The reasoning mechanism comprises a constraint satisfaction approach to spatial problem solving supported by a heuristically guided constraint selection mechanism so as to minimise the search space. The mechanism has been incorporated into a demonstration spatio-temporal reasoning system, the SPARTA (SPAtial Reasoning using Tesseral Addressing) system. Spatial problems are passed to this system in the form of a script comprising a set of object descriptions and a set of constraints defining the relationships "desired" to exist between pairs or groups of objects. The system then produces all solutions that satisfy the given constraints and outputs these solutions in a graphical or textual format as directed by the user.

2 REPRESENTATION

N-dimensional space can be conceptualised as comprising a set (P) of N-dimensional cells where each cell has some unique identifier referred to as an *address* or *reference*. Sub-spaces within this space can then be defined in terms of subsets of P. Spatial reasoning is concerned with the interpretation and manipulation of knowledge concerning the attributes of spatial objects. If we wish to manipulate objects whose attributes are expressed in terms of sets of addresses it is desirable to adopt an addressing mechanism that supports computationally effective processing of these sets. The most obvious starting point for the representation of such addresses is the well understood Cartesian coordinate system. A consequence of this will be the existence of negative coordinates, thus the set P should more accurately be described as a subset of a larger set (I) which includes the set of cells (N) representing negative space, i.e.

$$P \amalg N = I$$

(the *disjoint union* of P and N is equivalent to I), or

$$N = I \setminus P$$

(N is equivalent to the relative complement of I in P).

Although the Cartesian system is well understood, it is not consistent over differing numbers of dimensions, i.e. 4-D objects are referenced using 4 coordinates while 2-D objects only require 2 coordinates. Consequently the Cartesian system becomes extremely unwieldy when considering spaces defined by (say) more than two dimensions. These problems are eliminated by the referencing system proposed here, where the coordinates representing a particular cell are "compacted" into a single (signed) integer. For the purpose of this document we will assume a 64 bit signed integer with a bit pattern as follows:

$$\pm t..t\ t..t\ z..z\ z..z\ y..y\ y..y\ x..x\ x..x$$

where $\pm$ is the sign bit, and the symbols x (16 bits), y (16 bits), z (16 bits) and t (15 bits) represent the bits to be associated with particular coordinates used to reference (in this case) 4-D space. Note that:

1. It is not necessary to associate any particular dimension with any particular bit grouping.
2. Any other distribution of bits could equally well have been allocated provided that each dimension of interest comprises a continuous sequence of bits.
3. Alternative bit patterns can be defined to address (say) 8-D reasoning.
4. The sign bit is included to provide for negative addressing required in order to support various translation operations.

Assuming the above bit pattern we can redefine the set I as follows:

$$\begin{aligned} I \quad = \quad & \{r | r = x + 2^{16}y + 2^{32}z + 2^{48}t \ \cdot \ x, y, z, t \in integer \ \cdot \\ & -32767 \leq x, y, z \leq 32767 \ \cdot \ -16383 \leq t \leq 16383\} \end{aligned}$$

[1] Department of Computer Science, The University of Liverpool, Chadwick Building, P.O. Box 147, Liverpool L69 3BX, England. Tel: 0151 794 3698 Fax: 0151 794 3715 email: frans@csc.liv.ac.uk

ECAI 98. *13th European Conference on Artificial Intelligence*
Edited by Henri Prade
Published in 1998 by John Wiley & Sons, Ltd.

I is the set of elements r such that each r is calculated from $x + 2^{16}y + 2^{32}z + 2^{48}$ where x, y, z, t belong to the set of integers such that $-32767 \leq x, y, z \leq 32767$ and $-16383 \leq t \leq 16383$ hold. We can define the set P (positive space) in a similar manner:

$$\begin{aligned} P \quad = \quad & \{r | r = x + 2^{16}y + 2^{32}z + 2^{48}t \ \cdot \ x, y, z, t \in natural \ \cdot \\ & 0 \leq x, y, z, t \leq 32767 \ \cdot \ 0 \leq t \leq 16383\} \end{aligned}$$

and the set N:

$$\begin{aligned} N \quad = \quad & \{r | r = x + 2^{16}y + 2^{32}z + 2^{48}t \ \cdot \ x, y, z, t \in integer \ \cdot \\ & -32767 \leq x, y, z \leq 32767 \ \cdot \ -16383 \leq t \leq 16383 \ \cdot \\ & x \vee y \vee z \vee t < 0\} \end{aligned}$$

Negative space is defined as space in which at least one coordinate is negative.

Using this representation we can define any *object space* (i.e. the space which contains the objects we are interested in) subject to the limitations of the definition for I. An example space is presented in Figure 1. The general advantages of the representation are as follows:

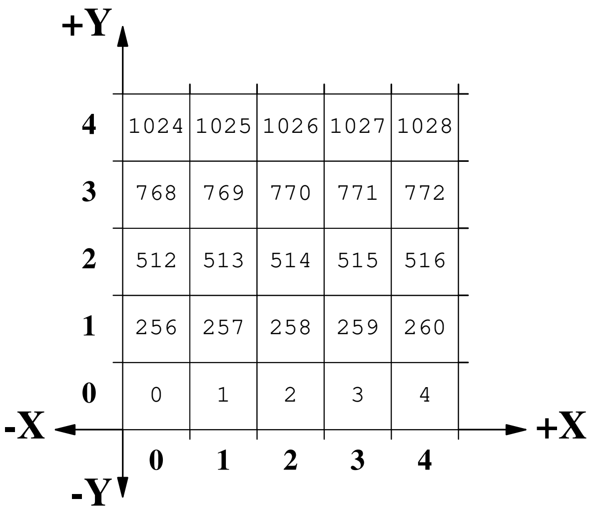

Figure 1. Addressing for 2D 5x5 object space

1. All references are unique and conceptually simple to generate.
2. The representation is applicable (without alteration) to any number of dimensions within the confines of the definition of the set I.
3. The representation has the effect of linearising N-dimensional space which in turn has significant benefits with respect to (a) data storage, (b) comparison of sets of addresses and (c) translation through the space (see below).

The referencing system proposed here has many similarities with ideas concerning tesseral representations ([**?**]). Tesseral representations are founded on ideas concerning the hierarchical decomposition of N-Dimensional space into isohedral sub-spaces (*cells*) until some application dependent resolution is reached. Each sub-space is then referenced using some number system. In the case of the representations proposed here, any given object space is conceptually tessellated down to some predefined resolution, and each cell given a reference (address) comprising a single integer whose bit pattern is made up of the coordinates of that cell. Hence we refer to our representation as a "tesseral" representation.

2.1 Data Storage

As a consequence of the linearisation the elements belonging to a set of addresses can be ordered. More usefully clusters of elements can be grouped into multi dimensional *boxes* defined by the "corner" address nearest the origin of a given object space (the 0 cell) and that furthest away. Thus in Figure 1 the set of cells:

$$\{257, 258, 259, 513, 514, 515, 769, 770, 771\}$$

can be defined as a box of the form:

$$\{257..771\}$$

provided that the `..` operators is understood to mean "the sequence of cells bounded by the multi-dimensional box whose corner address nearest to the origin is given by the prefix operand, and whose corner address furthest away from the origin is given by the postfix operand". More formally we can say that any set of addresses S can be defined in terms of a subset of "boxes" defined over I thus:

$$S \subseteq \{m..n | m, n \in I \wedge m \leq n\}$$

It should be noted that this form of "data compression" is particularly advantageous when dealing with spaces more general than that defined by two dimensions.

2.2 Comparison of spaces

Knowledge of the linearisation can also be used to advantage when comparing sets of addresses. For example given sets S_1 and S_2 of the form defined above, the relation *subset* which might link these two sets can be defined as:

$$S_1 subset S_2 \Leftrightarrow \forall m..n \in S_1 \ \cdot \ \exists s..t \in S_2 \ \cdot \ m..n \ within \ s..t$$

The relation $S_1 subset S_2$ is true if and only if for all boxes $m..n$ in the set S_1 there exists a box $s..t$ in the set S_2 such that $m..n$ is within $s..t$ (an appropriate definition for the $within$ relation is assumed). Thus to compare sets of addresses where the elements are defined in terms of boxes we do not have to consider all elements contained within a box but only the defining corner addresses. Again, this advantage is of particular relevance when dealing with spaces comprising more than two dimensions.

2.3 Translation of space

To move through the space `I` (as represented here) we can use standard integer addition and subtraction. For example, to move any address one cell to the right and two cells upwards we simply add the address 513 (measured from the 0 address). To move in the opposite direction we subtract 513 (or add -513). In this manner we can translate sets of addresses effectively and efficiently by defining a translate function:

$$\begin{aligned} translate(S_1, S_2) \quad \rightarrow \quad & \{n..m | \forall < p..q, s..t > \in S_1 \times S_2 \ \cdot \\ & m = p + s \wedge n = q + t\} \end{aligned}$$

the function $translate(S_1, S_2)$ produces the set of boxes $n..m$ such that for all Cartesian pairs ($< p..q, s..t >$) in S_1 and S_2, $m = p + s$ and $n = q + t$. For example, given a set:

$$S = \{257..771\}$$

we can use the translation function to (say) move S one cell to the right and two cells upwards:

$$translate(\{257..771\}, \{513..513\}) \rightarrow \{770..1284\}$$

In a similar manner we can uniformly expand the set S, or stretch it in a particular direction or directions.

3 SPATIAL OBJECTS

The reasoning system recognises a number of categories of object:

- *Fixed objects* which cannot be moved, i.e. they have a "fixed" location.
- *Free objects* which have a known shape but no fixed location (and thus they can be moved).
- *Shapeless objects* which have no given shape or location (they are only known to exist somewhere in the object space).

These objects can have a great many attributes, the most significant of which are

- *Location*
- *Shape*
- *Relationship to other objects*

To which we can add further attributes such as *size*, *contiguity* , *orientation* etc. Of course some attributes are implied by others. For example given a fixed object (i.e. an object whose location is known), all other attributes can be deduced. Similarly in the case of a free object the object's shape will also indicate the nature of attributes such as size, contiguity and so on.

Given an object n, belonging to a set of $objects$, the location (L_n) of that object can be expressed in terms of some subset of the object space O (the space in which all objects of interest are known to exist). Similarly the shape (S_n) of an object n can be expressed in terms of a subset (S_n) of I (shape need not necessarily be defined using positive coordinates only). Thus:

$$\forall n \in objects, L_n \subseteq O, S_n \subseteq I$$

4 SPATIAL RELATIONS

The connection between spatial objects can be expressed by relations that link the location(s) associated with one object to the location(s) associated with another object:

$$L_1 \ relation \ L_2$$

where L_1 and L_2 are two set each element of which comprises a set of addresses representing a possible location for an object (i.e. L_1 and L_2 are sets of sets of address).

In a system where attributes are expressed in terms of sets of uniquely identified cells, the primitive relations are the standard group of set relations. We can consider relationships as *filters* or as *mappings*. Filters return $True$ or $False$, whilst mappings "map" some operation onto one operand with respect to another operand to produce a "new" set of cells.

1. Example filters: equals, subset, superset, disjoint, intersects.
2. Example mappings: complement, intersection.

Using these relations many of the standard topological relations encountered in spatial reasoning can be expressed, e.g. within, contains, overlaps, disjoint ([**?**, **?**, **?**, **?**, **?**]). The expressiveness of these relations can be increased by allowing offsets to be applied to the locations before the relation is processed. For example we may have two objects L_1 and L_2 whose locations are given by the sets $\{257..514\}$ and $\{514..771\}$ and a subset relation:

$$L_1 \ subset \ L_2$$

to establish whether L_1 is contained within L_2. In this case the result will be $false$. However, we may wish to establish whether L_1 is contained within a space surrounding L_2 by (say) one cell. We can express this relation by first applying an offset to L_2 thus:

$$L_1 \ subset \ (translate(L_2, \{-257..257\}))$$

which will be equivalent to:

$$\{257..514\} \ subset \ \{257..1028\}$$

in which case the result will be $true$.

5 SPARTA SYSTEM

<script> ::= <spaceDef> <classDef> <instanceDef> <constraintDef>
<spaceDef> ::= space(<dimensions>).
<classDef> ::= class(CLASS_NAME,OBJECT_TYPE). | class(CLASS_NAME,OBJECT_TYPE,<shapeDef>).
<instanceDef> ::= instance(INSTANCE_NAME, CLASS_NAME,<locationDef>).
<constraintDef> ::= constraint(<operandDef>, OPERATOR,<operandDef>).
<operandDef> ::= (INSTANCE_NAME) | (INSTANCE_NAME,<offsetDef>).

Table 1. Top level syntax for SPARTA scripting language

The SPARTA (SPAtial Reasoning using Tesseral Addressing) system is designed to take as input a set of objects and a set of constraints which are desired to hold between these objects, and output one or more configurations of objects that satisfy the constraints (assuming a solution exists). The input is presented in the form of a script, expressed in a PROLOG like syntax, which is then analysed and processed. Syntactically the scripting language comprises a number of primary constructs of the form given in Table 1 (using BNF notation). The first construct defines the desired object space in terms of its maximum coordinates. Class definitions only include a shape definition if the object type referred to is a free object. The location definition that forms part of an instance definition then defines the object's location in the case of a fixed object, and a *location space* in which the object can be said to exist in the case of a free or shapeless object. Constraints are used to express the relations that are desired to exist between locations (possibly with offsets applied). Note that shape, location and offset definitions are expressed in terms of sets of tesseral addresses.

6 THE CONSTRAINT SATISFACTION PROCESS

The constraint satisfaction process commences with a single *root node* in a solution tree. Initially this node contains a list of *constraints-to-do* as defined in the input script, and an object space containing no objects but dimensioned according to the definition given in the script. A constraint is then selected from the constraints-to-do list following the constraint selection criteria outlined below (Section 8). The system then attempts to resolve this constraint with three possible outcomes:

1. The constraint cannot be satisfied in which case (at this stage) we conclude that no solution exists.
2. One or more compatible solutions exist; therefore update the object space with respect to the identified solutions.
3. A number (more than one) of non compatible solutions is produced; therefore create a new level in the tree with branches equivalent to the number of solutions, each branch terminating in a new node containing an updated version of the object space and constraints-to-do list.

In this manner a solution tree is dynamically created. If all the given constraints have only one solution the tree will consist of a single (root) node. If, however, the script includes constraints that have more than one solution, the tree will consist of a number of levels, each level representing a point in the solution process where the satisfaction of a constraint generates more than one solution. Whenever an additional level in the tree is created each branch is processed in turn until either all constraints have been satisfied, in which case the solution is stored; or an unsatisfiable constraint is discovered. On completion of processing a particular branch the current node is removed from the tree and the system backtracks to the previous node. If all branches emanating from this node have also been processed this node is also expunged. The process continues until all branches in the tree have been investigated and all solutions generated. As a result of this approach the solution tree in its entirety never exists, only the current branch and those higher level nodes which merit further investigation.

7 CONSTRAINT SATISFACTION

The individual constraint satisfaction process comprises five steps: (1) identify operands, (2) apply offsets, (3) identify Cartesian pairs, (4) apply the operator to the identified pairs and (5) grouping of results.

7.1 Identify operands

From Section 4 a constraint comprises an operator and two operands. The operands indicate the locations (one or more), expressed in terms of a set of of one or more sets of addresses, associated with two distinct objects. If the objects in question were referenced in a previously processed constraint the objects will be contained in the object space as defined in the current node in the solution tree and thus location information can be extracted directly. Otherwise the location for the object will need to be generated according to the nature of the object:

- If the object is a fixed object the location will be given in the script.
- If the object is a free object the operand will be equivalent to set of locations dictated by the different manners in which a the shape definition can be "fitted into" the prescribed location space.
- If the object is a shapeless object the required location is the location space (as defined in the script) within which the object is known to exist.

```
space([5,5]).

class(equals, fixed).
class(blob, shapeless).
class(plus, free, -256..256, -1..1).

instance(e1, equals, 0..2, 512..514).
instance(b1, blob, 1..1027).
instance(p1, plus, 257..1028).

constraint((e1), intersects,
    (p1, offset(-256..-256))).
constraint((b1), complement,
    (e1, offset(-257..257))).
```

Table 2. Example script

Consider the script given in Table 2. Here we have three classes of objects, one of each type, and an instance of each class. We also have two constraints relating objects `e1` and `p1`, and `b1` and `e1`. The first is a filter because neither of the objects is a shapeless object, and states that any valid pair of locations for the objects `e1` and `p1` must be such that the first intersects the second "moved down" in the -Y direction by one cell (offset $\{-256..-256\}$). The second constraint is a mapping because one of the objects (`b1`) is a shapeless object and states that, what ever the shape of `b1` might be, it should not include any part of the location of the fixed object `e1` expanded in all directions by one cell (offset $\{-257..257\}$).

The location for the fixed object `e1` is given in the script. The location to be associated with the shapeless object `b1` is equivalent to its location space (`1..1027`) also given in the script. The free object `p1` has a number of possible locations, i.e. the shape definition ($\{-256..256, -1..1\}$) can be fitted into the specified location space ($\{257..1028\}$) in four different ways:

```
{{258..770, 513..515},
 {259..771, 514..516},
 {514..1026, 769..771},
 {515..1027, 770..772}}
```

Thus the constraints can be rewritten as:

```
constraint(({{0..2, 512..514}}), intersects,
    ({{258..770, 513..515},
    {259..771, 514..516}, {514..1026,
    769..771}, {515..1027, 770..772}},
        offset({-256..-256}))).
constraint(({{1..770}}), complement,
    ({{0..2, 512..514}}, offset({-257..257}))).
```

7.2 Apply offsets

Once the operands have been identified the next stage is to apply the offsets (if any) by adding the offset to each location associated with the operand using the translation function. As a consequence the constraints are transformed to:

```
constraint(({{0..2, 512..514}}), intersects,
    ({{2..544, 257..259}, {3..515, 258..260},
    {258..770, 513..515}, {259..771,
    514..516}})).
constraint(({{1..1027}}), complement,
    ({{-257..259, 255..771}})).
```

7.3 Determine Cartesian pairs

The next step is to identify all Cartesian pairs that exist between the two operand sets. Thus the constraints can now effectively be rewritten as follows:

```
constraint((({0..2, 512..514}}), intersects,
    ({{2..544, 257..259}})).
constraint((({0..2, 512..514}}), intersects,
    ({{3..515, 258..260}})).
constraint((({0..2, 512..514}}), intersects,
    ({{258..770, 513..515}})).
constraint((({0..2, 512..514}}), intersects,
    ({{259..771, 514..516}})).
constraint(({{1..1027}}), complement,
    ({{-257..259, 255..771}})).
```

7.4 Apply operator

We are now in a position to apply the operator in each case according to whether it is a filter or a mapping constraint. Considering the filter constraints first, all but the second are satisfiable. Consequently the following pairs of locations are valid locations for objects `e1` and `p1`:

```
<{{0..2, 512..514}}, {{258..770, 513..515}}>
<{{0..2, 512..514}}, {{514..1026, 769..771}}>
<{{0..2, 512..514}}, {{515..1027, 770..772}}>
```

Thus three possible solutions exist. The second constraint is a mapping of the form:

```
{1..1027}\{-257..259, 255..771}={1025..1027}
```

Consequently the location space for `b1` is refined to $\{1..1027\}$ and thus we have a location pair for `b1` and `e1` of the form:

```
<{{1..1027}}, {{0..2, 512..514}}>
```

7.5 Grouping of results

Although, in the above subsection, satisfaction of the first constraint results in three solutions we can store these three solutions as a single set. Thus:

```
<{{0..2, 512..514}},    {{258..1026, 513..771,
    772..772, 1027..1027}}>
```

Thus we are saying that `e1` can be located anywhere in the first set, and `e2` anywhere in the second set. The advantage is that the constraint has only one (compound) solution and thus would not give rise to a new level in the solution tree. Thus we have one possible location for `e1` and `b1` and three possible locations for `p1` (but described by a single set.

8 CONCLUSION

A quantitative spatio-temporal reasoning mechanism has been described founded on a tesseral representation of space. The mechanism offers the following significant advantages:

1. It is universally applicable regardless of the number of dimensions under consideration.
2. It is fully compatible with Raster representations rendering it suited to a wide range of applications founded on such representations.
3. It is conceptually simple and computationally effective.

The proposed mechanism has been incorporated into a spatio-temporal reasoning system, the SPARTA system, which has been tested against a great many spatial problems including, site identification for civil engineering projects ([**?**]), Geographic Information Systems (GIS) ([**?**]), environmental impact assessment ([**?**]) and timetabling. Other, more esoteric, applications include classic AI problems such as N-queens problems and multi-dimensional shape fitting scenarios ([**?**]). Current work is focused on noise pollution modelling and assessment in the city of London, and marine electronic chart interaction.

REFERENCES

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

[2] J.F. Allen, Maintaining knowledge about temporal intervals, *Communications of the ACM* **26**(11) 832-843 (1983).

[3] D. Hernández, Relative representation of spatial knowledge: the 2-D case, in D.M. Mark and A.U. Frank, *Cognitive and Linguistic Aspects of Geographic Space*, Kluwer, Dordrecht, Netherlands, 373-385 (1991).

[4] C. Freksa, Temporal reasoning based on semi-intervals, *Artificial Intelligence* **54** 199-227 (1992).

[5] M.J. Egenhofer, Deriving the composition of binary topological relations, *Journal of Visual Languages and Computing* **5** 133-149 (1994).

[6] A.G. Cohn, J.M. Gooday, B. Bennet and N.M. Gotts, A logical approach to representing and reasoning about space, *Artificial Intelligence Review* **9** 255-259 (1995).

[7] B. Beattie, F.P. Coenen, T.J.M. Bench-Capon, B.M. Diaz and M.J.R. Shave, Spatial reasoning for GIS using a tesseral data representation, in N. Revell, and A.M. Tjoa (Eds), *Database and Expert Systems Applications* (Proceedings DEXA'95), Lecture Notes in Computer Science 978, Springer Verlag, 207-216 (1995).

[8] F.P. Coenen, B. Beattie, B.M. Diaz, T.J.M. Bench-Capon and M.J.R. Shave, A temporal calculus for GIS using tesseral addressing, in M.A. Bramer and A.L. Macintosh (Eds), *Research and Development in Expert Systems XI, Proceedings of ES'94* 261-273 (1994).

[9] F.P. Coenen, B. Beattie, B.M. Diaz, T.J.M. Bench-Capon and M.J.R. Shave, Temporal reasoning using tesseral addressing: towards an intelligent environmental impact assessment system, *Journal of Knowledge-Based Systems* **9**(5) 287-300 (1996).

[10] F.P. Coenen, B. Beattie, T.J.M. Bench-Capon, B.M. Diaz and M.J.R Shave, A tesseral approach to N-dimensional spatial reasoning, in A. Hameurlain and A.M. Tjoa (Eds), *Database and Expert Systems Applications* (Proceedings DEXA'97), Lecture Notes in Computer Science 1308, Springer Verlag, pp633-642 (19977).