A TESSERAL APPROACH TO N-DIMENSIONAL SPATIAL REASONING

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

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

Abstract. A qualitative multi-dimensional spatial reasoning system is described founded on a tesseral representation of space. Spatial problems are presented to this system in the form of a script describing the nature of the N-dimensional space (the *object* space), the spatial objects of interest and the relations that are desired to exist between those objects. Objects are defined in terms of classes and instances of classes with locations and shapes defined as sets of tesseral addresses. Relations are expressed in terms of topological set relations which may be quantified through the application of tesseral *offsets*. Solutions to spatial problems are generated using a heuristically guided constraint satisfaction mechanism. The heuristics are directed at limiting the growth of the search tree through a constraint selection strategy applied at each stage of the satisfaction process. The general advantages of the system are that it is conceptually simple, computationally effective and universally applicable to N-dimensional problem solving.

1. Introduction

An N-dimensional spatial reasoning system is described that combines a reasoning mechanism, founded on set relations, with quantitative input and output data is described. The approach is designed for incorporation into systems which use raster encoded data, particularly Geographic Information Systems (GIS), so that a spatio-temporal reasoning capability can be attached to those systems. Particular GIS end-user applications that have been investigated by the authors include optimal site location scenarios, noise pollution investigations and nautical chart interaction. Other end applications that have been considered include N-dimensional scheduling and timetabling scenarios, well known AI "puzzles" such as the 8-queens problem and multi-dimensional shape fitting tasks. The last named are used to illustrate this paper.

The system is founded on a tesseral representation of space and uses a heuristically guided constraint satisfaction mechanism to resolve spatial problems defined in the form of a script. The heuristics are directed at the effective selection of constraints so as to minimise the growth of the solution tree and the pursuit of unsuccessful branches within that tree. Scripts comprise object definitions (in terms of classes and instances of such classes) and constraint definitions describing the relations that are desired to exist between objects. Object locations and shapes are defined in terms of sets of tesseral addresses and may be generated through interaction with existing spatial data formats. Output can be in a number of file or graphical formats as directed by the user.

2. Representation

Tesseral representations are well established and a substantial literature exists concerning the nature of such representations (for example [3] and [4]). Broadly such representations are founded on ideas concerning the hierarchical subdivision of N-dimensional space into isohedral sub-spaces down to a predefined resolution. The resulting sub-spaces are then referenced using some addressing system which has the effect of linearising the tessellated space. Such systems can have an arithmetic associated with them which provides translation through the space, and rotation of objects within that space. A 2-D illustration of the tesseral representation used to support the reasoning mechanism is given in Figure 1. Note that addresses increase in steps of 256 in the Y direction. This *base* has been chosen for the benefit of this document, but any other base could equally well have been selected. Objects of interest are considered to exist only in the positive quadrant of the representation, referred to as the *object space*. However the remaining quadrants need to be referenced to support the arithmetic associated with the representation.

+Y

1020	1021	1022	1023	1024	1025	1026	1027	1028
764	765	766	767	768	769	770	771	772
508	509	510	511	512	513	514	515	516
252	253	254	255	256	257	258	259	260
−4	−3	−2	−1	0	1	2	3	4
−260	−259	−258	−257	−256	−255	−254	−253	−252
−516	−515	−514	−513	−512	−511	−510	−509	−508
−772	−771	−770	−769	−768	−767	−765	−764	−763
−1028	−1027	−1026	−1025	−1024	−1023	−1022	−1021	−1020

−X ← → +X

−Y

Figure 1: *Referencing for 2-D space*

The distinctive features of this representation, compared with other tesseral representations are:

1) The resulting linearisation is intuitively obvious and predictable (i.e. given any sub-space the addresses of all the physically adjacent sub-spaces can always be predicted). This is not the case with most other tessellations which follow a "Z" linearisation often referred to as the Morton sequence ([7]).

2) Translation and rotation algorithms are implemented using integer arithmetic. For example (with reference to figure 1) to move one tile in a "north-easterly" direction we add the address 257 to a "current" address, regardless of the physical location of that current address (to move to (say) the north-west we would add 255 and so on). Rotation is implemented using the standard approach taken from complex number theory. Consequently more efficient manipulation of objects is achieved than that supported by traditional tesseral arithmetics which, to be effective, required specialised tesseral processors.

The principal benefit of the linearisation (a feature of all tesseral addressing systems) is that N-dimensional space can be manipulated in one dimensional terms. Consequently the spatial reasoning technique described is universally applicable regardless of the number of dimensions under consideration. A further feature of tesseral representations is *runline encoding* which allows sequences of addresses to be stored in terms of start and end addresses. Knowledge of the relative location of runline encoded sequences within the linearisation then provides for effective comparison of sequences of addresses. For example to determine whether one sequence of addresses stands in some relation to another sequence of addresses it is not necessary to compare each address in one with each address in the other. Similarly when adding (or subtracting) one sequence of addresses to another it is only necessary to add (or subtract) the start and end addresses.

Cartesian representations, when compared to tesseral approaches, display the following general disadvantages:

1) Addresses (X-Y-Z-T coordinate tuples) are not uniform across the dimensions, the number of coordinates required to address a point increases with the number of dimensions under consideration.

2) Translation through the space can only be achieved by incrementing and decrementing the individual coordinates making up an address.

3) It is computationally expensive to compare and manipulate addresses.

4) The computer storage requirements are higher.

The representation can be extended to 3-D by applying multiples of a further base to the set of 2-D addresses. In this document we will use a base of 65536 (256^2) to define 3-D space. The representation can be extended in a similar manner to encompass further higher dimensions.

3. Object description

Using the above representation the shape and/or location of quantitatively defined spatial objects can be described in terms of a set of tesseral addresses. Using the system scripting language such objects are defined in terms of classes and instances of such classes using a syntax of the following form (single quotes surround literals and punctuation):

```
<CLASS>     : 'class(' CLASSNAME ',' OBJECTTYPE ').'
            | 'class(' CLASSNAME ',' OBJECTTYPE, SHAPE ').'
            ;

<INSTANCE> : 'instance(' INSTANCENAME ',' CLASSNAME ').'
            | 'instance(' INSTANCENAME ',' CLASSNAME ','
                          LOCATIONSPACE ').'
            | 'instance(' INSTANCENAME ',' CLASSNAME ','
                          MODIFIERS ').'
            | 'instance(' INSTANCENAME ',' CLASSNAME ','
                          LOCATIONSPACE ',' MODIFIERS ').'
            ;
```

A number of types of object are recognised:

1) Fixed objects: Objects which have a known location and consequently a known shape.
2) Free objects: Objects that have a known shape but no specific location other than a general *location space* within which the object is known to exist.
3) Shapeless objects: Objects that have no known shape or location (other than a general *location space* within which the object is known to exist).

In the case of a fixed object there is no requirement to define its shape as this will be given by its location. In the case of a free object, where the shape definition is omitted, the shape is assumed to comprise a single address. Where a location definition is omitted this is assumed, by default, to extend to the entire object space. The nature of locations (and shapes) can be augmented by the addition of "modifiers", for example we may include a `rotate` modifier indicating that the shape may be rotated.

4. Spatial relations

Using the proposed tesseral representations, relationships (constraints) between objects can be expressed using standard set relations (e.g. = - Equals, $\neq$ - notEquals, $\cap$ -Intersection, $\cap$ - not intersection, $\subseteq$ - Subset, etc.). With respect to the scripting language a relationship between two spatial objects can is expressed as follows:

```
<CONSTRAINT> : 'constraint(' OPERAND1, OPERATOR, OPERAND2 ').'
              | 'constraint(' OPERAND, OPERATOR ').'
              ;
```

where the operands are single instances, lists of instances or entire classes of instances. Where a constraint comprises only one operand this is a shorthand for expressing a set of constraints where the operator links all possible instance pairs defined by the operand (in this case, of course, the operand must describe at least two instances). Using the above format standard topological relations can be expressed. To increase the expressiveness of the range of set operators available *offsets* may be applied to locations associated with either operand so that the existing set of relations can be augmented with directions and/or distances. As a result we can formulate relations such as `toTheNorthOf`, `disjoint(N)` (disjoint by a distance `N`), `toTheNorthOf(N)` (to the north of by a distance `N`) and so on. Further discussion on the application of offsets can be found in [1].

The ontology used to develop the spatial scripting language used here can be found in [5].

5. One-dimensional scenario

To illustrate the system's operation we will consider three shape fitting scenarios of increasing complexity commencing with a 1-D example (this section), and progressing to 2-D and 3-D in the following two sections. This provides a clear demonstration of how the same technique is applied irrespective of the number of dimensions.

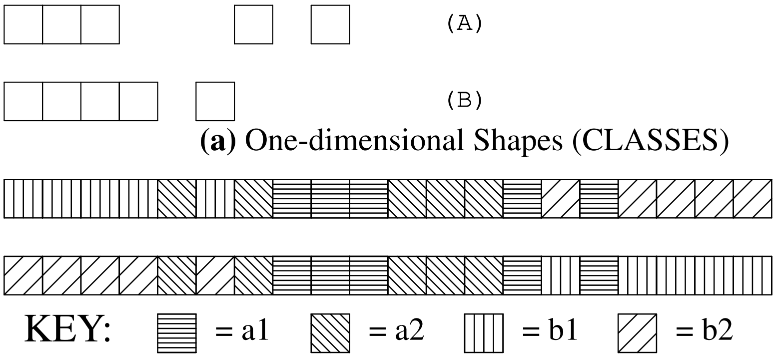
In Figure 2(a) two discontinuous 1-D shapes are presented, labelled `A` and `B`. They are defined in terms of a set of 1-D tesseral addresses (the set of addresses incorporating the 0 address and running immediately parallel to the X-axis in Figure 1). Let us assume: (1) that these two shape definitions are associated with two classes of free object, (2) that there are two instances of each of these classes, `a1`, `a2`, `b1` and `b2`, and (3) that we wish to fit these instances into a 1-D object space, comprising 20 addresses, in such a manner that no part of one instance's location coincides with that of another. Let us also assume that there is nothing that prevents us from rotating these objects (if only through 180°). We can express this problem in the form of a script as follows:

```
space([19]).

class(A, free, [0..2, 6, 8]).
class(B, free, [0..3, 5]).

instance(a1, A).
instance(a2, A, rotation).
instance(b1, B, rotation).
instance(b2, B, rotation).

constraint([a1, a2, b1, b2], notIntersection).
```

This script comprises four basic constructs: `space`, `class`, `instance` and `constraint`. The first is used to declare the object space we wish to work with. The second defines the classes of objects we are interested in and the third is used to declare instances of those classes. Note that, to reduce the amount of processing required, one of the instances is not rotated. Note also that the unary (one operand) constraint expression is a shorthand method of defining the following set of constraints:

```
constraint(a1, notIntersection, a2).
constraint(a1, notIntersection, b1).
constraint(a1, notIntersection, b2).
constraint(a2, notIntersection, b1).
constraint(a2, notIntersection, b2).
constraint(b1, notIntersection, b2).
```

When this script is passed to the tesseral spatial reasoning system two possible solutions are generated, these are illustrated in Figure 2(b).



(a) One-dimensional Shapes (CLASSES)

(b) Solutions

Figure 2: *One-dimensional shape fitting scenario.*

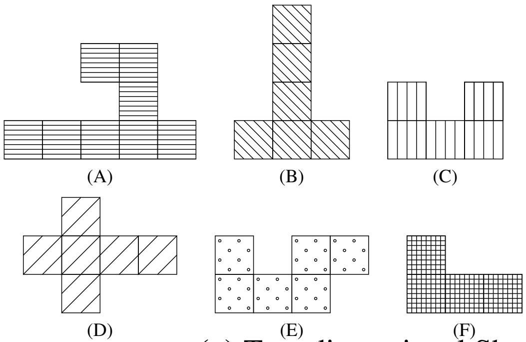
6. Two-dimensional scenario

Without requiring any change to the representation or the underlying constraint satisfaction mechanism the system can equally well be applied to two dimensional shape fitting problems. In Figure 3(a) a number of two dimensional shapes (Classes) are presented. Assuming one instance of each class and a 6x6 object space a script can be derived which will cause the spatial reasoning system to place these instances into this object space in such a manner that no instance overlaps any other. The script is as follows:

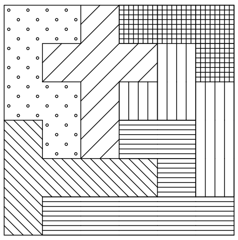
```
space(6,6)

class(A, free, [0..4, 259, 514..515]).
class(B, free, [0..2, 257, 259]).
class(C, free, [0..2, 258..259, 514]).
class(D, free, [-255, 0..3, 257]).
class(E, free, [0..2, 256, 258..259]).
class(F, free, [0..2, 256]).
```

```
instance(a1, A).
instance(b1, B, rotation).
instance(c1, C, rotation).
instance(d1, D, rotation).
instance(e1, E, rotation).
instance(f1, F, rotation).

constraint([a1, b1, c1, d1, e1, f1], notIntersection).
```

The definition of these shapes can be confirmed by reference to the 2-D "object space" given in Figure 1. Note that, to reduce the amount of work required to produce a result, one of the instances is again not rotated. The solution, on completion of the script, is given in Figure 3(b).



(a) Two-dimensional Shapes (CLASSES)



(b) Solutions

Figure 3: *Two-dimensional shape fitting scenario.*

7. Three dimensional scenario

A similar scenario can be designed with respect to a 3-D object space. Consider the seven shapes (classes) given in Figure 4(a). Assuming a 4x4x4 object space, and one instance of each class, we can write a script designed to fit these instances into the object space (without overlap) as follows:

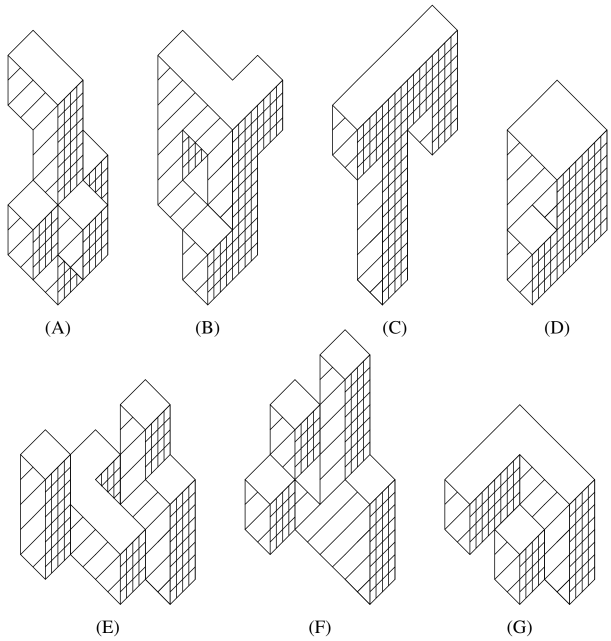
```
space(4,4,4)

class(A, free, [-65535, -255, 0..2, 65537, 131073, 131329]).
class(B, free, [0..1, 257, 513, 65537, 66049, 131073..131074,
                 131329, 131585]).
class(C, free, [0, 65536, 131072, 131074, 196607..196610]).
class(D, free, [0..2, 257..258, 65537..65538, 65793..65794]).
class(E, free, [0, 255..256, 511, 766..768, 65536, 65792,
                 66302, 131328]).
class(F, free, [0, 256, 512, 767..768, 65536, 65792, 66304,
                 131328, 196864]).
class(G, free, [0, 255..256, 65536, 65792, 66046..66048]).

instance(a1, A).
instance(b1, B, rotation).
instance(c1, C, rotation).
instance(d1, D, rotation).
instance(e1, E, rotation).
instance(f1, F, rotation).
instance(g1, G, rotation).

constraint([a1, b1, c1, d1, f1, e1,  g1], notIntersection).
```

Again, for reasons of efficiency, rotation is not permitted for one of the instances - if we allow rotation of instance a this will simply produce 24 different views of the same result. There is one solution to this problem as demonstrated in Figure 4(b).

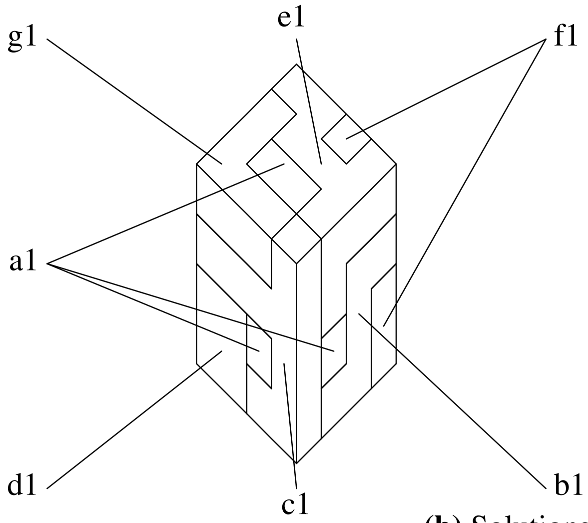
Although it may be argued that the 1-D and 2-D shape fitting scenarios given in the foregoing two sections may be considered to be relatively simple in that the number of possible combinations in each case is "small" (346112 in the 1-D scenario and 7077888000 in 2-D scenario), the above example is significantly more taxing. We can carry out some rough calculations to indicate that there are in the region of $6x10^{13}$ location combinations for the above scenario. The fact that the system can successfully resolve such problems using standard computer hardware is an indication of the strength of the approach, particularly the representation on which it is founded, but also the heuristically guided constraint satisfaction technique currently used to resolve scripts describing tesserally defined spatial problems. The foregoing sequence of examples also clearly illustrates that the system can successfully operate in any number of dimensions without requiring any alteration to the basic representation or the operation of the system. Because of space limitations we can not present a 4-D shape fitting scenario here, however, the system has proved itself successful at resolving such scenarios.

8. Conclusions

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



(a) Three-dimensional Shapes (CLASSES)



(b) Solutions

Figure 4: *Three-dimensional shape fitting scenario*.

1) It is universally applicable regardless of the number of dimensions under consideration.
2) It is fully compatible with Raster data representations rendering it suited to a wide range of applications, e.g. image analysis, reasoning for GIS and map interaction.
3) It is conceptually simple and computationally effective.

The approach has been incorporated into a spatial reasoning system, the SPARTA system, which has been tested against a great many application scenarios including environmental impact assessment and noise pollution (see [2] and [6]) for further detail). Current work is focused on noise pollution modelling and assessment in the city of London, and marine electronic chart interaction.

9. References

1. Beattie, B., Coenen, F.P., Bench-Capon, T.J.M., Diaz. B.M. and Shave, M.J.R. (1995). Spatial reasoning for GIS using a tesseral data representation, in N. Revell and A.M. Tjoa, A.M. (Eds), *Database and Expert Systems Applications, (Proceedings DEXA'95)*, Lecture Notes in Computer Science 978, Springer Verlag, pp207-216.
2. Beattie, B., Coenen, F.P., Hough, A. Bench-Capon, T.J.M., Diaz, B.M. and Shave, M.J.R (1996). Spatial reasoning for environmental impact assessment, *Third International Conference on GIS and Environmental Modelling*, Santa Barbara: National Centre for Geographic Information and Analysis, WWW and CD.
3. Bell, S.B.M., Diaz, B.M., Holroyd, F.C. and Jackson, M.J.J. (1983). Spatially referenced methods of processing raster and vector data, *Image and Vision Computing* **1**(4) 211-220.
4. Diaz, B.M. and Bell, S.B.M. (1986). Spatial data processing using tesseral methods, *Natural Environment Research Council publication*, Swindon, England (1986).
5. 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 R.R. Wagner and H. Thomas (Eds), *Database and Expert Systems Applications, (Proceedings DEXA'96)*, Lecture Notes in Computer Science 1134, Springer Verlag, 718-727.
6. Coenen, F.P., Beattie, B., Bench-Capon, T.J.M., Diaz, B.M. and Shave, M.J.R. (1996). Spatial reasoning for geographic information systems. *Proceedings 1st International Conference on GeoComputation*, School of Geography, University of Leeds **1** 121-131.
7. Morton, G.M. (1966). A computer oriented geodetic data base, and a new technique in file sequencing, *IBM Canada Ltd.*.