A Tesseral Approach To Multi-dimensional Reasoning

Frans Coenen, Michael Shave, Bernard (Diz) Diaz, Michael Shave,
Trevor Bench-Capon and Bridget Beattie
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 universally applicable multi-dimensional reasoning system is described. The system is founded on a dedicated tesseral representation of space and uses a heuristically guided constraint satisfaction mechanism to solve N-dimensional spatial problems. The representation allows space to be linearised so that the same technique is applicable to problems in any number of dimensions. The principal advantages offered by the system are that of general applicability and computational effectiveness. With respect to Cartesian and other tesseral representations it provides for efficient translation and rotation of space using standard arithmetic operations, effective data storage using "block" encoding and fast comparison of groups of addresses. The system has been successfully targeted at a number of application areas, but particularly Geographic Information System (GIS) and Environmental Impact Assessment (EIA).

1 Introduction

A N-dimensional quantitative spatial reasoning system, the SPARTA (SPAtial Reasoning using Tesseral Addressing) system, based on a tesseral representation of space is described. Using the system N-dimensional problems are defined in terms of an object and constraint scripting language founded on a PROLOG predicate style of syntax [Coenen, 1996]. Scripts are passed to the system which then processes the constraints using a heuristically guided depth first constraint satisfaction algorithm. The result (assuming that a solution exists) is one or more numerically referenced configurations of all objects, defined in the script, that satisfy the given constraints. The configuration(s) may be output in a number of formats depending on the nature of the application. Examples include two- and three- dimensional imaging, application specific file formats such as DXF (Drawing eXchange Format) and PBM, or encodings such as HTML.

2 Object Space

Given a particular application, all spatial entities of interest are considered to exist in some N-dimensional orthogonal parallelepiped space referred to as the *object space*. This space is tesselated (sub-divided) into a set of isohedral (same shape) N-dimensional cubic subspaces down to some desired resolution. Each subspace is then allocated a unique integer address (or reference) that reflects its location within the space with respect to some arbitrarily selected corner origin. Within the context of this paper we will assume that the maximum number of dimensions to be considered is four and that addresses are represented as 32bit signed integers whose bit pattern is defined as follows:

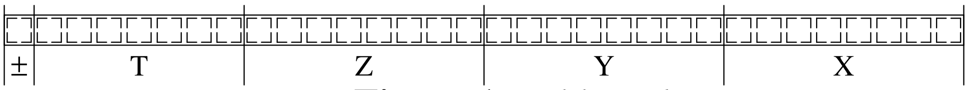

Figure 1*: Address bit pattern*

Of course any other bit pattern may be selected if it is more appropriate to a particular application. Similarly the number of dimensions to be considered can be increased or decreased, or longer or shorter integers may be used (for example 64 bit or 16 bit integers). Further, no particular relevance need be attached to the labelling of the bit groupings in Figure 1; the X, Y, Z and T (Time) labelling has simply been used here to facilitate understanding.

In Figure 2 an example object space is given defined by two dimensions, `X` and `Y`, such that `X = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}` and `Y = {0, 1, 2, 3, 4}`. Note that each address is made up of two coordinates using the bit pattern presented in Figure 1. Note also that, as a consequence of the addressing mechanism, the space is linearised commencing at the 0 bottom-left corner address (the *origin*) to the top-right address.

Y

1792	1793	1794	1795	1796	1797	1798	1799	
1536	1537	1538	1539	1540	1541	1542	1543	
1280	1281	1282	1283	1284	1285	1286	1287	
1024	1025	1026	1027	1028	1029	1030	1031	
768	769	770	771	772	773	774	775	
512	513	514	515	516	517	518	519	
256	257	258	259	260	261	262	263	
0	1	2	3	4	5	6	7	

X

Figure 2: *Tesselated 2-D space*

The advantages offered by the tesseral addressing mechanism, particularly with respect to the Cartesian system, are as follows:

1) Any cell can be addressed using a single integer. Consequently the computer storage requirements using tesseral addressing are both low and constant.
2) Translation through the space can be achieved through simple integer addition and subtraction. For example, in Figure 2, to move one tile to the "north-east" we always add the address 257 regardless of where we are in the space (to move to the south-west we add the fictitious address -257, and so on). The approach thus avoids recourse to computationally expensive incrementation and decrementation of individual coordinates or cumbersome trigonometric techniques.
3) The N-dimensional space in question is linearised in a broad bottom-left to top-right manner. Knowledge of this linearisation can be utilised as follows:
   - Block encoding of addresses: Groups of addresses contained in a single orthogonal parallelogram can be stored in terms of two addresses representing the corners closest to and furthest away from the origin. Consequently further savings in computer storage requirements are obtained. (We refer to this as *block encoding* so as to relate it to the more usual *run-line encoding*.)
   - Simple comparison of sets of addresses: To determine whether one set of addresses stands in some relation to another set of addresses it is not necessary to compare each address in one with each address in the other. Instead, knowledge of the relative location within the linearisation can be used to achieve efficient comparison of sets of addresses.
   - Simple addition (and subtraction) of blocks of addresses: When adding (or subtracting) one block of addresses to another it is only necessary to add (or subtract) the two pairs of corner addresses.

An additional advantage of the approach is that it is fully compatible with established Raster representations found in GIS (Geographic Information Systems), image encoding and other computer map representations. For further discussion relating to tesseral addressing and its merits interested readers are referred to [Diaz and Bell, 1986].

3 Object Classes

It is assumed that the size and resolution at which the object space is defined is sufficient to contain/represent all objects we wish to reason about. Three types of object are identified:

Fixed objects:
: Objects whose location is fixed within the object space. Consequently their nature (e.g. shape, size, contiguity etc.) is known.

Free objects:
: Objects whose shape is known but not their location.

Shapeless objects:
: An object of which nothing is known (other than that it exists somewhere within the object space).

Using the scripting language, objects are defined in terms of classes and instances of those classes. A class, identified by a class name, defines the object's type, and (in the case of a free object) its shape. The shape is defined in terms of a set of addresses which must incorporate the zero origin address (negative addresses may be used). Note that a fixed object's shape is defined by its location and therefore fixed object class declarations need not include a shape definition (locations are defined as part of instance declarations as described in Section 4 below). Some high level syntax used for the definition of classes is presented in Table 1.

```
<CLASSDECS>    : <CLASSDEC>
               | <CLASSDEC> <CLASSDECS>
               ;

<CLASSDEC>     : `class(' CLASSNAME `, fixed).'
               | `class(' CLASSNAME `, free,' <SHAPE> `).'
               | `class(' CLASSNAME `, shapeless).'
               ;

<SHAPE>        : `shape([' <SETTA> '])' ;
```

Table 1: *Class declaration syntax*

4 Object Instances

When declaring instances of classes, details of attributes that may be associated with spatial objects may be included according to the type of the object - fixed, free or shapeless. The most significant is the object's location or its location space. This is again defined in terms of a set of addresses. The distinction between locations and location spaces is that the first is associated with fixed objects, and describes a sub-space within the object space where fixed objects **are** located; while the second is associated with free or shapeless objects and describes the sub-spaces of the object space within which such objects **may** be located.

4.1 Free Object Instances

With respect to free objects there exists a special relationship between the shape of such objects and the associated location space. The number of possible locations for the object is defined (by default) as the number of different ways the entire shape can be fitted into the location space without rotation. Which of these locations is acceptable is then determined by the nature of the constraints associated with the spatial problem, which is in turn dependent on the nature of the application. Note that if no locations for a free object can be generated the object cannot be physically realised and consequently it can be said that no solution to the given spatial problem exists. Given a particular application it is possible to influence the generation of locations as follows:

1) Sub-shape: We may include (in the instance declaration) some tesseral sub-shape definition which is a sub-set of the shape defined in the associated class, and then determine candidate locations for the free object by attempting to fit only this sub-shape into the given location space.
2) Rotation: Given a particular application it may be appropriate to permit the shape to be rotated so that additional locations may be generated.
3) Inclusion: We may wish to insist that certain fixed groups of addresses form part of any candidate location for a fixed object. Typically such inclusion groups of addresses are associated with the locations of a fixed object.
4) Exclusion: We may wish to exclude certain groups of addresses form part of any candidate location.
5) Contiguity: Where we have included and/or excluded some set of addresses we may wish to insist that the result is only acceptable if the location set of addresses is in some way contiguous. In addition we may require it to be contiguous on a particular set of addresses.
6) Size: Similarly, where we have included and/or excluded some set of addresses, in may also be the case that we wish to specify some size qualification whereby the resulting locations are only acceptable if they are less than, equal to or greater than some given size (number of addresses).

4.2 Shapeless Object Instances

Shapeless objects have no known location or shape, only a location space in which they can be said to exist. Where such objects appear in a spatial problem scenario the aim is usually to refine, through an appropriately specified set of constraint *mappings* (see below), the location space associated with such objects. Although no knowledge is available concerning the shape, certain other attributes may be known which can be used to validate the result of any refinement of the location space that might take place as a consequence of the satisfaction of mapping constraints. Currently the following may be specified with respect to shapeless objects:

1) Size: For any resulting location/shape to be acceptable it may be required to be less than, greater than or equal to some magnitude (number of addresses).
2) Contiguity: For a result to be acceptable it may be necessary that the set of addresses describing an ultimate location is in some sense contiguous. In addition there may be a requirement that the contiguity holds over a particular set of addresses.

Using the scripting language instances are defined using syntax of the form indicated in Table 2. Note that, from the above, not all attributes are appropriate to all object types (this is not evident from the simplified syntax fragment given in Table 2).

5. Constraints

Given a spatial application, and using the proposed system, the relationships that are desired to exist between objects are expressed in terms of constraints on the possible locations for those objects. Constraints comprise two operands and an infix operator linking the two operands. Each operand, in turn, comprises one or more object locations (typically identified by an object name). Satisfaction of a constraint will result in one or more pairs of compatible object locations according to the nature of the constraint. For example given two objects, `X` and `Y`, each of which has three locations associated with it, then there are nine possible pairs of locations of which one or more may satisfy the constraint. Thus, as a result of satisfying a constraint, the maximum number of solutions that may result is equivalent to the size of the Cartesian product of the two groups of

```
<INSTANCEDECS>   : <INSTANCEDEC>
                 | <INSTANCEDEC> <INSTANCEDECS>
                 ;

<INSTANCEDEC>    : 'instance(' <ID> ',' CLASSNAME ').' ;
                 | 'instance(' <ID> ',' CLASSNAME ',' <LOCATION> ').'
                 | 'instance(' <ID> ',' CLASSNAME ',' <LOCATION> ','
                                <ATTRIBUTES> ').'
                 ;

<ATTRIBUTES>     : <ATTRIBUTE>
                 | <ATTRIBUTE> ','<ATTRIBUTES>
                 ;

<ATTRIBUTE>      : 'subShape(' <SETTA> ')'
                 | 'rotation()'
                 | 'include(' <ID> ')'
                 | 'exclude(' <ID> ')'
                 | 'size(' <OPERATOR> N ')'
                 | 'contiguous(' ')'

<ID>             : OBJECT
                 | <SETTA>
                 ;

<LOCATION>       : '[' <SETTA> ']' ;
```

Table 2: *Instance declaration syntax*

locations. In the following section, where the operation of the system is discussed, the efficiency concerns associated with the generation of a large number of solutions are addressed. Note that if no pair of locations satisfies the constraint the constraint is considered to have failed and consequently no solution can be generated for the given problem in its current form.

Three categories of constraint are supported:

1) Filters: Constraints that consider each pair of locations associated with a constraint and only "pass" those that satisfy the constraint operator. Such constraints can only operate between fixed and/or free objects where definite locations for objects either exist or can be generated. Currently the system supports five filters: `equals`, `subset`, `superset`, `disjoint` and `intersects`. The expressiveness of the last can be increased by specifying the nature of the intersection to be less than, greater than or equal to some magnitude.
2) Mappings: Constraints that operate between shapeless objects and fixed or free objects and serve to prune the location space associated with the shapeless object by mapping some operation onto the location space with respect to the location(s) associated with the given fixed or free objects. The system supports two mapping operators `complement` and `intersects`. Again in the case of the intersection operator the nature of the intersection can be specified. On satisfaction of a mapping constraint there may be contiguity and/or size requirements associated with the referenced shapeless object in which case the resulting space must be validated with respect to these requirements before satisfaction of the constraint can be said to be complete.
3) Group filters: Similar to (1) except that the prefix or postfix operand comprises locations associated with more than one object.

Using constraints of the above form many of the standard temporal ([Allen, 1983], [Dechter, 1991], [Ladkin, 1992]) and topological ([Egenhofer, 1994], [Cohn, 1995], [Hernández 1991]) relations encountered in the spatio-temporal literature can be specified. The expressiveness of the above constraint system can be significantly increased by incorporating the notion of offsets. Offsets are sets of addresses that are applied to locations, associated with the operands for a constraint, prior to attempting to satisfy that constraint. In this manner we can identify spaces associated with locations (for example we can identify (say) the area to the south-west of some 2-D location) or we can expand locations either uniformly or in some direction. Offsets may be applied to all the addresses associated with a location or only to the *origin* for the shape definition. The latter is defined as the corner address, of the minimum surrounding orthogonal parallelepiped for the shape, nearest the origin of the object space.

Although the expressiveness of constraints is greatly increased using offsets, they cannot easily be used to identify shapes defined by uniformly "shrinking" a location. To this end a *shrink* option is also provided. In addition offsets may be further influenced by insisting that certain groups of addresses are included or excluded.

Using the scripting language constraints are specified as indicated in Table 3.

```
<CONSTRAINTDEC>       : <CONSTRAINT>
                      | <CONSTRAINT> <CONSTRAINTDEC>
                      ;

<CONSTRAINT>          : 'constraint(' <OPERAND> ',' OPERATOR ','
                                      <OPERAND> ').' ;

<OPERAND>             : <OBJECT>
                      | <OBJECT> ',' <OFFSETS>
                      | <SETTA>
                      | <SETTA> ',' <OFFSETS>
                      ;

<OFFSETS>             : <OFFSET>
                      | <OFFSET> <OFFSET_ATTRIBUTE>
                      ;

<OFFSET_ATTRIBUTE>    : 'include(' <ID> ')'
                      | 'exclude(' <ID> ')'
                      ;
```

Table 3: *Constraint declaration syntax.*

6 Operation

Scripts are passed to the reasoning system. This incorporates a lexical analyser and a parser which translates the script into a set of object and constraint data structures. The constraints are then used to verify object locations, discriminate between candidate locations for free objects and to "prune" the location space associated with shapeless objects. The result is stored in a solution tree which is constructed and destructed dynamically as the solution process progresses. Nodes in the tree represent alternatives encountered during the solution process where a constraint can be satisfied in more than one manner. This typically occurs when a constraint incorporates a free object that has more than one candidate location associated with it. Wherever a node occurs in the tree the solution process continues in a depth first manner. If a given spatial problem has only one solution the resulting solution tree will comprise only a single node.

The aim of the constraint satisfaction strategy used is to limit the growth of the solution tree. This is achieved through:

1) A constraint selection process which, on each iteration, seeks to minimise the number of candidate solutions that may result on satisfaction of the constraint.

2) A merging mechanism which, should more than one solution be generated, attempts to identify solutions that can co-exist within the restrictions of the given constraint and consequently reduce the number of branches that might otherwise be generated.

The constraint selection process commences by attempting to identify constraints that reference a fixed object or a free object which has only one candidate location associated with it. These constraints will, after appropriate merging of solutions, produce only a single result and will therefore not generate branches in the solution tree. During the processing of these "single solution" constraints it is likely that the number of candidate locations associated with affected free objects will be decreased, and that a significant amount of pruning of location spaces associated with free objects will have been undertaken. In addition, given a particular scenario, single solution constraints are the most critical - if such constraints cannot be satisfied no solution to the problem can be generated. The constraint selection strategy used thus also aides the early identification of fruitless branches in the solution tree. Once all single solution constraints have been addressed the remaining constraints will reference free and shapeless objects. Selections are then made according to the number of candidate locations associated with free objects. A free object which has (for example) two candidate solutions can, in the worst case, result in only two branches in the solution tree; while a free object which has (say) ten branches associated with it will, in the worst case, result in ten branches in the solution tree.

The research team have conducted experiments into the application of alternative mechanisms whereby fruitless branches within the tree can be identified early in the search process. Examples include dependency based backtracking and *a priori* pruning techniques such as forward checking and "look ahead" (see [Mackworth, 1977] and [Jaffar, 1986] for further discussion). However, it was found that the computational overheads involved were such that no advantage would be gained from the adoption of such techniques. The current, heuristically guided, pruning mechanism therefore seems to be the most appropriate. Similar approaches have been used in many well established problem solving systems, for example ALICE [Lauriere, 1978]. Whatever the case, the design of the SPARTA system is such that the constraint satisfaction mechanism is independent of the representation, so that the system is not tied to any particular Constraint Satisfaction Problem (CSP) solving approach.

7. Conclusions

A qualitative spatio-temporal reasoning system has been described founded on a tesseral representation and linearisation 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 suited to a wide range of applications.
3) It is conceptually simple and computationally effective.

These advantages are gained primarily as a consequence of the particular tesseral representation on which the mechanism is founded which allows for computationally efficient manipulation of sets of addresses to support the resolution process. An additional advantage of the representation is that is compatible with many existing raster image and GIS formats.

The resulting system has been found to have a great many applications and has been successfully targeted at a number areas, particularly Geographic Information Systems (GIS) ([Beattie, 1995]) and Environmental Impact Assessment (EIA) ([Beattie, 1996]). In addition the system has been used to address classic AI problems, such as shape fitting problems ([Coenen, 1997]), and also a number of further applications which do not generally fall under the remit of spatial reasoning, for example timetabling and scheduling ([Coenen, 1995]). This testing has demonstrated that the approach can be applied to scenarios involving any number of dimensions without incurring any corresponding computational overhead as the number of dimensions increases. Current work is focused on noise pollution modelling and assessment in the city of London, and marine electronic chart interaction.

References

[Allen, 1983]
J.F. Allen (1983). Maintaining knowledge about temporal intervals. Communications of the ACM, 26(11), pp832-843.

[Beattie, 1995]
B. Beattie, F.P. Coenen, T.J.M. Bench-Capon, B.M. Diaz and M.J.R. Shave (1995). Spatial Reasoning for GIS using a Tesseral Data Representation. In Revell, N. and Tjoa, A.M. a(Eds), Database and Expert Systems Applications, (Proceedings DEXA'95), Lecture Notes in Computer Science 978, Springer Verlag, pp207-216.

[Beattie, 1996]
B. Beattie, F.P. Coenen, Hough, A., T.J.M. Bench-Capon, Diaz, B.M. and M.J.R. Shave (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.

[Coenen, 1995]
F.P. Coenen, B. Beattie, T.J.M. Bench-Capon, Shave, M.J.R and B.M. Diaz (1995). Spatial Reasoning for Timetabling: The TIMETABLER system. Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling (ICPTAT'95), Napier University, Edinburgh, pp57-68.

[Coenen, 1996]
F.P. Coenen, B. Beattie, T.J.M. Bench-Capon, Shave, M.J.R and B.M. Diaz (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.

[Coenen, 1997]
F.P. Coenen, B. Beattie, T.J.M. Bench-Capon, B.M. Diaz and Shave. M.J.R. (1997). A tesseral Approach to N-Dimensional Spatial Reasoning. To be Presented at DEXA'97.

[Cohn, 1995]
A.G. Cohn, J.M. Gooday, B. Bennet and N.M. Gotts (1995). A logical approach to representing and reasoning about space, Artificial Intelligence Review, Vol 9, pp255-259.

Dechter, 1991]
R. Dechter, I. Meiri and J. Pearl (1991). Temporal constraint networks. Artificial Intelligence, Vol 49, pp61-95.

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

[Egenhofer, 1994]
M.J. Egenhofer (1994). Deriving the composition of binary topological relations. Journal of Visual Languages and Computing, Vol 5, pp133-149.

[Hernández, 1991]
D. Hernández (1991). 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, pp373-385.

[Jaffar, 1986]
J. Jaffar, J-L. Lasses and M.J. Maher (1986). A logic programming language scheme. In D. DeGroot and G. Linderstrom (Eds), Logic Programming, Relations, Functions and equations, Prentice-hall, Englewood Cliffs, NJ, USA.

[Ladkin, 1992]
P. Ladkin (1992). Effective solution of qualitative interval constraint problems. Artificial Intelligence, Vol 52, pp105-124.

[Laurier, 1978]
J-L. Lauriere (1978). A language and a program for stating and solving combinatorial problems. Artificial Intelligence, 10(1), pp29-127.

Mackworth, 1977]
A.K. Mackworth (1977). Consistency in networks of relations. AI Journal 8(1), pp99-118.