TESSERAL SPATIO-TEMPORAL REASONING FOR MULTI-DIMENSIONAL DATA

Frans Coenen

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

SUMMARY

A quantitative spatio-temporal reasoning mechanism founded on a tesseral representation of space is described. The principal advantages of this mechanism are that it is generally applicable to any N-dimensional space while at the same time providing for effective manipulation and comparison of objects located within such spaces. These advantages are a direct consequence of the derived tesseral representation on which the mechanism is founded. The defining features of this representation include:

- Efficient translation through N-dimensional space using conventional integer arithmetic.
- Computationally effective rotation.
- Linearisation of space providing for the application of one-dimensional (temporal) reasoning techniques regardless of the number of dimensions under consideration.
- Run line encoding.
- Simple relational comparison of N-dimensional objects.

In addition the 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 mechanism is incorporated into a demonstration spatio-temporal reasoning system, the SPARTA (SPAtial Reasoning using Tesseral Addressing) system. This currently operates using a constraint satisfaction approach to spatial problem solving supported by a heuristically guided constraint selection mechanism so as to minimise the search space. Spatial problems are passed to this system in the form of a script comprising a set of object descriptions (defined in terms of classes and instances of those classes) and a set of constraints defining the relationships "desired" to exist between pairs or groups of objects. Successful resolution of a script results in the production all solutions that satisfy the given constraints (assuming a solution exists). The resulting output can be either graphical or textual as directed by the user.

The approach has been applied to a number of practical application areas including; site identification for civil engineering projects, electronic chart interaction and noise pollution analysis. Other, more esoteric, applications that have been investigated include classic AI problems such as N-queens problems, and multi-dimensional shape fitting scenarios.

The paper gives a formal description of the representation and an overview of the SPARTA N-dimensional reasoning system and its associated constraint satisfaction mechanism. The paper is illustrated with implementations of Alan's *interval calculaus* and Egenhofer's *9-Intersection* approach to "topological" N-dimensional spatial reasoning; which serve to give a flavour of the power of the proposed approach.

TESSERAL SPATIO-TEMPORAL REASONING FOR MULTI-DIMENSIONAL DATA

Frans Coenen

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 generally applicable approach to N-dimensional spatial reasoning is described. The approach is founded on a unique representation based on ideas concerning "tesseral" addressing. This offers many computational advantages including minimal data storage, computationally efficient translation of data, and simple data comparison, regardless of the number of dimensions under consideration. The representation allows spatial attributes associated with objects to be expressed simply and concisely in terms of sets of addresses which can then be related using standard set operations expressed as constraints. The approach has been incorporated into a spatial reasoning system — the SPARTA (SPAtial Reasoning using Tesseral Addressing) system — which has been successfully used in conjunction with a significant number of spatial application domains.

Keywords: Spatio-Temporal Reasoning, Tesseral Addressing, N-Dimensional information processing.

1 INTRODUCTION

A versatile and generally applicable approach to multi-dimensional spatial reasoning is described. The approach is founded on a tesseral quantitative representation of space that offers advantages of both effective data storage, and computationally efficient translation and comparison of data regardless of the number of dimensions under consideration. The nature of the representation, which is in many respects a raster encoding, is such that it is immediately compatible with all applications where spatial objects are represented using linear formats. Examples of such representations include image encodings (GIF, PBM etc.), some Geographic Information System (GIS) data representations, the U.K. Admiralty Raster Chart System (ARCS), and 3-D visualisation representations such as DXF. In addition a tesseral reference can be viewed as both a raster label and a vector quantity. Consequently the representation can also be interfaced to vector representations such as those prevalent in GIS, and standards such as the DX90 international hydrographic exchange standard. Further the quantitative representation can be extended to describe qualitative aspects of spatial reasoning. Although for quantitative spatial reasoning a reference framework is essential, qualitative reasoning still requires a reference framework to express relations such as "before" or "inFrontOf". In a purely topological qualitative approach no reference framework is required; however this can still be defined in terms of such a framework.

The tesseral approach to spatial reasoning described here has been incorporated into a general purpose spatial reasoning system — the SPARTA (SPAtial Reasoning using Tesseral Addressing) system. SPARTA operates using a constraint satisfaction approach to spatial problem solving supported by a heuristically guided constraint selection mechanism so as to minimise the search space. 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. Solutions are output in either graphical or textual formats as directed by the user. The general applicability offered by the 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 or vector), or as a primary representation in its own right.

Applications that have been investigated using the technique include; Geographic Information Systems (GIS) (Beattie et al.[1], Coenen et al[2]), noise pollution monitoring (Brown et al.[3]), environmental impact assessment (Beattie et al.[4]), shape fitting (Coenen et al.[5]), scheduling and timetabling (Coenen et al.[6]), rule base verification and validation (Coenen[7]) and also a number of "AI problems" such as the N-queens problem.

The rest of this paper is organised as follows. Full details of the unique tesseral representation on which the approach is founded are given in Section 2. In Section 3 the SPARTA system is introduced and a brief overview of the nature of the scripting language presented. The definition of the attributes that may be associated with spatial objects and the constraints that may link such objects are then described in Sections 4 and 5 respectively. Some notes concerning the constraint satisfaction mechanism are then given in Sections 6. In the following two sections the approach is illustrated firstly (Section 7) with respect to Allen's interval calculus[8] and secondly (Section 8) with respect to Egenhofer's 9-Intersection mechanism[9]. It should be noted that these two illustrations are of necessity brief and therefore only serve to give a flavour of the full power of the approach. Finally in section 9 some concluding remarks are presented.

1.1 NOTE ON SPATIAL REASONING

Spatial reasoning can be defined, very broadly, as the automated manipulation of data objects belonging to spatial domains so as to arrive at application dependent conclusions. A spatial domain, in this context, is considered to imply any N-dimensional space (real or imaginary, including 1-D temporal spaces) in which objects of interest may be contained. Automated manipulation implies the computer simulation of some higher mental processes (not just the simple response to some stimulus or the mechanical performance of an algorithm). Spatial reasoning is therefore not concerned with (say) the automated retrieval of spatially referenced data contained in some database format. At its simplest, spatial reasoning can be considered to revolve round the identification of (previously unknown) relationships that exist between spatial objects according to the relationships that are known or desired to exist between such objects. In more complex systems the identified relations are then used as the foundation whereby further reasoning can take place, and consequently additional conclusions drawn.

2 REPRESENTATION

The tesseral representation used assumes that multi-dimensional space comprises a set of N-dimensional isohedral (same shape) *cells* each of which is labelled with a unique *address* (see Subsection 2.1 for further detail on tesseral representations). 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). In such systems coordinates can be conceived of as having either a positive or negative sign; consequently both positive and negative space can be identified (where the latter is comprised of labels which include at least one negative coordinate). Thus the universe of discourse can be described as the set U of all possible labels defined as follows:

$$U = P \sqcup N$$

(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 application will then consist of some domain of discourse (U_D):

$$U_D = P_D \sqcup N_D$$

comprising a subset of U such that each dimension (D) is defined in terms of the set of its possible values:

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

(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.

*** **Figure 1 in here** ***

Figure 1: Bit Allocation

Although the Cartesian system offers the advantages that it is well understood and consequently widely accepted, the disadvantage is that 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 objects with more than two dimensions. The desire to overcome these disadvantages was the initial motivation for the particular tesseral representation described here; however the representation is much 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 $min..max$, 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 integral 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

***** Figure 2 in here *****

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

$base_n$ 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:

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

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:

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

The function can be implemented using appropriate left and right shift operations to isolate bit groupings.

2.1 Relationship to other tesseral representations

Tesseral representations are spatial representations which are derived by repeatedly dividing a space into isohedral sub-spaces until some predefined resolution is reached (Bell et al.[10], Diaz and Bell[11]). The resulting sub-spaces are then allocated (usually numeric) "addresses" in such a way as to reflect the hierarchical decomposition. The approach is founded on ideas concerning hierarchical clustering developed in the 1960s and 1970s to improve data access times (Morton[12]), and atomic isohedral tiling strategies developed in the 1970s and 1980s concerned with group theory (Grunbaum and Shephard[13]). These two strands were brought together in the early 1980s when a *tesseral arithmetic* was defined for the representation (Holroyd[14]). There are many variations on the tesseral theme of which the "quad-tesseral" approach is arguably the most popular (Samet[15], Gargantini[16]).

The representation described here is categorised as a tesseral representation because it is a mechanism for addressing spaces decomposed into isohedral subspaces, but at some predefined resolution. There is no explicit inclusion of the concept of hierarchical decomposition and consequently this is not reflected in the addressing system. This method of deriving the decomposition is the feature which sets the described representation apart from all other tesseral representations. The representation also offers a number of significant advantages over other tesseral representations (as well as Cartesian approaches) as follows:

1. All references are unique and conceptually simple to generate.

2. It results in a "left-to-right" linearisation of space (see Figure 2) which in turn offers advantages of:

    - General applicability to any number of dimensions without requiring alteration of program code or recourse to procedures and functions dedicated to a particular number of dimensions (as in the case of systems founded on Cartesian approaches) — all spaces are treated in one dimensional terms.
    - Efficient data storage using ideas founded on "run-line" encoding strategies (see note in Sub-section 2.2).
    - Computationally effective comparison of sets of addresses (see Sub-section 2.4 for further comment).
    - Predictability, i.e. given a location it is always possible to predict the addresses of physically adjacent cells (this is not the case with tesseral approaches that adopt a Morton or Hilbert linearisation).

3. It 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).

4. It p+rovides for the rotation of objects.

5. If desired, addresses can be *efficiently* converted to and from Cartesian tuples as described above (not the case with other tesseral approaches).

6. 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.

***** Figure 3 in here *****

Figure 3: 3-D example shape

2.2 Data storage

In Section 2 the representation was described in terms of an approach to data compression. However, a much greater degree of storage efficiency can be achieved by taking into consideration the linearisation that results from the encoding. Linearisation is a feature of "all" tesseral representations although in most cases the linearisation follows a Morton sequence or a Hilbert curve, which is not as intuitively obvious as the left to right linearisation associated with the SPARTA representation. Whatever the case, "run-line" techniques can be used to store sequences of addresses in terms of a *start* and an *end* address. In the case of the SPARTA representation this would result in the definition of spatial attributes (such as location and shape) in terms of a sequence of "bars". This is adequate for one and two dimensional spaces but results in unacceptable overheads when higher dimensional spaces ($N > 2$) are considered. Consequently an alternative approach has been adopted here whereby spatial attributes are expressed in terms of N-dimensional "boxes", referred to as *N-cubes*, defined in terms of a start and an end address. The start address is defined as the reference geometrically nearest to the origin, and the end address that geometrically furthest away. For example consider the 3-D "plus" shape shown in Figure 3. In the context of our 3-D example encoding this might be defined as follows:

```
{9..18, 65..146, 72..144, 75..147, 201..210}
```

Note that sequences are indicated using a ".."operator and that they are ordered according to the numeric location of their start address within the overall linearisation. The shape in Figure 3 is thus defined in terms of 10 integers. Using a "bar" encoding this would have required 24 integers. Using a "linear quad-tesseral" encoding (with a Morton linearisation) 36 integers (18 sequences) would be required. Without any run-line encoding 64 integers (tesseral addresses) would be needed, and using a Cartesian system 192 integers (64 3-tuples). For the particular shape shown in the figure a "hierarchical quad-tesseral" representation, where the space is decomposed at different resolutions according to the required application, would offer no advantages over the "linear quad-tesseral" version. The N-cube encoding thus offers significant data storage advantages over other related representations. Although it is difficult to quantify this advantage as some types of "shape" are more advantageous to some encodings than others, the above gives an indication of the degree of saving that can be made. In addition the advantage in space saving gained increases with the number of dimensions under consideration.

2.3 Translation

A further important advantage of the SPARTA representation is the computational ease with which tesserally defined "shapes" can be translated (moved) through a space U_D. For example, considering the space presented in Figure 2, to move one "cell" to the right from any given reference the address 1 is added to the reference. To move to the left the address 1 is subtracted. To move diagonally (say) one space to the right and two upwards 17 is added (subtracted to move in the opposite direction). To move diagonally (say) one space to the left and two upwards a negative address (15) is required. It is for this reason (to allow translation in all directions) that the representation must include the negative equivalents of any space P_D of interest.

Thus tesserally defined shapes can be translated in any direction, by any distance, using standard integer addition and subtraction. Further the addition/subtraction need only be applied to the start and end references for each sequence. Also, a given shape can be expanded along a particular axis and/or in any particular direction. For example, using the given example space, to expand a single cell in all directions the sequence $-73..73$ would be added.

A translation function $f_{translate}$ is thus defined as follows:

***** Figure 4 in here *****

Figure 4: 2-D example shape

$$
\begin{aligned}
Dom(f_{translate} &= \{< S_1, \{c..d\} >|\ S_1 = \{a..b \mid a, b \in R \wedge R \subset D_U\} \ \cdot \ c, d \in D_U\} \\
Cod(f_{translate} &= \{S_2 \mid S_1 = \{e..f \mid e, f \in R \wedge R \subset D_U\} \\
Gr(f_{translate} &= \{< S_1, \{c..d\}, S_2 >|\ \forall a..b \in S_1 \ \cdot \ \exists e..f \in S_2 \ \textit{such that} \\
&\quad e = a + c \wedge f = b + d\}
\end{aligned}
$$

The implementation of this function requires the inclusion of a mechanism to deal with the situation where an attribute is wholly or partially translated out of U_D. The operation of this function can best be defined by considering the 2-D example "H" shape given in Figure 4 which might be encoded as follows:

$$S_h = \{-2..14, 2..18, 7..9\}$$

To move this shape one cell to the left and two cells downwards the set $S_t = \{-17.. -17\}$. must be added:

$$f_{translate}(S_h, S_t) \rightarrow \{-19.. -3, 15..1, -10.. -8\}$$

Similarly if the "H" is to be expanded in both directions along the D_1 axis by one cell then set $S_t = \{-1..1\}$:

$$f_{translate}(S_h, S_t) \rightarrow \{-1..15, 1..19, 6..10\}$$

Alternatively to expand the "H" in all 3 directions (in our example space) $S_t = \{-73..73\}$.

2.4 Comparison

Knowledge of the linearisation can also be used to advantage when comparing sets of addresses. For example given the sets of addresses S_1 and S_2, the validity of the Boolean relation S_1 *subset* S_2 can be established as follows:

$$S_1 \ subset \ S_2 \Leftrightarrow \forall m..n \in S_1 \ \cdot \ \exists s..t \in S_2 \ such \ that \ m \leq s \wedge n \leq t$$

The relation S_1 *subset* S_2 is true if and only if for all N-cubes $m..n$ in the set S_1 there exists a N-cube $s..t$ in the set S_2 such that $m..n$ is within $s..t$. Thus to compare sets of addresses where the elements are defined in terms of N-cubes we do not have to consider individual elements. Again, this advantage is of particular relevance when dealing with spaces comprising more than two dimensions.

3 SPATIAL APPLICATION DESCRIPTION

Using the representation spatial attributes associated with objects can be defined in terms of sets of N-cubes. The relationships that are desired to exist between objects may then be defined in terms of constraints linking attributes associated with pairs of objects. With respect to the SPARTA system such descriptions are expressed as a sequence of PROLOG facts contained in a *script*. A script typically comprises four sections: a space definition, object descriptions defined in terms of classes and instances of those classes, and constraint descriptions.

The space definition describes the limits of the set (U_D) in terms of the positive maximum coordinate for each dimension:

$$space(\{max_1, max_2, ..., max_n\}).$$

The space of interest, i.e the space within which the objects pertaining to the application under consideration exist, is then considered to be equivalent to the positive part of this space (P_D). The negative part of this space (N_D) is required when translating objects through the space (see Subsection 2.3 above).

A class definition takes one of the following two formats:

$$class(ClassName, ObjectType).$$

$$class(ClassName, ObjectType, ShapeDef).$$

Every class has a name and a type definition for the objects belonging to that class. Currently three object "types" are supported:

- *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 can be moved).
- *Shapeless objects* which have no given shape or location but are known to exist somewhere within the space P_D.

Only free objects can have a shape definition associated with them (in the case of fixed objects this is implied by the location, in the case of shapeless objects this is unknown).

An instance definition then takes the form:

$$instance(InstanceName, ClassName, Attribute_1, Attribute_2, ..., Attribute_3).$$

Possible attributes that can be associated with any given instance include:

- Location or location space
- Shape
- Rotation
- Size (actual, maximum, minimum)
- Contiguity (a spatial entity does not have to be comprised of a continuous set of cells)

although not every attribute is applicable to every class of object. The distinction between a location and a location space is that the first describes the precise location of a fixed object, while

ATTRIBUTES	**FIXED**	**FREE**	**SHAPELESS**
Location/location space	$\surd$	$\times$	$\times$
Shape	$\times$	$\surd$	$\times$
Rotation	$\times$	$\surd$	$\times$
Size	$\times$	$\times$	$\surd$
Contiguity	$\times$	$\times$	$\surd$

Table 1: Possible attribute definitions for different object types

the second describes a set of addresses somewhere within which a free or shapeless objec6t is known to exist. The attributes that may be defined for each type of object are listed in Table 1. Further detail concerning each of these attributes are given in the following section.

Constraints (at their simplest) are of the form:

$$constraint(Operand1, Relation, Operand2).$$

The operands are locations or location spaces associated with single instances, list of instances or entire classes of instances and even lists of classes; indicated using the appropriate instance or class names. The possible relations that can be used to link the operands are the standard set operations. The nature of constraints is described further in Section 5.

4 OBJECT ATTRIBUTES

In this section the nature of the attributes listed in the foregoing section are described in further detail.

4.1 Location

Other than an identifier the second most important attribute that may be associated with a spatial entity is its location. In the case of a fixed object this is precisely known. This is not the case for free or shapeless object where only a general *location space* in which the object may exist is known. Whatever the case the location or location space is described in terms of some subset ($Location$ or $LocationSpace$) of P_D:

$$\begin{aligned} Location &= \{l \mid l \in L \wedge L \subseteq P_D\} \\ LocationSpace &= \{l \mid l \in L \wedge L \subseteq P_D\} \end{aligned}$$

The definitions are the same, but the interpretations are different. In the absence of any other information $LocationSpace$ is assumed to be equal to P_D. Note also that in the case of a fixed object the nature of the set $Location$ also implies the nature of all of the other attributes associated with such an object, and hence there is no need to define these attributes (see Table 1).

To support the manipulation of objects every location (location space) has a *local origin* associated with it. This is defined as the corner address of the minimum bounding N-cube surrounding the entire location definition that is geometrically closest to the global origin for the given set U_D.

4.2 Shape

Shape is arguably the third most significant attribute that a spatial entity has. In the case of a fixed object this is defined by implication, whereas in the case of a free object shape it is defined in terms of a set $Shape$:

$$Shape = \{s \mid s \in S \wedge S \subseteq U_D \wedge globalOrigin \in S\}$$

Note that in this case the shape set is a subset of U_D as opposed to P_D as stipulated for $Location$ and $LocationSpace$ sets. Note also that the global origin must be included in the shape definition — this then acts as the reference address with respect to which the set can be manipulated.

Given a free object, the shape definition also implies further attributes such as size and contiguity. Note also that a free object must have some location space associated with it — somewhere within which it is known to exist. From Sub-section 4.1 this may be either the entire space P_D or some subset of P_D.

Given a set *Shape* and a set *LocationSpace* a set of sets *LocationCandidates* (the set of candidate location sets) is derived using a function $f_{candidateLocations}$:

$$\begin{aligned}
Don(f_{candidateLocations}) &= \{< L_s, S >| \ L_s = LocationSpace \ \cdot \ S = Shape\} \\
Cod(f_{candidateLocations}) &= \{L_c \mid L_c \subseteq L_s\} \\
Gr(f_{candidateLocations}) &= \{< L_s, S, L_c >| \ L_c = \bigcup(\forall x \in L_s \ \cdot \ f_{translate}(S, \{x..x\}) \subseteq L_s)\}
\end{aligned}$$

Note that the above would not identify "all" the candidate locations if the zero address (the global origin for U_D) was not required to be included in the definition for the set *Shape*.

4.3 Rotation

A further important attribute associated with free objects is orientation — given a particular object this may be either fixed or free to rotate. Whether a shape/object can be rotated or not is defined in terms of a constant *rotation* which may be included in the predicate defining the object in question.

4.4 Size

In the absence of location and shape information knowledge may be available concerning size. The size of a spatial object is expressed in terms of the number of cells (not N-cubes) in the set describing its shape, i.e. the *cardinality* of that set. In the case of a fixed object this will be equivalent to $f_{cardinality}(Location)$; where the function $f_{cardinality}$ returns the "size" of its argument which must be a set of N-cubes. In the case of a free object this will be equivalent to $f_{cardinality}(Shape)$. In the case of a shapeless object, where the precise definition of these sets is not known the required number of elements for the set *Location*/*Shape* can still be expressed. The minimum size of any object (that can be physically realised) is 1 indicating that it is represented by a single cell. The maximum size of an object is equivalent to $f_{cardinality}(P_D)$ — otherwise the object exceeds the application domain. In addition the nature of the size of an object can be expressed in terms of (i) a minimum size, (ii) a maximum size, or (iii) an actual size — the set of operators $\{<,=,>\}$. Thus a set $Size$, comprising a single 2-tuple, is defined as follows:

$$Size = \{\langle q,m\rangle \mid q \in \{<,=,>\} \;\cdot\; m \in \{1 \,..\, cardinality(P_D)\}\}$$

4.5 Contiguity

Finally something may be known about whether an object's location/shape is represented by a contiguous set of addresses or not. In this context contiguity is defined as the situation where each cell of a *Location* or *shape* associated with any object is adjacent to at least one other element of this set (adjacency is assumed to mean either edge or corner adjacency). The nature of the connectivity attribute is defined in terms of a constant *contiguous* which may be included in the predicate defining a particular object.

5 CONSTRAINTS

Form section 3 constraints are used to link pairs of object locations (where locations are indicated using instance names, list of such names, class names or lists of class names). Given that locations are described in terms of sets of N-cubes it is natural to consider relations in terms of the standard operators found in set theory: equals ($=$), intersection ($\cap$), union ($\cup$), subset ($\subseteq$), superset ($\supseteq$), "proper" subset ($\subset$) and "proper" superset ($\supset$); and also the negation of these relations. In the context of spatial reasoning these can be incorporated into Boolean functions (functions that return true or false) and non-Boolean functions (functions that return a "new" set) depending on the nature of the relation. With respect to the SPARTA system the Boolean functions are incorporated into constraints that act as filters to "test" location pairs. Eight such filters are supported:

$$\begin{aligned} constraint(A, filterEquals, B) &= f_{filterEquals}(A,B) \Rightarrow true\ if\ A = B \\ constraint(A, filterNotEquals, B) &= f_{filterNotEquals}(A,B) \Rightarrow true\ if\ A \neq B \end{aligned}$$

$$
\begin{aligned}
constraint(A, filterIntersects, B) &= f_{filterIntersects}(A, B) \Rightarrow true\ if\ A \cap B \neq \emptyset \\
constraint(A, filterNotIntersects, B) &= f_{filterNotIntersects}(A, B) \Rightarrow true\ if\ A \cap B = \emptyset \\
constraint(A, filterSubset, B) &= f_{filterSubset}(A, B) \Rightarrow true\ if\ A \subset B \\
constraint(A, filterNotSubset, B) &= f_{filterNotSubset}(A, B) \Rightarrow true\ if\ A \not\subset B \\
constraint(A, filterSuperset, B) &= f_{filterSuperset}(A, B) \Rightarrow true\ if\ A \supset B \\
constraint(A, filterNotSuperset, B) &= f_{filterNotSuperset}(A, B) \Rightarrow true\ if\ A \not\supset B
\end{aligned}
$$

No distinction is made between a "proper" subset (superset) and a subset (superset) because no applications have been found (to date) where the distinction is significant. Boolean filters are thus used to test locations for fixed objects and/or candidate locations for free objects against each other. Note that the implementation of the above functions makes full use of knowledge of the linearisation as described in Subsection 2.4.

Non-Boolean functions are incorporated into constraints that act as mappings in the sense that they "map" a non-Boolean set operation (intersection, union, complement) on to the prefix operand with respect to the postfix operand to produce a new set. In the context of the SPARTA system the prefix operand must represent a location space associated with a shapeless object and the postfix operand either a fixed location associated with a fixed object or a candidate location associated with a free object. The function then returns a revised location space fpr the prefix operand. Only two mapping constraints are supported:

$$
\begin{aligned}
constraint(A, mapIntersects, B) &= {}_{mapIntersects}(A, B) \Rightarrow A' = A \cap B \\
constraint(A, mapComplement, B) &= {}_{mapComplement}(A, B) \Rightarrow A' = B \setminus A
\end{aligned}
$$

For procedural reasons it is desirable to maintain a declarative reading of constraints (i.e. constraint interpretation should be independent of any ordering in which they might be presented) Thus a union mapping constraint is not supported as this would have the effect of increasing the location space associated with a shapeless object and consequently the final output would be influenced by the ordering in which constraints are processed, i.e. the sequence in which constraints are presented becomes significant.

Using the above filters and mappings it is possible to express many of the standard relations encountered in spatial reasoning applications (such as those discussed by Retz-Schmidt[17], Egenhofer[9] and Cohn[18] amongst many others). However, to increase the expressiveness of these constraints an offset can be applied to the location associated with an identifier prior to satisfaction of the constraint. An offset is some subset of U_D which is applied (using the $f_{translate}$ function) to either (a) all the elements describing a location, location space or set of candidate locations or (b) the local origin for such sets. The effect in the first case is to translate or expand (or both) the space as illustrated in Subsection 2.3. In the second case this provides the means whereby other "locations" may be defined with respect to the given location (location space or candidate location). A facility is also provided to "shrink" a location by a number of cells either uniformly or along specific axes. Whatever the case offsets are defined by a predicate $offset$ as follows:

$$offset(Q, P)$$

where:

$$\begin{array}{rcl} Q & = & \{q | q \in \{ref, all, shrink\}\} \\ P & = & \{p | pinR \wedge R \subseteq U_D\} \end{array}$$

Thus a constraint is defined using one of the following formats:

$$constraint(Operand1, Relation, Operand2).$$

$$constraint(Operand1, Offset1, Relation, Operand2).$$

$$constraint(Operand1, Relation, Operand2, Offset2).$$

$$constraint(Operand1, Offset1, Relation, Operand2, Offset2).$$

Finally, the expressiveness of the *intersects* relations (either in addition to or as an an alternative to the use of offsets) may be further expanded by quantifying the size of the intersection. This is achieved by allowing two optional arguments for the intersection relation — an operator and a desired cardinality. The set of possible operators is as follows:

$$<, \leq, =, \geq, >$$

The cardinality is expressed as a positive integer in the range of 1 to $f_{cardinality}(P_D)$. Thus we can insist that an intersection is (say) equal to a cardinality of 4 as follows:

$$constraint(A, filterIntersects(=, 4), B).$$

6 THE CONSTRAINT SATISFACTION PROCESS

There are many computer applications where a solution is required to be generated which conforms to a set of constraints. Examples include: machine vision, belief maintenance, scheduling and time-tabling, graph colouring, logic puzzles, floor plan design and circuit design. This class of applications is often referred to as Constraint Satisfaction Problems (CSP) (van Hentenryck[19]). Temporal and spatial reasoning problems also fall into the category of CSP. More formally a CSP can be defined in terms of a set of variables $I = \{X1, X2, .., Xn\}$ which can take values from the finite domains $I = \{D1, D2, .., Dn\}$ respectively, and a set of constraints which specify which values are compatible with each other. Thus a constraint of the form $constraint(Xi1, Xi2, .., Xik)$ between k variables from I is a subset of the Cartesian product $\{Di1 \times Di2 \times Dik\}$. A solution to a CSP is then an assignment of values to all variables which satisfy all constraints. Given that the domains $\{D1, D2, .., Dn\}$ are finite, a solution (or solutions) can always be found provided that constraints are not contradictory. The real problem associated with CSP is that of complexity (CSP are *NP-complete*) and, as a result, efficiency.

The solution to CSP is typically founded using some tree searching strategy supported by a mechanism whereby fruitless branches within the tree can be identified early in the search process. Examples included dependency based backtracking and *a priori* pruning techniques such as forward checking and "look ahead" (Mackworth[20]). Given a particular CSP the nature of the problem can be further exploited to achieve additional pruning by guiding the search using application dependent heuristics. This was the idea behind a number of problem solving systems such as ALICE (Lauriere[21]).

CSP tree searching strategies are well suited to encoding using logic programming languages (e.g. PROLOG). However, the current disadvantage of such languages is that their execution is very inefficient. Search procedures (i.e. backtracking) are based on recovery from failure rather than avoiding failure. Consequently a number of constraint (logic) programming languages have been developed to address the disadvantages associated with standard logic languages. One example is CHIP (Constraint Handling in PROLOG) which extends standard PROLOG to include the concepts of finite domains, Boolean terms and relational terms (Dincbas et al.[22]). Further examples include: PROLOG III (Colmerauer[23]), and CLP (Constraint Logic Programming) which is a framework for constraint handling in logic (Jaffar and Michaylov[24]), which in turn is an extension of the earlier logic programming language "Scheme" (Jaffar et al.[25]). Alternatively, many researchers have resorted to implementing CSP solution strategies using more traditional imperative programming languages. The current version of the SPARTA system is written in C.

The design of the SPARTA system is such that the constraint satisfaction mechanism is independent of the representation, consequently the system is not tied to any particular approach to CSP solving. However, the current version of the system uses a heuristically guided depth first search strategy. 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* corresponding to P_D containing no objects. A constraint is then selected from the constraints to do list following the constraint selection criteria (see Section 6.1). The system then attempts to resolve this constraint with three possible outcomes:

1. The constraint cannot be satisfied in which case (at this stage) it is concluded that no appropriate configuration of objects can be found and therefore the satisfaction process is stopped.

2. One or more compatible solutions exist therefore include the referenced objects in the object space, adjust the constraints to do list and select another constraint. Note that two solutions are compatible if the current constraint will continue to hold if the solutions are combined. This will be the case where one of the operands is a fixed object. Where both operands are free objects it is likely that solutions will not be compatible.

3. A number (more than one) of non compatible solutions is produced therefore create a number of branches, equivalent to the number of solution, emanating from the current node each 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 (non-compatible) solution. Whenever an additional level in the tree is created each branch is processed in a “depth-first” manner 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 result is output and the current node removed from the tree, the system then 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. Consequently the solution tree in its entirety never exists, only the current branch and those higher level nodes which merit further investigation.

6.1 Constraint selection strategy

In practice constraints are processed in a sequential fashion. This implies an ordering and consequently a selection strategy. However, from a user perspective, it is desirable that constraints can be presented regardless of any ordering requirement. Further, from an implementational perspective, it is desirable that constraints are processed in a computationally efficient manner and that the growth of the solution tree is minimised. This is achieved by delaying the satisfaction of constraints that may cause the generation of a new level in the solution tree for as long as possible. Constraints are therefore selected according to the following sequence:

1. All constraints relating two fixed objects or a fixed object and a shapeless object. Fixed and shapeless objects have only one possible location/location space associated with them and therefore are the most vulnerable constraints, i.e. those constraints that are hardest to satisfy.

2. Constraints linking free objects to fixed objects. Although free objects may have many candidate locations, when related to a fixed object (which by definition can have only one location) only one set of compatible solutions can be produced. Constraints are selected here according to the number of candidate locations associated with each free object (calculated using the $f_{candidateLocations}$ function described in Subsection 4.2. Constraints involving free objects with a low candidate location set cardinality are more vulnerable than those with a higher cardinality.

3. The remaining constraints linking free objects to shapeless objects, or free objects to one another, again according to the cardinality associated with the operands (a shapeless object's location space has a cardinality of 1). The product of the two cardinalities then dictates the maximum number of branches that may be produced on satisfaction of the constraint, although it may be that on satisfaction some or all of the solutions are compatible. Constraints are thus selected so as to minimise $cardinality_1 \times cardinality_2$.

The above constraint selection strategy thus has the effect of limiting the growth of the search tree while at the same time causing the most vulnerable constraints to be procesed first.

***** Figure 5 in here *****

Figure 5: Allen's 13 interval relations

7 EXAMPLE 1: TEMPORAL REASONING USING ALLEN'S INTERVAL CALCULUS APPROACH

The most influential modern work carried out in the field of temporal reasoning can be considered to be James Allen's Interval Calculus. Although Allen's work cannot be said to be definitive, its importance lies in that it is the basis on which many subsequent temporal (and spatial) reasoning systems have been built, or if not the basis at least the catalyst for such work. For example Freksa[26], Hernández[27] and Mukerjee and Joe[28] have all transferred the approach to the spatial domain, while Ligozat[29] has generalised the interval concept for reasoning with chains of events. Good and available accounts of Allen's original work can be found in [30] and [31] (see also Allen and Koomen[32]) and a more general overview in [8]. The work has subsequently been extended to incorporate points as well as intervals (see Allen and Hayes[33,34,35]). A good critique of Allen's work can be found in Galton[36], while Vilain et al.[37,38] and Nokel[39] studied the computational complexity of Allen's reasoning scheme and some of its variants.

Allen, in his interval calculus, identifies 13 "interval" relations which are used to link one dimensional objects (Figure 5). The objects are stored in a network where each arc represents one or more relations. The network is typically expressed as a set of tuples of the form:

$$< node1, relationList, node2 >$$

By adding further tuples to the network we can check the continuing consistency of the network, remove possible relations which are no longer applicable (provided that at least one relation always remains linking each node pair), or deduce further relations not previously included in the network. Allen uses a *transitivity* table technique to achieve these revisions. Using the SPARTA representation Allen's 13 interval relations can be expressed as follows (SPARTA relations given to the right):

$$
\begin{aligned}
a\ before\ b &\Leftrightarrow constraint(a, filterSubset, b, offset(ref, \{-max_{D_1}\ ..\ -1\})).\\
a\ after\ b &\Leftrightarrow constraint(a, filterSubset, b, offset(ref, \{(length+1)\ ..\ max_{D_1}\})).\\
a\ during\ b &\Leftrightarrow constraint(a, filterSubset, b).\\
a\ contains\ b &\Leftrightarrow constraint(a, filterSuperset, b).\\
a\ overlaps\ b &\Leftrightarrow constraint(a, filterIntersects, b).\\
&\ constraint(a, offset(ref, \{0\}), filterNotSubset, b).\\
a\ overlappedby\ b &\Leftrightarrow constraint(a, filterIntersects, b).\\
&\ constraint(a, filterNotSubset, b, offset(ref, \{length\})).\\
a\ meets\ b &\Leftrightarrow constraint(a, offset(ref, \{length\}), filterEquals, b, offset(ref, \{-1\})).\\
a\ metby\ b &\Leftrightarrow constraint(a, offset(ref, \{-1\}), filterEquals, b, offset(ref, \{length\})).\\
a\ starts\ b &\Leftrightarrow constraint(a, offset(ref, \{0\}), filterEquals, b, offfest(ref, \{0\})).\\
&\ constraint(a, filterSubset, b).\\
a\ startedby\ b &\Leftrightarrow constraint(a, offset(ref, \{0\}), filterEquals, b, offset(ref, \{0\})).\\
&\ constraint(a, filterSuperset, b).\\
a\ finishes\ b &\Leftrightarrow constraint(a, offset(ref, \{length\}), filterEquals, b, offsert(ref, \{length\})).\\
&\ constraint(a, filterSuperset, b).\\
a\ finishedby\ b &\Leftrightarrow constraint(a, offset(ref, \{length\}), filterEquals, b, offset(ref, \{length\})).\\
&\ constraint(a, filterSubset, b).\\
a\ equals\ b &\Leftrightarrow constraint(a, filterEquals, b).
\end{aligned}
$$

where max_{D_1} is the maximal value for the dimension D_1, and the identifier *length* indicates the "duration" of the interval in question. Note also that the application of $offset(ref, \{0\})$ has the effect of isolating the local origin of the set referred to.

Thus, given the relations *a startedBy b* and *b overlappedBy c*, using Allen's transitivity table technique, *c overlappedBy a* can be deduced. This can be expressed in the form of a SPARTA script as follows:

```
space(MaxD1).

class(interval,free,S).

instance(a,interval).
instance(b,interval).
instance(c,interval).

constraint(a,offset(ref,{0}),equals,b,offset(ref,{0})).
constraint(a,superset,b).
constraint(b,intersects,c).
constraint(c,notSubset,b,offset(ref,{length}).
```

Note that because a quantitative representation is used the scenario has been dimensioned. Thus we have assumed a 1-D application domain comprising $MaxD1$ addresses and defined each interval as a free object with some shape S which is contained in a location space which has a default value equivalent to P_D. Using the SPARTA system all possible configurations for the set of objects will be returned. In the above case each configuration will also illustrate the possible relationships between the instances a and c, i.e. c overlapped by a.

8 EXAMPLE 2: TOPOLOGICAL SPATIAL REASONING USING EGENHOFER'S 9-INTERSECTION APPROACH

The most obvious approach to spatial reasoning is to extend established temporal reasoning techniques so as to encompass more than one dimension. For example it is possible to consider that the relations that exist in 1-D space exist along all axes of interest in an N-dimensional space. Individual relations are then defined by the intersection of any two 1-D relations. This approach has been considered by many authors such as Freksa[26], Hernández[27] and Puller and Egenhofer[40]. It is generally acknowledged that there are some 26 1-D relations (including Allen's 13 interval relations) that can exist between points, intervals, and points and intervals in a 1-D space (Hamblin[41]). Extending these relations to N dimensions results in an exponential increase in the number of relations as the number of dimensions increases. This is the principal reason why one dimensional reasoning techniques do not lend themselves to easy adaptation to higher dimensions. This has been recognised by many authors (see Hernández[27], Chang et al.[42] and Frank et al.[43] amongst many others), and much work has been done on techniques to address these concerns. One approach is to represent information about individual dimensions independently and reason about each dimension in isolation (Malik ad Binford[44]). Alternatively some authors differentiate between projection and orientation relations (Hernández[27]). However, although these techniques all serve to reduce the severity of the problem, the basic difficulty — that the complexity of the problem increases exponentially with the number of dimensions under consideration — has not been removed.

An alternative approach is to consider only *topological* relations. Egenhofer defines such relations as follows:

"those spatial relations that are invariant under topological transformations and, therefore, preserved if the objects are translated, rotated or scaled." (Egenhofer[9])

intersection (interior(A), interior(B))	intersection (interior(A), boundary(B))	intersection (interior(A), exterior(B))
intersection (boundary(A), interior(B))	intersection (boundary(A), boundary(B))	intersection (boundary(A), exterior(B))
intersection (exterior(A), interior(B))	intersection (exterior(A), boundary(B))	intersection (exterior(A), exterior(B))

Table 2: Egenhofer's 9-Intersection Table

Egenhofer uses a *9-Intersection* model (Egenhofer[9], Papadias et al.[45]) founded on an earlier *4-Intersection* model (Egenhofer[46]) to identify and manipulate such relations. In this model spatial objects are considered to have three parts (a) an interior, (b) a boundary and (c) an exterior. The topological relation between two "point sets" (sets of addresses), A and B, is then described by the nine intersections of the interior, boundary and exterior of A with those of B as demonstrated in Table 2.

Various topological invariants can be used to evaluate and characterise Egenhofer's 9-intersection model. However it can be shown that there are 2^9 (512) possible combinations (Egenhofer and Franzosa[47] of which only a small subset can be physically realised as follows:

- 8 between objects without holes (contiguous region objects).
- 18 between objects with holes.
- 33 between simple lines.
- 19 between an object without a hole and a line.

Considering only the relationships between contiguous region objects these can be labelled as shown in Figure 6. The labelling in the figure is applied to 2-D examples, however it is equally applicable to any number of dimensions. The 8 topological relations presented in Figure 6 can be expressed using the SPARTA representation as follows:

*** **Figure 6 in here** ***

Figure 6: The 8 "contiguous object" topological relations

$$
\begin{aligned}
a\ disjoint\ b &\Leftrightarrow constraint(a, offset(all, sw..se), filterNotIntersects, b).\\
a\ meets\ b &\Leftrightarrow constraint(a, offset(all, sw..se), filterIntersects, b).\\
& \quad\ \ constraint(a, filterNotIntersects, b).\\
a\ overlaps\ b &\Leftrightarrow constraint(a, filterIntersects, b).\\
& \quad\ \ constraint(a, notSubset, b).\\
a\ coveredby\ b &\Leftrightarrow constraint(a, filterSubset, b).\\
& \quad\ \ constraint(a, filterNotSubset, b, offsert(shrink, \{1\})).\\
a\ covers\ b &\Leftrightarrow constraint(a, filterSuperset, b).\\
& \quad\ \ constraint(a, offset(shrink, \{1\}), filterNotSuperset, b).\\
a\ inside\ b &\Leftrightarrow constraint(a, filterSubset, b, offset(shrink, \{1\})).\\
a\ contains\ b &\Leftrightarrow constraint(a, offset(shrink, \{1\}), filterSuperset, b).\\
a\ equals\ b &\Leftrightarrow constraint(a, equals, b).
\end{aligned}
$$

where the identifiers sw and se represent the addresses required to uniformly expand a location by a factor of one cell in all directions. The application of $offset(shrink, \{1\})$ has the opposite effect.

Given two 9-intersections, representing two topological relations between two pairs of objects a and b, and b and c, the 9-intersection for the relation linking a to c can be determined by deriving a combined 9-intersection from the two given 9-intersections. To this end Egenhofer specifies eight "inference" rules to describe the dependencies of the intersections so that given 9-intersections linking a to b and b to c, the 9-intersection linking a to c can be found. The significance here is that the relationships can be produced without recourse to a transitivity table (although the possible ways in which the above 8 relationships can be combined can be expressed using such a table).

Thus given the relations a *covers* b and b *overlaps* c, using Egenhofer's inference rules we can deduce that either, a *overlaps* c or a *covers* c or a *contains* b. We can express this using the SPARTA representation as follows:

```
space(maxD1,maxDN).

class(fixedRegion,fixed).
class(freeRegion,free,S).

instance(a,fixedRegion,L).
instance(b,freeRegion).
instance(c,freeRegion).

constraint(a,superset,b).
constraint(a,offset(shrink,{1}),notSuperset,b).
constraint(b,intersects,c).
constraint(b,notSubset,c).
```

Note that here, for illustrative purposes only, one of the instances has been defined as a fixed object located at L (this will have the effect of reducing the size of the search tree). The remaining two regions are then defined as free objects with shape S. When processed this script will produce all possible configurations for the objects, each configuration illustrating a possible relation between the objects a and c, i.e. that a overlaps, covers or contains c in this case.

The concept of topological relations forms an important part of spatio-temporal theory and has been adopted by many researchers working in the field. For example the spatial axiomisations of Clark[48,49], Cohn[50], Randel et al.[51,52,53] (see also Cui et al.[54]), and Vieu[55] are all directed at a topological conceptualisation of spatial relations.

9 Conclusions

A qualitative spatio-temporal reasoning mechanism 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 fully compatible with both raster and vector representations rendering it 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.

The proposed mechanism has been incorporated into a spatio-temporal reasoning system, the SPARTA system, which has been tested against a significant number spatial problem domains including GIS, environmental impact assessment and noise pollution. However, the approach is not complete. There are still many application areas, such as map and chart interaction, spatial simulation and environmental planning; which require further mechanisms to increase the expressiveness of the relations. With respect to the attributes that can be associated with spatial entities there are also still a number of variations on the given list that require further investigation. For example we can conceive of partially shaped objects in that we may know a minimum shape for the object. Thus there is still much work to do; however the work to date has resulted in a foundation which is already widely applicable.

10 ACKNOWLEDGEMENTS

Early work on the SPARTA systems was carried out as part of the dGKBIS (dynamic Geographic Knowledge Based Information Systems) project (`http://www.csc.liv.ac.uk/~frans/dGKBIS/dGKBIS.html`) funded by the UK Engineering and Physical Sciences Research Council. The author is indebted to the ground breaking work of the original research team involved in this project, namely: Bridget Beattie, Trevor Bench-Capon, Bernard (Diz) Diaz and Michael Shave. The author would also like to thank Paul Leng for his helpful comments on an earlier version of this paper.

References

[1] 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 Revell, N. and Tjoa, A.M. (Eds), *Database and Expert Systems Applications* (Proceedings DEXA'95), Lecture Notes in Computer Science 978, Springer Verlag, 207-216 (1995).

[2] F.P. Coenen, B. Beattie, T.J.M. Bench-Capon, B.M. Diaz and M.J.R. Shave, Spatial reasoning for geographic information systems. *Proceedings 1st International Conference on GeoComputation*, School of Geography, University of Leeds, **1** 121-131 (1996).

[3] A.G.P. Brown, F.P. Coenen, M.J.R. Shave and M.W. Knight, An AI approach to noise prediction, To appear in: *Building Acoustics* **4**(2) (1998).

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

[5] F.P. Coenen, B. Beattie, T.J.M. Bench-Capon, B.M. Diaz and M.J.R. Shave, Spatio-temporal reasoning using a multi-dimensional tesseral representation, *Proceedings ECAI'98*, John Wiley & Sons, 140-144 (1998).

[6] F.P. Coenen, B. Beattie, T.J.M. Bench-Capon, M.J.R Shave and B.M. Diaz, 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, 57-68 (1995).

[7] F.P. Coenen, Rulebase Checking Using a Spatial Representation, to appear in *Database and Expert Systems Applications* (Proceedings DEXA'98), Lecture Notes in Computer Science, Springer Verlag, (1998).

[8] J.F. Allen, Time and time again: the many ways to represent time, *International Journal of Intelligent Systems* **6** 341-355 (1991).

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

[10] S.B.M. Bell, B.M. Diaz, F.C. Holroyd and M.J.J. Jackson, Spatially referenced methods of processing raster and vector data, *Image and Vision Computing* **1**(4) 211-220 (1983).

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

[12] G.M. Morton, A computer oriented geodetic data base, and a new technique in file sequencing, *IBM Canada Ltd.* (1966).

[13] B. Grunbaum and G.C. Shephard, *Tilings and Patterns*, Freeman, New York (1987).

[14] F.C. Holroyd, The Geometry of tiling hierarchies, *Ars Combinatoria* **16B** 211-244 (1983).

[15] H. Samet, The Quadtree and Related Data Structures, *ACM Comput. Survey* **16**(2)187-260 (1984).

[16] I. Gargantini, Linear octrees for fast processing of three dimensional objects. *Computer Graphics and Image Processing* **20** 365-374 (1982).

[17] G. Retz-Schmidt, Various views on spatial preposition. *AI Magazine* **9**(2) 95-105 (1988).

[18] 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).

[19] P. van Hentenryck, *Constraint Satisfaction in Logic Programming*, MIT Press, Cambridge, Massachusetts (1989).

[20] A.K. Mackworth, Consistency in networks of relations *AI Journal* **8**(1) 99-118 (1977).

[21] J-L. Lauriere, A language and a program for stating and solving combinatorial problems, *Artificial Intelligence* **10**(1) 29-127 (1978).

[22] M. Dincbas, P. van Hentenryck, H. Simonis, A. Aggoun, T. Graf and F. Berthier, The constraint logic programming language CHIP, *Proceedings of the International Conference on Fifth Generation Computer Systems (GGCS'88)*, Tokyo, Japan (1988).

[23] A. Colmerauer, Opening the PROLOG-III universe, *Byte Magazine* **12**(9) (1987).

[24] J. Jaffar and S. Michaylov, Methodology and implementation of a CLP system, *Fourth International Conference on Logic Programming*, Melbourne, Australia (1987).

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

[26] C. Freksa, Qualitative spatial reasoning, in: D.M. Mark and A.U. Frank eds., *Cognitive and linguistic aspects of geographic space*, Kluwer, Dordrecht, Netherlands, 361-372 (1991).

[27] 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).

[28] A. Mukerjee and G. Joe, A qualitative model of space, *Proceedings AAAI'90* 721-727 (1990).

[29] G. Ligozat, Weak representations of interval algebras, *Proceedings AAAI'90* 715-720 (1990).

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

[31] J.F. Allen, Towards a general theory of action and time, *Artificial Intelligence* **23** 123-154 (1984).

[32] . J.F. Allen and J.A. Koomen, Planning using a temporal world model, *Proceedings IJCAI'83* 2, Morgan Kaufman, Los Altos, 741-747 (1983).

[33] J.F. Allen and P.J. Hayes, A Common Sense Theory of Time, *Proceedings IJCAI'85*, Morgan Kaufman, Los Altos, 528-531 (1985).

[34] J.F. Allen and P.J. Hayes, Short time periods, *Proceedings IJCAI'87*, Morgan Kaufman, Los Altos, 981-983 (1987).

[35] J.F. Allen and P.J. Hayes, Moments and points in an interval-based temporal logic. *Computat. Intell. (Canada)* **5** 225-238 (1989).

[36] A. Galton, A critical examination of Allen's theory of action and time. *Artificial Intelligence*, **42** 159-188 (1990).

[37] M.B. Vilain and H. Kautz, Constraint propagation algorithms for temporal reasoning, *Proceedings AAAI'86* Morgan Kaufmann, Los Altos, CA, 377-382 (1986).

[38] M.B. Vilain, H. Kautz and P.G. van Beek, Constraint propagation algorithms for temporal reasoning: a revised report, in: A.D. Weld and J. de Kleer, eds., *Readings in Qualitative Reasoning about Physical Systems*, Morgan Kaufmann, San Mateo, CA., 373-381 (1989).

[39] K. Nokel, Convex relations between time intervals, *Proceedings 5th Osterreichische Artificial-Intelligence-Tagung*, Springer, Berlin, 298-302 (1989).

[40] D. Puller and M.J. Egenhofer, Towards formal definitions of topological relations among spatial objects, in: D. Ma rble ed., *Proceedings Third International Symposium on Spatial Data Handling* 225 -242 (1988).

[41] C.L. Hamblin, Instants and intervals, *Studium Generale* **24** 127-134 (1971).

[42] S.K. Chang, Q.Y. Shi and C.W. Yan, Iconic Indexing by 2-D strings, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAM1-9**(3) 413-428 (1987).

[43] A. Frank, R. Barrera and K.K. Al-taha, Temporal relations in geographic information systems, *ACM SIGMOS Record* **20**(3) 85-91 (1991).

[44] J. Malik and T.O. Binford, Reasoning in time and space. *Proceedings, International Joint Conference on Artificial Intelligence (IJCAI'83)* **1** 343-345 (1983).

[45] D. Papadias, Y. Theodoridis, T. Sellis and M.J. Egenhofer, Topological relations in the world of minimum bounding rectangles: a study with R-trees, *SIGMOD'95*, San Jose, CA, USA, 92-103 (1995).

[46] M.J. Egenhofer, A formal definition of binary topological relationships, in: W. Litwin and H-J. Scheck eds., *Proceedings third International Conference on Foundations of Data Organisation and Algorithms (FODO), Lecture Notes in Computer Science 367*, Springer-Verlag, New-York, 457-472 (1989).

[47] M.J. Egenhofer and R. Franzosa, Point-set topological spatial relationships, *International Journal of Geographic Information Systems* **5** 161-174 (1991).

[48] B.L. Clarke, A Calculus of individuals based on connection, *Notre Dame Journal of Formal Logic* **2**(3) (1981).

[49] B.L. Clarke, Individuals and points, *Notre Dame Journal of Formal Logic* **26**(1) (1985).

[50] A.G. Cohn, A more expressive formulation of many sorted logic, *Jo of Automation and Reasoning* **3**(2) 113-200 (1987).

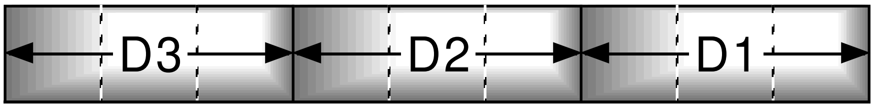
[51] A.D. Randell, Analysing the familiar: reasoning about space and time in the everyday world, PhD. Thesis, University of Warwick, UK (1991).

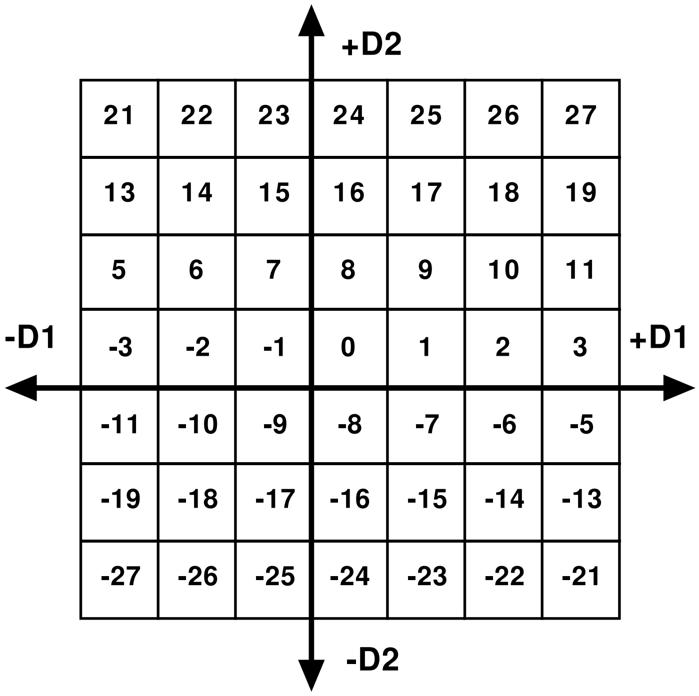
[52] A.D. Randell, Z. Cui, and A.G. Cohn, An interval logic for space based on "connections". *Proceedings of ECAI'92* (1992).

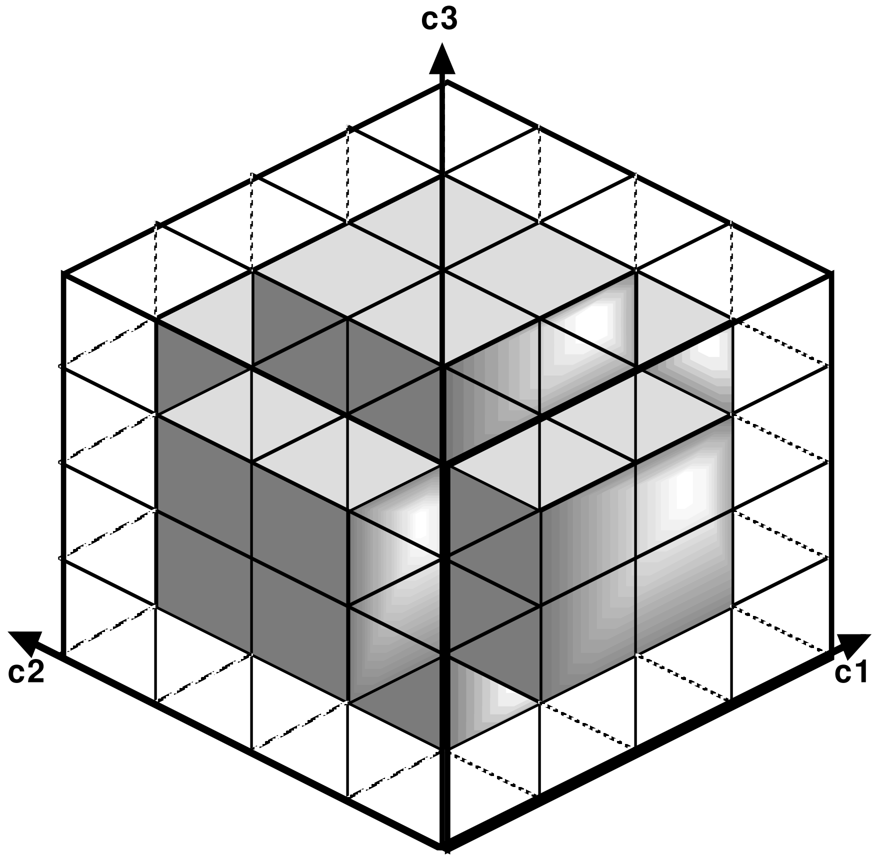
[53] A.D. Randell, Z. Cui and A.G. Cohn, A spatial logic based on regions and connection. *Proceedings 3rd International Conference on Principals of Knowledge Representation and Reasoning*, Morgan Kaufmann (1992).

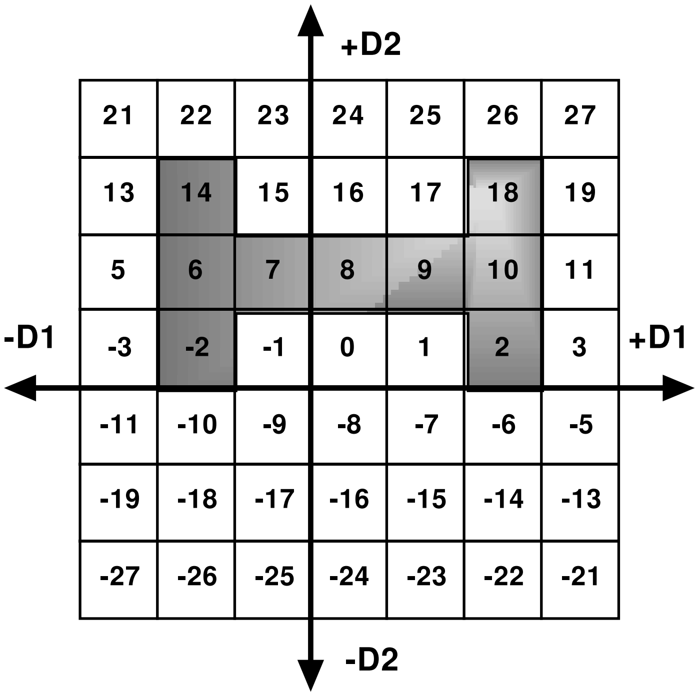
[54] Z. Cui, A.G. Cohn and D.A. Randell. Qualitative simulation based on a logic of space and time, *Proceedings AISB Workshop "Qualitative and causal Reasoning" (QCR'93)*, University of Birmingham (1993).

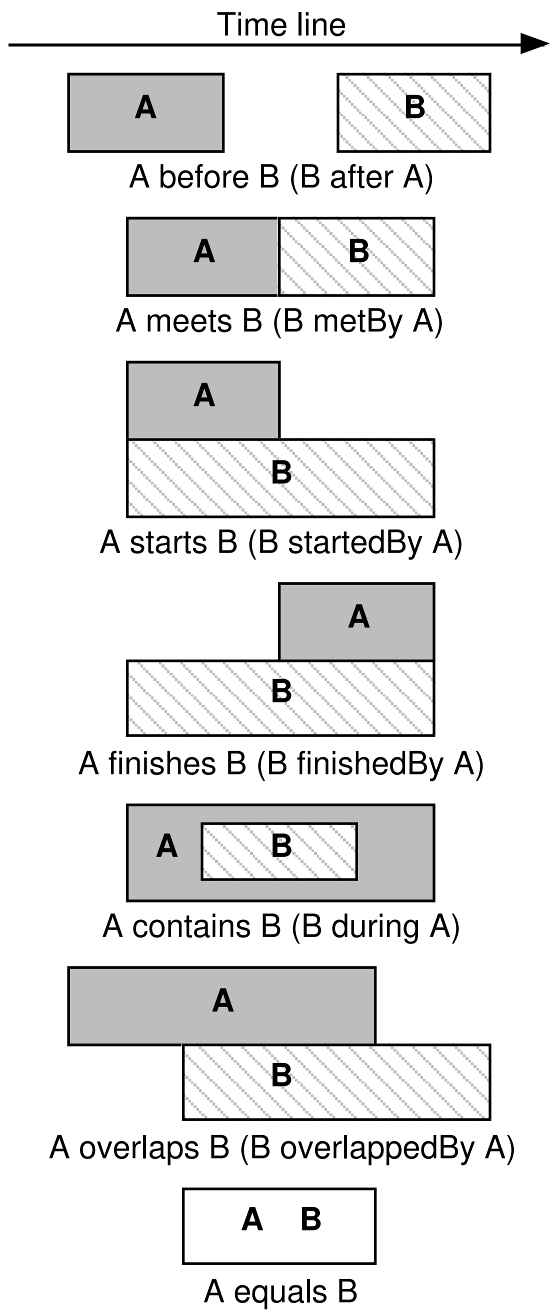
[55] L. Vieu, A logical framework for reasoning about space, in: A.U. Frank and I. Campari, eds., *Spatial Information theory: A Theoretical Basis for GIS, proceedings COSIT'93*, Springer-Verlag, Berlin, 25-35 (1993).


D3
D2
D1


+D2
21 22 23 24 25 26 27
13 14 15 16 17 18 19
5 6 7 8 9 10 11
-D1
-3 -2 -1 0 1 2 3
+D1
-11 -10 -9 -8 -7 -6 -5
-19 -18 -17 -16 -15 -14 -13
-27 -26 -25 -24 -23 -22 -21
-D2


c3
c2
c1


+D2
21 22 23 24 25 26 27
13 14 15 16 17 18 19
5 6 7 8 9 10 11
-D1
-3 -2 -1 0 1 2 3
+D1
-11 -10 -9 -8 -7 -6 -5
-19 -18 -17 -16 -15 -14 -13
-27 -26 -25 -24 -23 -22 -21
-D2


Time line
A
B
A before B (B after A)
A
B
A meets B (B metBy A)
A
B
A starts B (B startedBy A)
A
B
A finishes B (B finishedBy A)
A
B
A contains B (B during A)
A
B
A overlaps B (B overlappedBy A)
A B
A equals B

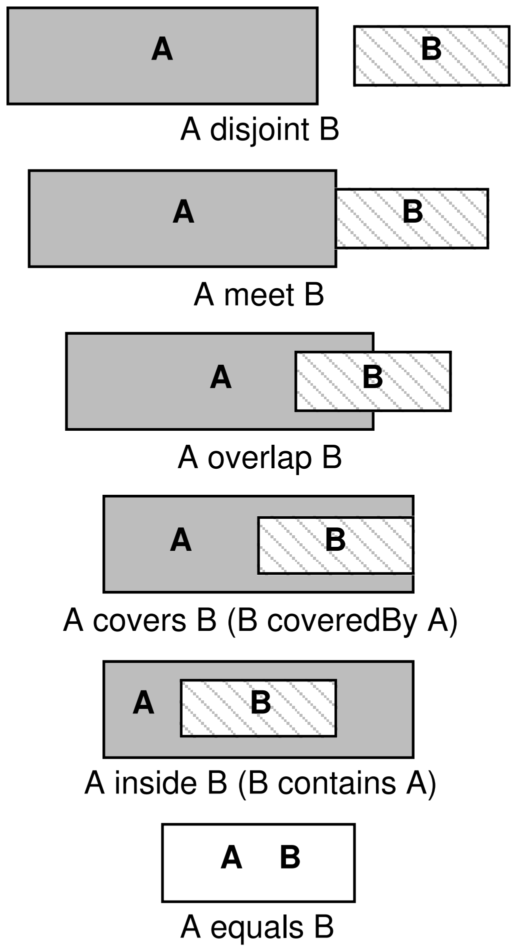

A
B
A disjoint B
A
B
A meet B
A
B
A overlap B
A
B
A covers B (B coveredBy A)
A
B
A inside B (B contains A)
A B
A equals B

Figure 1: Bit allocation

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

Figure 3: 3-D example shape

Figure 4: 2-D example shape

Figure 5: Allen's 13 interval relations

Figure 6: The 8 "contiguous object" topological relations

BIOGRAPHICAL SKETCH

Frans Coenen is a lecturer within the Department of Computer Science at the University of Liverpool. His current research interests include: spatio-temporal reasoning - particularly its implementation using tesseral approaches and its application to Geographic Information Systems (GIS) and marine electronic chart systems: Knowledge Based Systems (KBS) and their validation, verification and maintenance; and the application of all forms of computer science to the maritime industry. He has published widely on all these subjects.