SOME THOUGHTS ON THE APPLICATION OF SPATIAL REPRESENTATION TECHNIQUES TO SUPPORT CASE BASED REASONING

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

1 INTRODUCTION

Below are presented some ideas on how quantitative spatial representations may usefully be employed to support Case Based Reasoning (CBR). The representation used is founded on the SPARTA (SPAtial Reasoning using Tesseral Addressing) system ([2]) which in turn was developed as part of the dGKBIS (dynamic Geographic Knowledge Based Information Systems) projects ([1, 3]).

The document is organised as follows. Section 2 gives an overview of the SPARTA tesseral representation together with brief notes on tesseral representations in general, data storage and tesseral arithmetic. An outline of the SPARTA scripting language used to define tesserally referenced spatial problems, use of which will be made later in the text, is then presented in Section 3. In Section 4 a hypothetical example CBR application, taken from a legal domain, is presented which is then encoded using the SPARTA representation and interacted with using the approach to spatial reasoning developed as part of the dGKBIS project. In Section 5 some alternative case base encapsulation are described and finally, in Section 6, some discussion is presented.

2 THE SPARTA TESSERAL REPRESENTATION

The SPARTA representation is a mechanism for referencing N-Dimensional space in a manner that offers advantages with respect to data storage and computational efficiency regardless of the number of dimensions under consideration. To reference a space S we commence with the finite set of dimensions associated with this space $\{D_1, D_2, \ldots, D_n\}$. Each D then has a range of values associated with it defined as follows $< V_{min}, \ldots, 0, \ldots, V_{max} >$ (where the magnitude of V_{min} is equivalent to V_{max}). We then allocate bits in an integer type to each dimension. For example let us assume a positive 4-D space measuring $10 \times 4 \times 3 \times 5$. D_1 to D_4 would then be defined as follows:

$$\begin{aligned} D_1 &:= \; =< -9 \ldots 9 > \\ D_2 &:= \; =< -3 \ldots 3 > \\ D_3 &:= \; =< -2 \ldots 2 > \\ D_3 &:= \; =< -4 \ldots 4 > \end{aligned}$$

Note that the negative equivalents of the positive space have been included in the above because these values are required when we come to manipulate objects contained in the space. Assuming

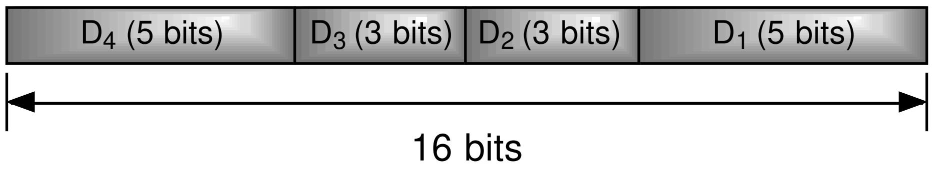

Figure 1: Bit pattern for example space

(for illustrative purposes) a 16 bit integer type the bit pattern shown in Figure 1 would be appropriate (in the Figure the least significant bit is to the right). Using this bit pattern we can actually represent a larger positive space measuring $32 \times 4 \times 4 \times 32$, however, we are only interest in a subset of this larger space. We refer to the space of interest as the *object space* (the space in which the objects we are interested in are contained). Note also that the objects space always comprises only positive space and includes the zero address (the *global origin* for the space). To calculate references within a 4-D space we use the following identity:

$$address := (V_1 \times Base_1) + (V_2 \times Base_2) + \ldots + (V_n \times Base_n)$$

With respect to the bit pattern presented in Figure 1 the bases will be as follows: $Base_1 = 1$, $Base_2 = 4$, $Base_3 = 256$, and $Base_4 = 2048$. Thus the address for the *cell* defined by the coordinate 4-tuple $< 7, 2, 0, 0 >$ will be:

$$address := (7 \times 1) \ + \ (2 \times 32) \ + \ (0 \times 256) \ + \ (0 \times 2048) := 71$$

Although it is difficult to display a 4-D space using a 2-D medium we can conveniently display 2-D "slices" of the space. Figure 2 shows the slice where D_3 and D_4 both equal 0. Using this figure we can verify the address 71 calculated above.

Conversion from an address back to a set of coordinates can also be achieved in a simple manner using appropriate left and right "bit shifts". The advantages of the representation (with respect to Cartesian and other "tesseral" representations) are as follows:

1. It results in a "left-to-right" linearisation of space which in turn offers advantages of:
   - General applicability to any number of dimensions without requiring alteration of the code or recourse to procedures and functions dedicated to a particular number of dimensions (as in the case of systems founded on Cartesian approaches).
   - Efficient data storage using ideas founded on "run-line" encoding strategies (see note in Sub-section 3.2 for further details).
   - Computationally effective comparison and translation of sets of addresses.
   - Predictability, i.e. given a location we can always predict the addresses of physically adjacent cells (this is not the case with other tesseral approaches).
2. Supports computationally efficient translation through the space using straightforward integer addition and subtraction, unlike other tesseral systems which required recourse to "look-up" tables or specialised tesseral processors (see note in Sub-section 3.3 for further details).
3. Provides for the rotation of objects.
4. If desired, addresses can be converted to and from Cartesian tuples in an extremely effective manner as described above (not the case with standard tesseral approaches).

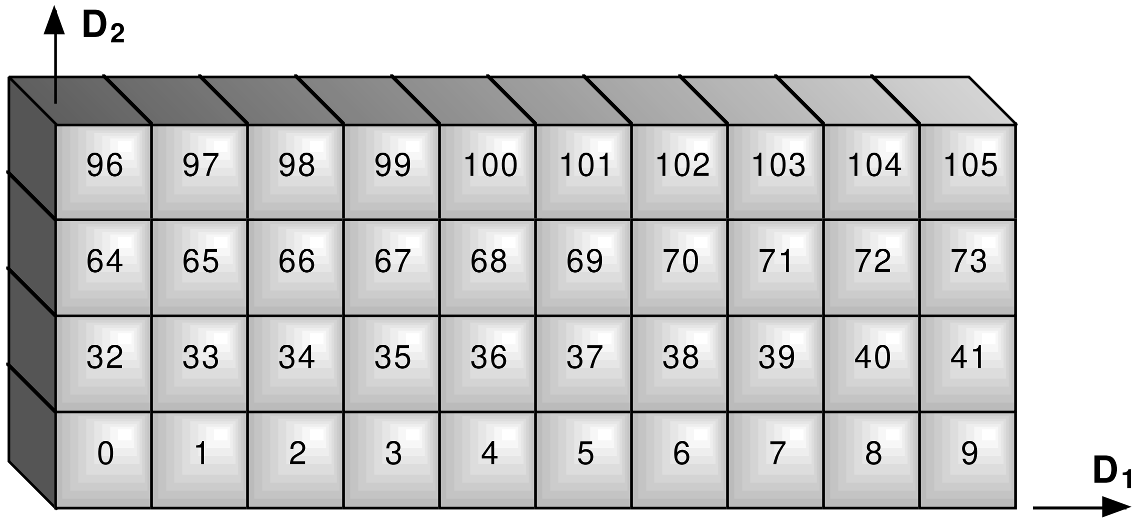

Figure 2: 2-D slice through example space ($D_3 = 0$ and $D_4 = 0$)

.

5. The addressing system is intuitive.

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

2.1 Note on tesseral representations

Tesseral representations are spatial representations which are derived by repeatedly dividing a space into isohedral sub-spaces until some predefined resolution is achieved. The resulting sub-spaces are then allocated (usually numeric) "addresses" in such a way as to reflect the hierarchical decomposition. The SPARTA representation described above is categorised as a tesseral representation because it is a mechanism for addressing spaces decomposed into isohedral subspaces at some predefined resolution, however, there is no explicit inclusion of the concept of hierarchical decomposition and consequently this is not reflected in the addressing system.

2.2 Note on data storage

Using the representation the locations of objects within the object space are presented in terms of sequences of 4-D *cells*. Each cell is defined in terms of a prefix address and a postfix address separated by an operator "..". The prefix address is the address geometrically nearest to the origin, and the post fix address is the address furthest away. Thus the entire objects space in Figure 2 would be defined by the set:

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

while a two dimensional "H" shape by the set:

$$\{2..66, 6..70, 35..37\}$$

(the latter can be verified by reference to Figure 2). Note the ordering of cells according to the location of the prefix address within the linearisation. A single point is defined as a cell where the prefix and postfix addresses are identical, e,g. $\{64..64\}$, for ease of reading in the following text we present such points using only a single address.

2.3 Note on tesseral arithmetic

One of the most useful advantages of the representation is that we can move around (translate through) any space, referenced using the SPARTA approach, using simple integer arithmetic. For example, to move one point to the right (given the orientation in Figure 2) from a given reference we simply add 1 ($Base_1$), to move in the opposite direction we subtract 1. If we wish to move the "H" defined in the above section two points to the right and one point upwards we simply add 34 to each address in the sequence:

$$\{2..66, 6..70, 35..37\} + \{34\} = \{36..100, 40..104, 69..71\}$$

If, alternatively, we wish to expand the "H" in both directions along the D_1 axis by one point we add the cell $-1..1$. Moreover to complete this addition we only need to add the prefix operands in the first sequence to the prefix operands in the second sequence, and similarly with the postfix operands. Thus the expanding operation only requires six simply addition operations:

$$\{2..66, 6..70, 35..37\} + \{-1..1\} = \{1..68, 5..71, 68..71\}$$

The overlap that this entails is not significant. Similarly if we wish to expand the "H" in all directions we would add the cell $\{-2337..2337\}$.

3 SPARTA SCRIPTING LANGUAGE

A scripting language has been developed ([4]) to define tesserally referenced spatial problems. This language is founded on a subset of the PROLOG AI programming language. A script comprises the following parts:

$$< script >:< spaceDec >, < objectDec >, < constraintsDec >;$$

The *spaceDec* part is used to define the object space for a given application. Thus the above object space would be achieved as follows:

$$script(10, 4, 3, 5).$$

The *objectDec* part is then used to define the spatial objects of interest. In the *constraintsDec* part the relationships which exist or are desired to exist between spatial objects are expressed. If desired objects can also be grouped together in *classes*, this then conveniently allows several objects to be referenced in a single constraint (example will be given later in the text).

The significance of the scripting language to this text is that it is the medium used to describe CBR scenarios in Sections 4, 5 and 6. For more comprehensive details concerning the scripting language interested readers are referred to [4].

4 AN EXAMPLE CBR APPLICATION

Consider the following set of a cases taken from a hypothetical "house burglary" database:

1. **CASE 1:** Burglary occurred during the day whilst occupants were in close proximity (in garden) penalty 2x.
2. **CASE 2:** Burglary occurred at dawn while occupants were out (on holiday), penalty 2x.
3. **CASE 3:** Burglary occurred at dusk while occupants were in (asleep), penalty 6x.
4. **CASE 4:** Burglary occurred during the day whilst occupants were in (upstairs), penalty 3x.
5. **CASE 5:** Burglary occurred at night whilst occupants were in close proximity (in garage), penalty 6x.

Each cases has four common attributes:

$$
\begin{aligned}
penalty &:= range\{0x..9x\} \\
timeOfDay &:= \{day, dusk, night, dawn\} \\
occupancy &:= \{out, closeBy, in\} \\
caseNumber &:= range\{1..5\}
\end{aligned}
$$

If we assume that each attribute represents a dimension, and that the values given for each attribute describe the range of values to be associated with each dimension, we can define a 4-D space as follows:

$$
\begin{aligned}
penalty - D_1 &:= < -9, \ldots, 9 > \\
timeOfDay - D_2 &:= < -3, \ldots, 3 > \\
occupancy - D_2 &:= < -2, \ldots, 2 > \\
caseNumber - D_2 &:= < -4, \ldots, 4 >
\end{aligned}
$$

Note that we have described the enumerated value sets as ordered lists, in some cases there may be some significance to this ordering (e.g. time of day), in others this significance may be less clear (e.g. occupancy). We might also wish to reorder the enumerations, for example $\{dawn, day, dusk, night\}$ for the $timeOfDay$ attribute instead of $\{day, dusk, night, dawn\}$. We have also extended the penalty range so that it starts at 0x and ends a 9x. As a result of the above we have defined a $10 \times 4 \times 3 \times 5$ objects space, such as that developed in Section 1, in which the cases can be said to exist. The individual case references are as follows:

$$
\begin{aligned}
case1 &:= 2 + 0 \times 32 + 1 \times 256 + 0 \times 2048 := 2 + 32 + 256 + 0 := 258 \\
case2 &:= 2 + 3 \times 32 + 0 \times 256 + 1 \times 2048 := 2 + 0 + 0 + 2048 := 2146 \\
case3 &:= 6 + 1 \times 32 + 2 \times 256 + 2 \times 2048 := 6 + 64 + 512 + 4096 := 4646 \\
case4 &:= 3 + 0 \times 32 + 2 \times 256 + 3 \times 2048 := 3 + 96 + 512 + 6144 := 6659 \\
case5 &:= 6 + 2 \times 32 + 1 \times 256 + 4 \times 2048 := 6 + 96 + 256 + 8192 := 8518
\end{aligned}
$$

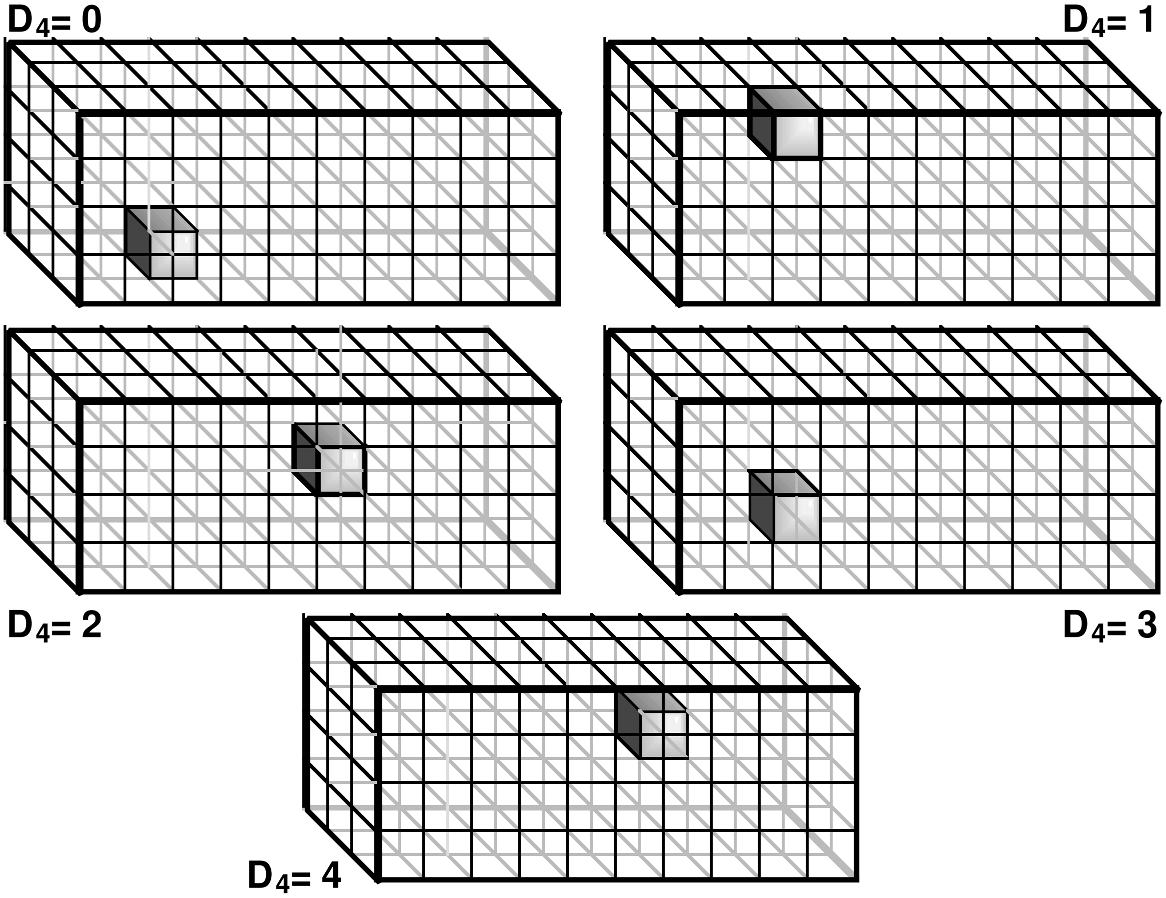

Figure 3: A spatial case base

.

Each of the resulting references uniquely encapsulates a case and all its attributes in to a single number. Using a sequencing technique (often used to represent 4-D domains where the fourth dimension is time) we can display the result as shown in Figure 3.

In terms of the scripting language (introduced in Section 3) the case base can now be defined as follows:

```
space(10,4,3,5).

object(knownCases,fixed,{258,2146,4646,6659,8518}).
```

Thus we have defined a $10 \times 4 \times 3 \times 5$ objects space, and an object *knownCases* which has the attribute "fixed" and is located at $\{258, 2146, 4646, 6659, 8518\}$ within the object space (spatial objects do not have to be contiguous). The attribute *fixed* indicates that the object in question can not be moved - for many spatial problem scenarios we often also wish to consider objects which have a particular shape but no "fixed" location or objects that have no known shape or location (we refer to such objects as *free* and *shapeless* objects respectively).

We can now use this space to address CBR style queries expressed in terms of spatial constraints. Suppose we have a new case where the burglary occurred at night whilst occupants were in (asleep upstairs). One of the first things we might wish to do is establish whether there are any

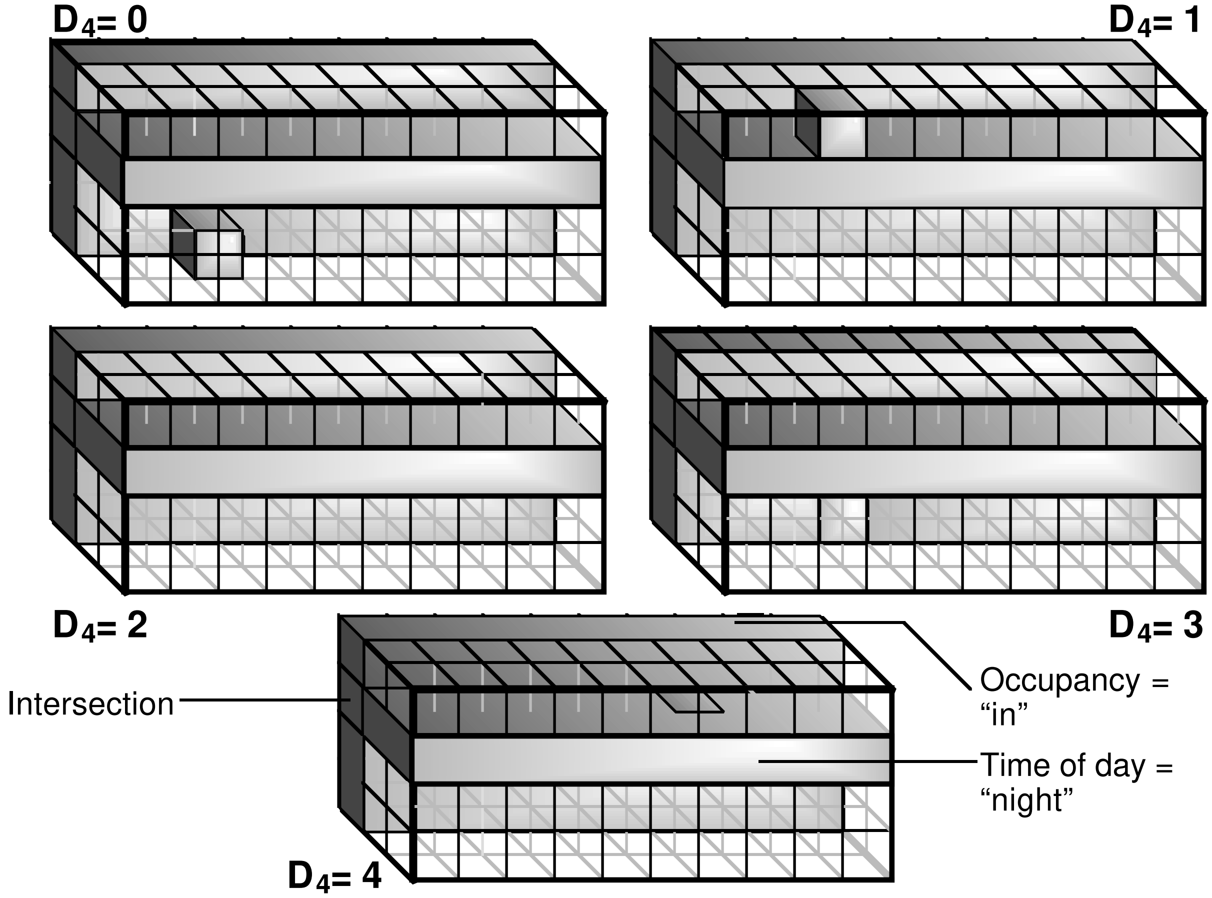

Figure 4: Subshapes

.

identical cases in the case base. The subspace in the object space where the attribute $timeOfDay$ has the value "night" (i.e. 2) is defined by the cell $\{64..8777\}$ ($64 := 0 + 2\times32 + 0\times256 + 0\times2048$ and $8768 := 9 + 2 \times 32 + 2 \times 256 + 4 \times 2048$). The subspace in the object space where the attribute *occupancy* has the value "in" (i.e. 2) is defined by the cell 512..8809 ($512 := 0 + 0 \times 32 + 2 \times 256 + 0 \times 2048$ and $8809 := 9 + 3 \times 32 + 2 \times 256 + 4 \times 2048$). These sub shapes are illustrated in Figure 4. We can now define our case retrieval query as follows by extending the above script:

```
space(10,4,3,5).

object(knownCases,fixed,{258,2146,4646,6659,8518}).
object(timeOfDayIsNight,fixed,{64..8777}).
object(occupancyIsIn,fixed,{512..8809}).
object(identicalCases,shapeless,{0..8809}).
```

```
constraint(identicalCases,intersect,timeOfDayIsNight).
constraint(identicalCases,intersect,occupancyIsIn).
constraint(identicalCases,intersect,knownCases).
```

Note that in the above script that *identicalCases* object has an attribute *shapeless*. This indicates that we have no knowledge about the shape or location of the object referred to other than that it exists somewhere within the object space ($\{0..8809\}$). The constraints express the requirement that for a case to be "identical" it must have the attribute values $timeOfDayIsNight$ and $occupancyIsIn$ and must be a member of the set of cases. From Figure 4 we can see that there are no cases located within the intersection where the time of day attribute has a value of *night* and the occupancy attribute has a value of *in*, the identical cases set returned will thus be :

$$\begin{aligned}
identicalCases &:= \{0..8809\} \cap (\{64..8777\} \cap \{512..8809\} \cap \{258, 2146, 4646, 6659, 8518\}) \\
identicalCases &:= \{64..8777\} \cap (\{64..8777\} \cap \{512..8809\} \cap \{258, 2146, 4646, 6659, 8518\}) \\
&:= \{576..8777\} \cap \{512..8809\} \cap \{258, 2146, 4646, 6659, 8518\} \\
&:= \{\}
\end{aligned}$$

The above result could of course have been established using a straight forward SQL style query, however the number of comparisons required to establish the result using the above spatial approach is significantly less than that required to compare individual rows in (say) a table of cases — we only need to compare the single integers representing each case as opposed to groups of attribute values (integer or otherwise) pertaining to particular cases. Having established that there is no direct match we will wish to expand our search, we can do this in either a uniform manner, or we can expand along particular axes (as in the "H" expansion example presented in Sub-section 3.3) and/or in particular directions. Let us assume that wish to extend our search by expanding one point each way along the time of day axis. To do this we revise the first of the constraints as follows:

```
...
object(similarCases,shapeless,{0..8809}).

constraint(similarCases,intersect,timeOfDayIsNight,offset{-32..32}).
constraint(similarCases,intersect,occupancyIsIn).
constraint(similarCases,intersect,knownCases).
```

The "offset" attribute included in the above indicates that the set $\{-32..32\}$ should be added to the $timeOfDayIsNight$ set before the intersect operation is performed. Thus:

$$\begin{aligned}
similarCases &:= \{0..8809\} \cap ((\{64..8777\} + \{-32..32\}) \cap \{512..8809\} \cap \\
&\qquad\qquad \{258, 2146, 4646, 6659, 8518\}) \\
&:= \{0..8809\} \cap (\{32..8809\}) \cap \{512..8809\} \cap \{258, 2146, 4646, 6659, 8518\}) \\
&:= \{32..8809\} \cap (\{512..8809\} \cap \{258, 2146, 4646, 6659, 8518\}) \\
&:= \{544..8809\} \cap \{258, 2146, 4646, 6659, 8518\} \\
&:= \{4678\}
\end{aligned}$$

Here we have identified a single case (case 3) which differs from the new case only in that it occurred at dusk. The reference 4678 equates to the coordinate 4-tuple $< 6, 3, 2, 2 >$ and thus we

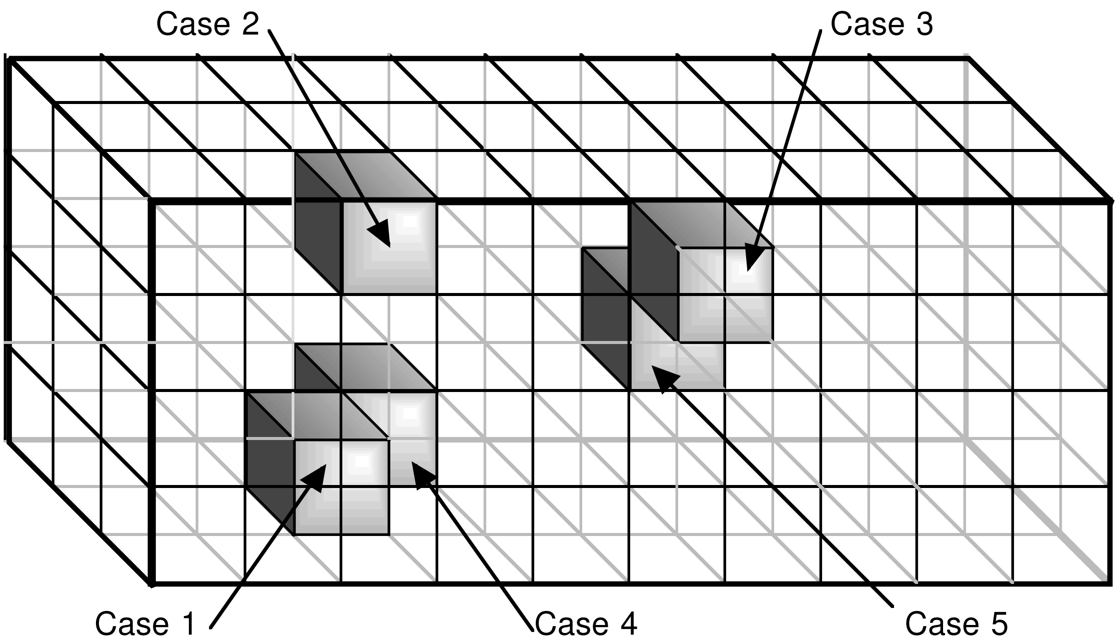

Figure 5: 3-D encapsulation of the case base

.

can deduce that the likely penalty is $6x$. Alternatively, we could also have expanded the occupancy attribute, in which case further cases might be retrieved and a more accurate judgement as to the likely penalty, based on a number of cases, could consequently be made.

5 ALTERNATIVE ENCAPSULATIONS

An alternative approach to encapsulating the case base is to consider each case to be an object. Thus we reduce the objects space from a 4-D space to a 3-D space. Although this offers few computational advantages a 3-D space is simpler to envisaged. Given the example case base this approach would give an objects space measuring $10 \times 4 \times 3$:

```
space(9,3,2).
```

The cases of interest would then be described as follows:

```
object(case1,fixed,{290}).
object(case2,fixed,{98}).
object(case3,fixed,{550}).
object(case4,fixed,{547}).
object(case5,fixed,{326}).
```

The resulting object space is shown in Figure 5. This representation of cases is probably more amenable to CBR practitioners as it has a closer affinity to accepted case base formalisms as opposed to the encapsulation presented in Section 4 above. The attributes of the new case are then described as follows:

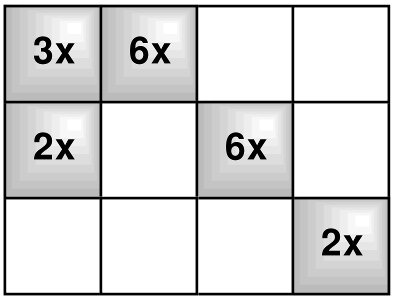

Figure 6: 2-D encapsulation of the case base

.

```
object(newCase,fixed,{576..585}).
```

Note that we have combined the attributes $timeOfDayIsNight$ and $occupancyIsIn$ into a single attribute *newCase*. The constraints would remain unchanged:

```
constraint(identicalCases,intersect,newCase).
constraint(identicalCases,intersect,knownCases).
```

(Note that the identifier *knownCases* used here is a "class" label in that it indicate the set of cases - $\{case1, case2, case3, case4, case5\}$). The calculation would then be as follows:

$$\begin{aligned} identicalCases &:= \{576..585\} \cap (\{290\} \cap \{98\} \cap \{550\} \cap \{547\} \cap \{326\}) \\ &:= \{\} \end{aligned}$$

Note that using this encapsulations the "work" to be done is slightly reduced. As before, in the absence of an identical case, we can now expand the object *newCase* in a particular direction or globally, in an attempt to find "similar" case.

A further alternative encapsulation is to consider the differing "penalties" as objects in a 4×3 2-D space (and ignore individual cases). In this case $base_1 = 1$ and $Base_2 = 8$. Thus:

```
penalty2x := {(3 + 0, 0 + 8)}  := {3, 8}
penalty3x := {(0  + 16)}  := {16}
penalty6x := {(2 + 8, 1 + 16)}  := {10,17}
```

The result is shown in Figure 6. This 2-D encapsulation is probably the most simplistic although we have lost detail of particular cases. This information would be required, given our legal example application domain, should we wish to argue a case, in a court of law, on "precedent". Using this 2-D encapsulation the object *newCase* would be defined as follows:

```
object(newCase,fixed,{19}).
```

we can now define a script:

```
space(4,3).

class(penalties,{penalty2x,penalty3x,penalty6x)).
```

```
object(penalty2x,fixed,{3, 8}).
object(penalty3x,fixed,{16}).
object(penalty6x,fixed,{10,17}).
object(newCase,fixed,{19}).
object(expectedPenalty,shapeless,{0..19}).

constraint(expectedPenalty,intersects,penalties).
constraint(expectedPenalty,intersects,newCase).
```

Thus:

$$\begin{aligned} identicalCases &:= \{0..19\} \cap (\{3, 8, 10, 16, 17\} \cap \{19\} \\ &:= \{\} \end{aligned}$$

In the absence of a "direct hit", to find the nearest penalty we can now expand the *newCase* object in one ore more directions as dictated by the nature of the application until an appropriate "hit" is made.

6 DISCUSSION

In the foregoing we have essentially used the dGKBIS approach to reference a case base and interact with this case base in a fairly standard manner using the *intersection* set operation. The straight forward case retrieval is achieved in a similar manner as if an SQL style query had been used — although the result is arguably obtained in a more computational effective manner (especially where a large case base with many attribute/columns is under consideration).

The expansion technique (offsets) which may be invoked when ever a direct hit is not made is more interesting in that it is computationally cheap to expand a set describing the nature of the object, particularly given a large number of attributes. In addition the expansion does not necessarily require detailed knowledge of the application — in the absence of any information dictating otherwise an attribute set can be expanded uniformly in all directions.

There are, of course, a number of other set operations that can be brought into play, namely: *subset*, *superset*, *equals*, *complement* and *union* . Note that the first are Boolean operators which return "true" or "false", i.e. *A subset B* will return true if *A* is "within" *B* and false otherwise. The complement and union operators return non-Boolean values, i.e. *A complement B* will return the set of elements which is in *A* but not in *B* (this set might of course be the empty set). The *intersection* operator can be used as both a Boolean and non-Boolean function as desired. Although we have extensively illustrated the use of the *intersection* non-Boolean operator in the context of case base interaction, we have not illustrated the use of the others. Indeed, for many of these operators, the meaning of the operation with respect to CBR is not entirely clear. We will briefly discus each of the these other operators, in the context of CBR, below:

Subset: With respect to the CBR application domain under consideration here, given *A subset B* this would return true if the attribute values for A were a subset of those for B (where A and B represent individual cases). This would occur where all attribute values that can be associated with B are not precisely know, i.e. one or more of the attributes in question have a number of candidate alternate values associated with them. For example if we have a new case where the burglary occurred at night but we have no information as to occupancy (using the 3-D representation presented in Figure 5):

```
object{newCase,fixed,{96..617})
```

then we might define a script:

```
space(9,3,2).

object(case1,fixed,{290}).
object(case2,fixed,{98}).
object(case3,fixed,{550}).
object(case4,fixed,{547}).
object(case5,fixed,{326}).
object{newCase,fixed,{96..617})

constraint(case5,subset,newCase).
```

This would return true ($\{326\} \subset \{96..617\}$ — remember that $\{96..617\}$ describes a 3-D box) however, given a large number of cases, we would have to test each case for a match therefore this would be unlikely to lead to any real advantage unless there was some way of ascertaining whether case 5 was likely to be a similar case.

Superset: The *superset* operator represents the reverse situation to that associated with the *subset* operator. The above discussion is therefore applicable.

Equals: The *equals* operator can be used in a similar fashion to that described for the subset and superset) operators described above to determine whether, given a new case, a particular case is identical to it. Again this would require us to test every case.

Boolean intersection: Given a number of attributes associated with a new case the Boolean *intersection* operator can be used to determine whether a particular case intersects with the new attribute definition. For example given the two cases (say case1 and case2 above) we might wish to determine whether these two cases had anything in common. This would be archived using the constraint:

```
constraint(case1,offset({-9..9,-96..96,-512..512},booleanIntersects,case2).
```

In effect what we are asking is *"Do the two cases have any common coordinates?"*. The offset has the effect of superimposing a 3-D "cross" over the case which we then test against the address for case 2 to determine whether an intersection exists (or not). In the above case the operator would return "true" (the two cases share the same penalty coordinate).

Complement: The complement operation is the reverse of the intersection non-Boolean operation the use of which was extensively illustrated in Sections 3 and 4. This can be used in an identical manner to (say) determine "similar cases" but with respect to those attributes we are not interested in. For example the constraints presented in the first scenario may, using the complement operator, be expressed as follows:

```
space(10,4,3,5).

object(knownCases,fixed,{258,2146,4646,6659,8518}).
object(timeOfDayIsNotNight,fixed,{0..8736, 96..8809}).
object(occupancyIsNotIn,fixed,{0..8553}).
object(identicalCases,shapeless,{0..8809}).
```

```
constraint(identicalCases,complement,timeOfDayIsNight).
constraint(identicalCases,complement,occupancyIsIn
constraint(identicalCases,intersect,knownCases).
```

In this example the amount of work to be done to resolve the constraints is greater than that required when the scenario is expressed in terms of the intersection operator (see Section 3). This disadvantage probably extends to most scenarios where the number of attributes of interest, compared to the total, are in a minority.

Union: All of the above operators are supported by the SPARTA system, and all may be used in conjunction with the offsets concept (in addition the intersection operator, in both forms, may also be quantified). The union operator on the other hand is not supported by the system. The operator could be used to enlarge candidate sets of attribute values for objects, however this would necessitate an ordering of constraints which intern would obviate the declarative nature of the scripting language. This is seen as undesirable and thus the union operator remains unsupported.

7 CONCLUSIONS

In this paper we have considered the application of the SPARTA/dGKBIS representation to the CBR domain. The paper clearly establishes that the casebase can be represented in a spatial context. The principal advantages of the encapsulation are:

- A case and all its attributes (regardless of how many) can be represented as a single integer. This in turn clearly offers computational advantages especially where a large number of cases are under consideration.
- The computational advantages are of particular relevance when we wish to expand a given set of attributes so as to identify similar cases
- The approach is also perfectly capable of operating with missing information or limited information.

Although these advantages are significant, the question yet to be answered is *"whether the advantages are of sufficient significance to suggest the approach as an appropriate tool to support CBR?"*.

References

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

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

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

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