SPATIAL REASONING FOR TIMETABLING: THE TIMETABLER SYSTEM

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

Department of Computer Science,
The University of Liverpool,
Chadwick Building, P.O. Box 147,
Liverpool L69 3BX, England.
email: frans@uk.ac.liverpool.compsci
Tel: 051 794 3698
Fax: 051 794 3715

The TIMETABLER system is described. This is a constraint-based time-tabling system founded on a spatial reasoning mechanism. The philosophy behind the system is that many time-tabling (and scheduling) tasks are traditionally implemented using a two-dimensional paper table or chart. Hence, it is conjectured, a spatial approach to addressing the automation of the time-tabling class of problem is the most natural option since it is precisely analogous to the approach used by expert time-tablers in the "field". The spatial reasoning mechanism in question uses a Quad Tesseral representation. This readily supports the description of spatial entities (resources) while at the same time supporting a clear and concise reasoning mechanism.

KEYWORDS: Constraint-based programming, Spatial reasoning, Quad tesseral addressing, TIMETABLER

1. INTRODUCTION

A time-tabling system, TIMETABLER, is described based on a spatial reasoning mechanism. The philosophy behind this system is that many time-tabling activities are traditionally implemented using a two-dimensional table or chart. These tables are handed out to employees and can be seen in offices throughout the world. They are usually used to match up three (or more) resources. There are many examples such as bus routes, buses and drivers; teachers, classes and class rooms; road length, time and maintenance gangs etc. Two of the resources of interest are represented by the X and Y axes, additional resources are then "juggled" between the intersections of pairs of X and Y coordinates. Note that the resources can be temporal (i.e. time), spatial (such as a road) or can simply represent some arbitrary set of entities (for example buses or machine operators). The time-tablers task is to match up these resources according to a given set of constraints using the table/chart as a support tool. This is usually implemented by allocating coloured pins or magnetic tags to individual intersections. The system described here automates this process in a manner analogous to that employed by a skilled time-tabler.

For the TIMETABLER system the time-tabling table or chart is referred to as the *entity space*, the space in which all the entities we wish to consider can be expected to exist. This space is then represented as a set of Quad Tesseral addresses. Quad Tesseral addressing (Diaz and Bell 1986, Gargantini 1983) is essentially a method of representing *tiles* (points of unit dimension) in multi-dimensional space with the result that the space in question is linearised. This offers the advantage that all (multi-dimensional) reasoning can take place along a single line so avoiding many of the combinatorial problems that result when attempting to reason about multi-dimensional space using a more traditional Cartesian approach. In addition the tesseral addressing technique has an arithmetic associated with it which allows easy manipulation of individual tesseral addresses.

Time-tabling problems are presented to the TIMETABLER system in the form of a script which comprises two parts, a declaration part and a constraint part. In the declaration section the individual entities and their attributes are defined, and in the constraints section the required relationships between entities. To this end the system supports two types of relationship, *within* and *notWithin*.

The TIMETABLER system operates using a constraint propagation technique whereby the possible locations for entities range over a set of candidate locations and the constraints are used to restrict this set of candidate locations. As a result one or more "time-tables" will be produced.

2. PREVIOUS WORK

Time-tabling applications can be considered to be a sub-set of a much larger category of application usually referred to as scheduling (Beck 1991, identifies six sub-classes of scheduling application). Time-tabling, in its purest form, is (arguably) the most straightforward to implement. The classic time-tabling application is the students/courses type time-tabling problem (Ferland and Roy 1985, Laporte and Desroches 1986). School time-tabling is also a frequently used example application (see Appleby et al. 1960, Berghuis et al. 1964, Barraclough 1965, de Gans 1981 and Aubin and Ferland 1989). Another frequently cited example is vehicle scheduling (see Lamma et al. 1994 or Duncan 1994).

It is suggested that the time-tabling system described here is applicable to a much wider set of applications than the straightforward time-tabling applications described above. The system is applicable to any scheduling application which can be presented in the form of a two-dimensional display. This includes applications such as project planning (Palmer 1994) and factory scheduling (Collyer and Hamazaki 1991). However, in this paper, we have chosen to illustrate the system with a school time-tabling application. This application offers the principal advantage that it is well understood while at the same time displaying all the characteristics of time-tabling/scheduling that we wish to highlight.

More generally, scheduling problems can be classed as Constraint Satisfaction Problems (CSP) and can be addressed using CSP techniques. CSP problems can generally be defined in terms of a set of variables `I = {X1, X2, .., Xn}` that can take values from the finite domains `{D1, D2, .., Dn}` respectively, and a set of constraints which specify which values are compatible with each other. A solution to a CSP is then an assignment of values to all variables which satisfies all constraints. Given that the domains under consideration are finite, a solution or solutions to a CSP is always possible (provided that no two constraints are mutually incompatible). The real problem associated with CSP is that of complexity (CSP are NP-complete) and, as a result, efficiency.

The most straightforward method of solving a CSP is to use a "generate and test" paradigm whereby every solution to a CSP is generated and tested against the given constraints. However, this is a very inefficient approach. A more desirable paradigm is to use a depth-first tree search strategy supported by some form of *a priori* pruning so that fruitless branches can be found as early as possible (see van Hentenryck (1989) for further discussion). An alternative approach is to use constraint propagation techniques (also sometimes referred to as label inference). Here variables range over a set of values, and constraints are used to restrict this set of values (see Mackworth 1977, Mackworth and Freuder 1985 and Mohr and Henderson 1986). This is the approach that is advocated here. Constraints are used to "prune" the location sets of tesseral addresses associated with entities so that a final time-table (or set of time-tables), which satisfies all the constraints, is arrived at.

There are many other approaches which have been used to address the scheduling problem. Of note are "branch and bound", Knowledge Based Systems (KBS), dispatching rules, Constraint Logic Programming (CLP) and graph colouring. Branch and bound refers to a strategy of resolving scheduling problems by pruning the potential search space through the calculation of upper bounds on the final solution value (see Applegate and Cook 1991, Carlier and Pinson 1989 and Barker and McMahon 1985, for further discussion). A good review of the use of KBS for scheduling is given in Charalambous and Hindi (1991). Dispatching rules are essentially priority rules often used for factory scheduling applications. Blackstone et al. (1982) give a review of the use of dispatching rules for conventional job-shop scheduling (see also Khoshnevis and Qingmei (1990) and Sundaram and Fu (1988)). There is much current interest in CLP which has a much wider field of application than just the solution of CSP, for example many graphics languages are based on CLP techniques. A good review is given in Leler (1988). Finally graph colouring is a technique originally developed to identify the minimum number of colours necessary (referred to as the *chromatic* number) to colour all the vertices of a graph so that no two adjacent entities are coloured with the same colour (Welsh and Powell 1967). The technique has many applications including scheduling and is perhaps of particular relevance here because of its spatial connotation.

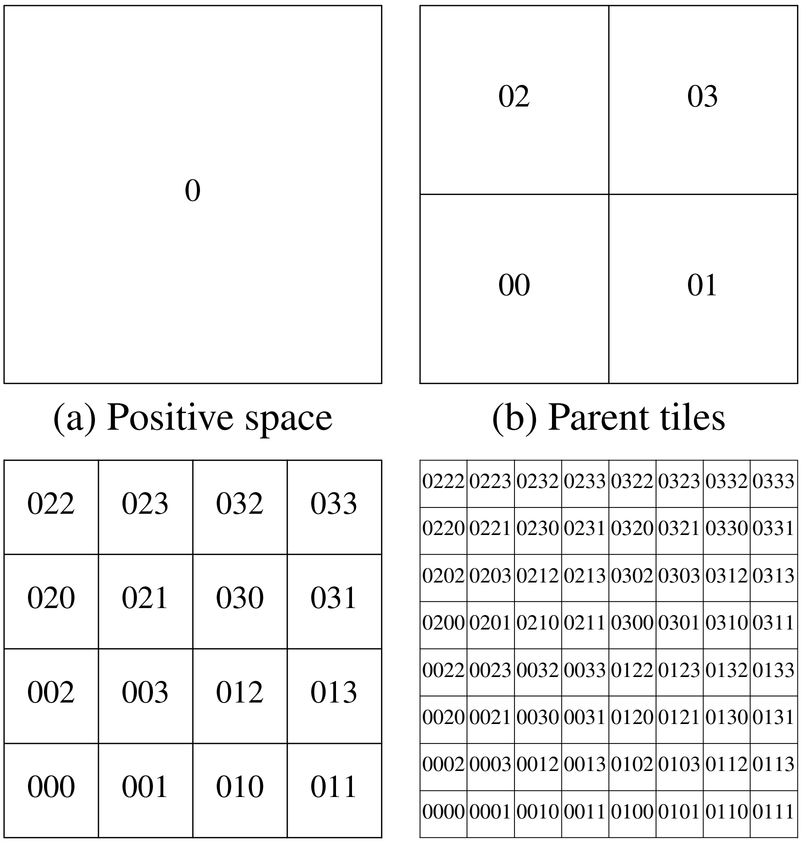
3. QUAD TESSERAL ADDRESSING

Tesseral addressing is based on ideas concerning hierarchical clustering developed in the 1960s and 1970s to improve data access times (Morton 1966, see also Samet 1984), and atomic isohedral (same shape) tiling strategies developed in the 1970s and 1980s concerned with group theory (Grunbaum & Shephard 1987). These two strands were brought together in the early 1980s when a *tesseral arithmetic* was defined for the representation (Holroyd 1983). There are many excellent works that deal in detail with the concept of tesseral addressing (see Diaz and Bell 1986, Gargantini 1983), however, to facilitate a complete understanding of this paper a minimal explanation of the Quad Tesseral representation is seen as essential and hence is presented here.

At its simplest tesseral addressing can be viewed as a method of labeling `tiles` in a two-dimensional space. The addressing mechanism is as follows. Given a square area (Figure 1(a)) we give this the address `0`. If we now quarter this area (i.e. *tesselate* it) each quadrant (tile) will be labelled as shown in figure 1(b) assuming that the origin is located at the bottom left hand corner. If we now also quarter these *parent* tiles we will produce *child* tiles as shown in Figure 1(c). Note that each child tile takes an address made up of a left hand symbol corresponding to its parents tesseral address and a right hand symbol corresponding to its position in relation to its siblings (i.e. `0`, `1`, `2` or `3`). In a similar manner we can produce grandchild nodes (Figure 1(d)) and so on until some desired depth of tesselation is arrived at with new tesseral addresses generated by always appending to the right of the parent tesseral address.

3.1. Linearisation and the Morton sequence

One of the principal advantages offered by tesseral representations is that they serve to linearise the entity space of interest. This is achieved by ordering the tesseral addresses into a decimal sequence. If we consider the tesseral addresses given in Figure 1(d) to represent integers to the base four we can convert these to decimals (Figure 2(a)) which then describe what is termed the Morton sequence (Figure 2(b)) after Morton (1966). Effectively, the result is that the space can be unravelled to form a ribbon one tile wide and of a length equivalent to the number of tiles into which the space is tesselated. In this paper Morton sequences will be indicated using the infix operator '..', whose operands are two tesseral addresses, the first representing the start of the sequence and the second the end of the sequence. Thus the tesseral sequence given in Figure 2 can be expressed as `0000 .. 0333`.

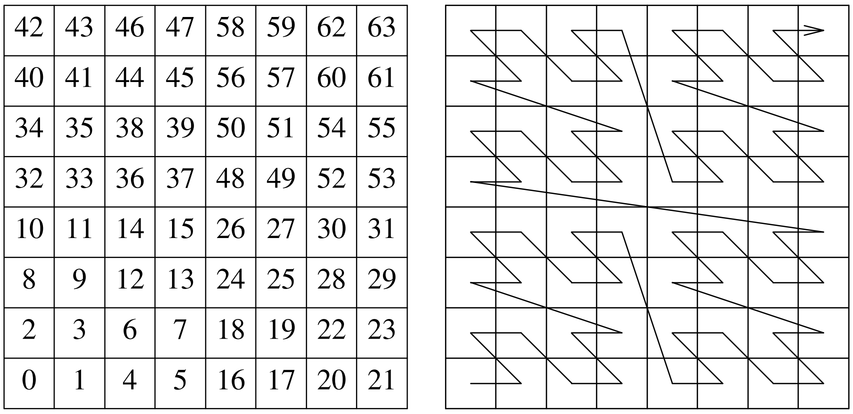


(a) Positive space (b) Parent tiles

(c) Child tiles (d) Grand-child tiles

Figure 1: *Tesseral addressing*

Note that sequences of the above form can be considered to represent sets, or sub-sets, of tesseral addresses. In addition, knowledge of Morton sequencing provides for the simple comparison of such sets. For example to establish that `{a..b}`$\subseteq$`{c..d}` is true it is only necessary to compare `a` with `c` and `b` with `d` (`ac` and `bd`).



(a) Morton numbering (b) Morton sequence

Figure 2: *Morton sequencing*

3.2. Tesseral Addition and Subtraction

Linearisation of space and Morton sequencing are not the only advantages offered by the Quad Tesseral representation. The representation also has an arithmetic associated with it (Holroyd 1983) which facilitates translation through the space in any desired direction simply by applying (adding or subtracting) a tesseral *displacement*. From Figure 1(b) we can see that to move (translate) from tile `00` to the right we must add `1`; similarly to move up or diagonally to the top-right hand corner of the entity space we add `2` or `3` respectively. By generating an addition table (Table 1) we can determine the result of adding any tesseral displacement to any tesseral address.

+	0	1	2	3
0	0	1	2	3
1	1	10	3	12
2	2	3	20	21
3	3	12	21	30

Table 1: *Tesseral addition table*

The generation of this table is well understood and is described in Bell et al. (1983). Note that in the table where a look up gives two digits the left most digit represents a carry. Note also that multi-digit addition is performed using the place system in a fashion exactly analogous to ordinary (base 10) arithmetic. Thus, from Figure 1(d), if we wish to translate 3 tiles to the right and 2 tiles up from the origin we will arrive at tile `0031`. If we wish to make the same translation from (say) tile `0023` we must add the tesseral displacement `0031` to the address `0023`. From the table this will produce the expected tesseral address `0302`.

Subtraction can be implemented in a similar manner.

3.3. Negative Tesseral Addresses

Using tesseral addition we can translate to the right, upwards and diagonally in a broad bottom-left to top-right direction. To translate in the opposite directions we can use tesseral subtraction. However, the only mechanism whereby we can translate in the broad bottom-right to top-left direction is to add a negative tesseral address (an address not included in the positive quadrant of Cartesian space shown in Figure 1). Thus to support translation in every direction we also need to tesserally label the negative quadrants of Cartesian space. In addition this must be done in such a manner that the arithmetic is still supported. In Figure 1(a) we labelled the positive quadrant `0` which gave rise to the parent tesseral addresses `00`, `01`, `02` and `03` (Figure 1(b)). Assuming a constant depth of resolution (i.e. all tesseral addresses are made up of the same number of symbols) and with reference to Table 1, to move from tile `01` to tile `00` we must add the tesseral offset `11` (ignoring any carry beyond the address length). Thus, in Figure 1(b), the tile immediately to the left of tile `00` must be tile `11` and hence the quadrant immediately to the left of the positive quadrant must have the label `1`. Using the same process of deduction we can also determine the labels for the remaining two negative quadrants (Figure 3(a)). The entire Cartesian space can now be labelled as before (see Figures 3(b) and 3(c)). As a result it is possible to translate in any direction through a tesselated space by adding or subtracting positive or negative tesseral offsets as appropriate.

1	0
3	2

(a) Cartesian Space

12	13	02	03
10	11	00	01
32	33	22	23
30	31	20	21

(b) Parent tiles

122	123	132	133	022	023	032	033
120	121	130	131	020	021	030	031
102	103	112	113	002	003	012	013
100	101	110	111	000	001	010	011
322	323	332	333	222	223	232	233
320	321	330	331	220	221	230	231
302	303	312	313	202	203	212	213
300	301	310	311	200	201	210	211

(c) Child tiles

Figure 3: *Tesseral addressing in all four quadrants*

4. DECLARATIONS

In the introduction to this paper we noted that time-tabling problems are presented to the TIMETABLER system in the form of a script comprising a declaration and a constraint part. The declaration part of this script will be considered in this section and the constraint part in the following section.

In the declarations part the entities (resources) which we wish to consider are defined. Each entity is considered to poses three attributes:

1) A name attribute describing the nature of the resource, e.g. buses, times, operators, etc.
2) A shape attribute describing the shape of the area taken up by the entity in the entity space. To this end two types of entity are identified, *tile* and *zone* entities, where a tile entity is a unit entity which has a shape corresponding to a single tile, and a zone entity has a shape corresponding to two or more tiles.
3) A location attribute indicating the entities location within the entity space. To this end there are, again, two possibilities, an entity can be a *fixed* entity or a *free* entity. In the first case the entity has associated with it a definite location, while in the second case the entity has a number of candidate locations.

The general form of a declaration statement is as follows:

```
entity: NAME <SHAPE> <LOCATION>
```

where `entity:` is a keyword defined for use by the system's parser to indicate that what follows is an entity declaration, `NAME` is an entity identifier, `<SHAPE>` defines the shape of a entity and `<LOCATION>` the location of the entity. The sets of tesseral addresses defining the shape or location attribute of an entity are defined in terms of the following constructs.

1) Single tesseral address, e.g. `101`.
2) Sequences of tesseral addresses expressed using the '..' infix operator described in Subsection 3.1, e.g. `000..003`.
3) Box constructs described using the syntax `box: TA1 TA2` where `TA1` represents the south-west corner of the box and `TA2` the north-east corner, e.g. `box 000 021` which

describes the set of tesseral addresses `000`, `001`, `002`, `003`, `020` and `021`.

4) Line constructs described using the syntax `line: TA1 TA2` where `TA1` represents the start of the line and `TA2` the end of the line, e.g. `line 000 013` which describes the set of tesseral addresses `000`, `001`, `012` and `013`.

We will illustrate the above using a representative school time-tabling scenario. Let us assume that our school has five rooms available (say two class rooms, a laboratory, a workshop and a sports hall) and that the school week is divided into five days each comprising 6 slots. If we consider the 30 available slots to be represented by the X-axis, and the Y-axis to be the available rooms, a 30×5 entity space is defined which might be described using a box construct of the `box 000000 011301`. This set of tesseral addresses then describes the universal set of tesseral addresses for our application. In the rest of this paper we will indicate this set using the variable `E`. Note that all addresses within this set are of length 6, however, for reasons of brevity, where appropriate, we will omit leading zeroes. Thus the address `000012` will be written simply as `12`.

We will now "declare" a number of example entities that we might wish to include in our school time-table application.

EXAMPLE 1 (fixed entities):

In the definition of the above time-table scenario each day comprises a 5×6 rectangle of tesseral addresses, each with a particular location in the entity space. We declare these "day" entities as follows:

```
entity: monday [box 0 301] [box 0 301]
entity: tuesday [box 0 301] [box 110 1211]
entity: wednesday [box 0 301] [box 1100 10201]
etc.
```

Note that the location attribute describes the same "shape" as the shape attributes thus the entities are all fixed entities in that the location attributes do not allow for any alternatives in which the given shape will fit.

The room entities (resources) to be considered in our time-tabling scenario may be declared in a similar manner, but using line constructs:

```
entity: lab [line 0 11101] [line 0 11101]
entity: gym [line 0 11101] [line 2 11103]
etc.
```

Again the "shape" defined by the location attributes is the same as that defined by the shape attributes thus the room resources are all fixed entities.

EXAMPLE 2 (free entities):

The remaining resources we wish to fit into the time-table, e.g. lessons and teachers, are all free entities. If we assume that lessons can either be single lessons or "double" lessons these can be defined as follows:

```
entity: maths1 [0] E
entity: science1 [0, 1] E
etc.
```

where the entity maths1 is a single lesson and the entity science1 is a double lesson. Here the entities are considered to be free entities in that the shape definition can be fitted into the location definition in more than one manner, 150 in the case of the entity maths1 and 145 in the case of the entity science1.

Teachers can be declared in a similar manner:

```
entity: smith [0] E
entity: jones [0] E
etc.
```

5. CONSTRAINTS

Constraints are used to express the relationships that are required to exist between the time-table entities (resources) of interest. More specifically they are used to express the requirement that an entity is located *within* or *notWithin* some area. The area in question is defined by applying a set of tesseral offsets to the location of some other entity. The general form of a constraint statement is as follows:

```
constraint: NAME1 <RELATION> NAME2 <OFFSETS>
```

where constraint: is a reserved words used by the TIMETABLER parser, NAME1 and NAME2 are entity labels, <RELATION> indicates the nature of the relation (within or notWithin) and <OFFSETS> the set of tesseral offsets to be applied described in terms of the same set of constructs used to define the shape and location attributes considered in the previous Section.

Again application of the above can best be illustrated by considering a number of examples.

EXAMPLE 3 (within):

Suppose we have a lesson entity music1 which can only be taught on a Wednesday (possibly because the subject is taught by a peripatetic music teacher who is only available on a Wednesday) this would be expressed as follows:

```
constraint: music1 within wednesday [0]
```

Here the tesseral address 0 describes the offset to be applied to the location tesseral addresses associated with the entity wednesday. Because the offset is zero the resulting area will coincide with the location set of tesseral addresses for the entity wednesday thus we are effectively describing an "equals" constraint. Given that the entity wednesday has a location set of tesseral addresses associated with it of the form box 1100 10201 (see Section 4) the constraint expresses the restriction that the entity music1 must be located within the same set of tesseral addresses.

EXAMPLE 4 (notWithin):

To express the constraint that an entity, physics1, must not occur immediately before or after (in the X direction) another entity, chemistry1, we would express this as follows:

```
constraint: physics1 notWithin chemistry1 [line 111111 1]
```

Here the construct line 111111 1 describes an area one tile before to one tile after the entity chemistry1. Thus if the entity chemistry1 is located at address 131, physics1 cannot be located anywhere along the line 130 (131 + 111111) to 1020 (131 + 1). Note that the tesseral address 111111 is a negative tesseral address in a 6 digit tesselation (see sub-section 3.3).

6. OPERATION

Individual constraints are resolved using simple set operations (equal to or subset of ($\subseteq$), not equal to or subset of ($\subseteq$) etc.). Full use is made of Morton sequencing when attempting to satisfy such set operations. Thus given the constraint:

```
constraint: maths1 notWithin maths2 [line 222200 200]
```

where maths1 is a free entity with a candidate set of tesseral addresses E and maths2 is a fixed entity located at address 1121 then the constraint is resolved as follows:

```
E ⊆ (1121 + [line 222200 200]) =
E ⊆ [line 223321 1321] (offsets applied to 1121 to give new line) =
E ⊆ [line 1101 1301] (addresses outside the entity space removed) =
[box 0 1300, box 1110 11301]
```

More generally the system operates using a constraint propagation paradigm. We commence with a start configuration generated according the declarations provided. This configuration essentially comprises the given set of entities and their individual location sets of tesseral addresses. Constraints are then used to prune this configuration. As a result of satisfying a constraint one or more possible configurations will result, i.e. one or more branches in a *solution tree* will be created. Consider the following declarations:

```
entity: wednesday [box 0 301] [box 1100 10201]
entity: music1 [0] E
entity: smith [0] [box 1100 10201]
```

and the constraint:

```
constraint: english1 within smith [0]
```

There are 30 possible locations for the entities `smith` and `english1` under which this constraint will hold, i.e. as a result of satisfying the constraint the possible locations for the entity, `english1`, will have been pruned from 150 to 30, but with the undesirable result that a node from which 30 branches "sprout" has been added to the solution tree. However, if we consider the constraint:

```
constraint: english1 within wednesday [0]
```

satisfaction of this constraint will also cause the locations for the entity, `english1`, to be pruned from 150 to 30 but in this case the 30 options will be defined as a single set of candidate location tesseral addresses giving rise to a node in the solution tree with only a single branch sprouting from it - a more desirable result.

The distinction here is that the entity `smith` is a free entity which has 30 possible locations associated with it while the entity `wednesday` is a fixed entity which has only 1 location associated with it. Given that we can calculate the maximum number of branches that a constraint will produce in advance this knowledge can be used to select constraints in such a manner so as to limit the growth of the solution tree.

If a constraint cannot be satisfied the current branch in the solution tree will be terminated and the system will "backtrack" to follow other branches. If no more branches are available the time-tabling problem will be considered to have no solution. Otherwise the system will proceed in the above manner until all constraints have been satisfied. The final leaf nodes to the solution tree will then contain solutions to the time-tabling problem under consideration.

7. CONCLUSIONS

In this paper we have described what we believe to be an innovative and novel approach to the time-tabling problem based on a spatial reasoning paradigm that uses the Quad Tesseral representation. Features of note are:

- The approach described very closely matches that followed by "expert" time-tablers in the field, as opposed to an approach advocated by many other systems which abstracts away from this practice.
- The end result is a graphical representation precisely analogous to a paper time-table/chart.
- A solution tree is maintained which can be traced and queried.
- All possible solutions are generated as opposed to simply "a" solution.

In addition the spatial reasoning technique used can be scaled up to address spatial reasoning in higher dimensions thus opening the possibility of addressing scheduling problems using multiple (more than two) axes to represent multiple resources.

The current version of the TIMETABLER system has been tested on a number of applications of the "students and courses" type, giving results which the research team have found greatly encouraging. Currently the research team are working on the inclusion of desirability factors so that the time-tables/charts produced can be quantified accordingly and hence optimised, and on techniques to increase the expressiveness of the constraints used.

REFERENCES

1. Appleby, J.S., Blake, D.V. and Newman, E.A. (1960). Techniques for Producing School Time-tables on a Computer and Their Application to Other Scheduling Problems. The Computer Journal, Vol 3, pp237-245.
2. Applegate, D. and Cook, W. (1991). A Computational Study of the Job-Shop Scheduling Problem. OSRA Journal of Computing, Vol 3, No 2, pp149-156.
3. Aubin, J. and Ferland, J.A. (1989). A Large Scale Time-tabling Problem. Computers and Operations Research, Vol 16, pp67-77.
4. Barraclough, E.D. (1965). The Application of a Digital Computer to the Construction of Time-tables. The Computer Journal, Vol 8, pp136-146.
5. Barker, J.R. and McMahon, G.B. (1985). Scheduling the General Job-Shop. Management Science, Vol 31, No 5, pp594-598.
6. Beck, H. (1991). An overview of AI Scheduling in the UK. In Graham, I.M., and Milne, R.W. (eds), Research and Development in Expert Systems VIII, Proceedings of ES'91, pp228-243.
7. Berghuis, J., van der Heiden, J. and Bakker, R. (1964). The Preparation of School Time-tables by Electronic Computer. BIT, Vol 4, pp106-114.
8. Blackstone, J.H., Philips, D.T., and Hogg, G.L. (1982). A State-of-the-art Survey of Dispatching Rules for Manufacturing Job Shop Operations. Int J. Prod. Res., Vol 20, No 1, pp27-45.
9. Carlier, J. and Pinson, E. (1989). An algorithm for Solving the Job-Shop Problem. Management Science, Vol 35, No 2, pp164-176.
10. Charalambous, O. and Hindi, K.S. (1991). A Review of Artificial Intelligence-based Job-Shop Scheduling. Annals of the CIRP, Vol 34, No 1, pp413-417.
11. Collyer, K.A. and Hamazaki, T. (1991). AI-Base Factory Scheduling Applications. In Graham, I.M., and Milne, R.W. (eds), Research and Development in Expert Systems VIII, Proceedings of ES'91, pp244-251.
12. Diaz, B.M. and Bell, S.B.M. (1986). Spatial Data Processing Using Tesseral Methods. Natural Environment Research Council publication, Swindon, England.
13. Duncan, T. (1994). Intelligent Vehicle Scheduling: Experience with a Constraint-Based Approach. In Milne, R. and Montgomery, A. (Eds.), Applications and Innovations in Expert Systems II, Proceedings of ES'94 Applications Stream, SGES Publications, Oxford, pp281-291.
14. Ferland, J.A. and Roy, S. (1985). Time-tabling Problem for University as Assignment of Activities to Resources. Computers and Operations Research, Vol 12, pp207-218.
15. de Gans, O.B. (1981). A Computer Time-tabling System for Secondary Schools in the Netherlands. European Journal of Operational Research, Vol 7, pp175-182.
16. Gargantini, I. (1983). An effective way to Represent Quadtrees. Communications of the ACM, Vol 25, No 12, pp905-910.
17. Grunbaum, B. and Shephard, G.C. (1987). Tilings and Patterns, Freeman, New York.
18. van Hentenryck (1989). Constraint Satisfaction in Logic Programming. MIT Press.
19 Holroyd, F.C. (1983). The Geometry of Tiling Hierarchies. Ars Combinatoria, Vol 16B, pp211-244.
20. Khoshnevis, B. and Qingmei C. (1990). Integration of Process Planning and Scheduling Functions. Journal of Intelligent Manufacturing, Vol 1, pp165-176.
21. Lamma, E., Mello, P., Milano, M. and Torreggiani, A. (1994). A Constraint Logic Programming Approach to Train Scheduling Problem. In Milne, R. and Montgomery, A. (Eds.), Applications and Innovations in Expert Systems II, Proceedings of ES'94 Applications Stream, SGES Publications, Oxford, pp293-306.

22. Laporte, G. and Desroches, S. (1986). The Problem of Assigning Students to Courses Sections in a Large Engineering School. Computers and Operations Research, Vol 13, pp387-394.

23. Leler, W. (1988). Constraint Programming Languages: Their Specification and Generation. Addison-Wesley, Reading.

24. Mackworth, A.K. (1977). Consistency in Networks of Relations. AI Journal, Vol 8, No 1, pp99-118.

25. Mackworth, A.K. and Freuder, E.C. (1985). The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems. Artificial Intelligence, Vol 25, pp65-74.

26. Morton, G.M. (1966). A Computer Oriented Geodetic Data Base, and a New Technique in File Sequencing. IBM Canada Ltd., 1 March 1966.

27. Mohr, R. and Henderson, J.C. (1986). Arc and Path Consistency Revisited. Artificial Intelligence, Vol 28, pp225-233.

28. Palmer, G.J. (1994). The Limitations of a Rule-based Approach to Integrated Manufacturing Planning. In Bramer, M.A. and Macintosh, A.L. (Eds.), Proceedings ES'94, SGES Publications, Oxford, pp317-330.

29. Samet, H. (1984). The Quadtree and Related Data Structures. ACM Comput. Survey, Vol 16, No 2, pp187-260.

30. Sundaram, R.M. and Fu, S.-S (1988). Process planning and Scheduling - a Method of Integration for Productivity Improvement. Computers and Industrial Engineering, Vol 15, No 1-4, pp296-301.

31. Welsh, D.J.A. and Powell, M.B. (1967). An Upper Bound for the Chromatic Number of a Graph and its Application to Time-tabling Problems. The Computer Journal, Vol 10, pp85-86.