RULE-BASED CARTOGRAPHIC NAME PLACEMENT WITH PROLOG

Christopher B. Jones
Anthony C. Cook
Polytechnic of Wales
Pontypridd
Mid Glamorgan
CF37 1DL, UK

ABSTRACT

Positioning text on maps is a complex task subject to numerous rules which may vary according to the purpose of the map and who is making it. Ideally therefore, automated systems for name placement should be flexible in terms of the rules used to control them. Because the logic programming language Prolog is essentially rule-based, it appears to be appropriate for developing such systems. In practice, it has proved possible to use the language for writing relatively short and clear programs to find non-conflicting positions for names and for specifying rules for selecting names and potential label positions. An experimental Prolog name placement system has been developed and has been used to place names on a variety of maps. The system provides access to a spatial database which facilitates the creation of rules which depend upon spatial relationships between names and other map features.

INTRODUCTION

The results of cartographic name placement can vary greatly according to the theme, purpose and style of the map onto which names are to be placed. In general, name placement may be regarded as subject to sets of rules which differ from one type of map to another, as well as between cartographers and cartographic organisations. The concept of rules in name placement was integral to Imhof's (1975) review of the subject and it has been incorporated in automated systems such as those of Freeman and Ahn (1984) and Pfefferkorn et al (1985). A major problem with implementing automated rule-based systems is to find a way of programming the rules such that they can be changed easily to meet user requirements. With this objective in mind, the logic programming language Prolog (e.g. Clocksin and Mellish, 1981) is of particular interest, as it uses rules and associated facts as its basic constructs, which are referred to as predicates.

In being a declarative, rule-based language, Prolog appears to offer potential for programming name placement at a relatively high level, provided that the cartographic rules can be translated into the syntax of Prolog. Thus the appeal of logic programming for name placement is the possibility of writing programs which consist of descriptions of rules rather than the algorithms for implementing them. Ideally these programs should be short, readable and easily amended. It is important to realise in this context that some of the apparent advantages of Prolog derive from the fact that the language uses a built-in inference mechanism to deduce solutions which obey the facts and rules which specify the problem. This may be contrasted with conventional procedural languages (such as Pascal, Fortran and C) which work by executing a set of instructions based on an algorithm which specifies how to solve a problem.

Name placement can be thought of as a combination of two processes, one of which is concerned with resolving conflicts between name positions (or labels), while the other is concerned with the selection of names and their associated label characteristics and positions. The remainder of this paper falls into two parts, corresponding to these two processes. In the first part a logic programming strategy is described for finding combinations of name positions which minimise conflict between names and between names and unrelated features. This is confined here to the problem of labelling point-referenced features and is based on Jones (1987). The second part relates to the problems of selecting names and of generating trial or candidate labels which satisfy rules of text configuration and of graphic association between the text and the named feature. Examples are given for name placement problems which include point, line and area-referenced names. They are based on recent research by one of the authors who has developed a name placement system which uses Prolog for conflict resolution and for implementing rules for selecting names and appropriate labels (Cook, 1988). This system, called NAMEX, uses a spatial database which may be accessed from Prolog by means of calls to subroutines which are written in Fortran.

PLACEMENT STRATEGIES FOR AVOIDING CONFLICT

In order to find a set of name positions which satisfies rules of conflict avoidance with other names and features, it is necessary to provide mechanisms for generating trial positions for each name to be placed and for detecting whether any given position results in conflict. Let us assume that there is a Prolog predicate, find_trial_position, which on being called returns an horizontal trial position for a label (Name) defined by location co-ordinates X, Y in a raster image co-ordinate system and by its Length, which is measured in pixels. This predicate could take the following form:

```
find_trial_position(Name, X, Y, Length).
```

The presence of horizontal labels which have been placed can be recorded with predicates of the form

```
label_at(Name, X, Y, Len).
```

where X and Y are the co-ordinates of the start of a row of pixels of length Len occupied by the label. Depending on the height of the text, several such predicates could be asserted for each label. For simplicity of explanation, we assume here that only a single row is required. To ensure that point symbols can be uniquely identified, so that a label's own symbol can be distinguished from others, the presence of symbols can be recorded in terms of rows of pixels defined by predicates of the form

```
symbol_at(Name, X, Y, Len).
```

where the parameters are as before. The presence of all other map features can be recorded in the Prolog database using additional fact predicates. One possible method is to encode map features in a run length format, as described in Jones (1987). Overlap or conflict detection can now be performed by a second predicate (no_conflict) which is proven to be true if no conflict is found between the given position and any other label and, if necessary, any other map feature. It can do this by examining the positions of labels already placed and, if appropriate, by searching the database to identify the presence of other map features which could be affected by this possible placement. It may be defined in terms of a rule which first attempts to prove that there is an overlap, in which case it fails, otherwise it is true. For the purpose of testing for overlap with other labels, this can be expressed as follows (note that the symbols /* and */ are used for delimiting comments, while :- means 'if' and , and ; mean 'and' and 'or' respectively):

```
no_conflict(Xs, Ys, Search_length, Ignore):-
/*          look for names or symbols on given scan line */
  (label_at(Name, X, Ys, Length) ;
   symbol_at(Name, X, Ys, Length)),
   not(Name=Ignore), /* should name or symbol be ignored */
                                /* now test for overlap */
   Search_end is (Xs+Search_length-1), Search_end >= X,
   Data_end is (X+Length-1),          Xs =< Data_end,
   !, fail.       /* fail completely if any overlap found */
/*       succeed only if not possible to prove overlap    */
no_conflict(_,_,_,_).
```

This definition can also be used in situations where all map features must be considered, provided that the trial positions given by the predicate find_trial_position have already taken account of them. Such an approach has the merit of performing all spatial data searches, for potential feature conflicts, once only in a preprocessing stage, rather than in the course of conflict resolution. Alternatively, as described in Jones (1987), the logic program database may be loaded with a complete description of the map, which can be searched when the no_conflict predicate is called. This may however introduce repetition of searches.

A relatively simple strategy can now be applied in which, for each name, a predicate place_label uses the above predicates to generate a possible position which is tested for overlap. If the position is found to be acceptable, it is recorded in the logic program database by means of a predicate record_position which asserts label_at predicates. When placing several names however, it may be impossible to place an individual name because other previously placed names are in conflict with it. Such a situation induces backtracking in which one or more previously placed labels will be removed and alternative trial positions used, before going forward to try again with the label which could not be placed. To enable this backtracking, the predicate for placing a label calls a predicate (clear_previous_label) which retracts from the Prolog database any previously asserted label_at predicates for that name, before any other is recorded. The place_label predicate may be described as follows:

```
place_label(Name):-
        find_trial_position(Name, X, Y, L),
        no_conflict(Name, X, Y, L),
        clear_previous_label(Name),
        record_position(Name, X, Y, L).
```

Placement of a group of labels can be done by applying the place_label predicate to all labels to be placed. One way of handling the group of labels is to put them in a list. Lists in Prolog can be referred to in terms of the first item in the list (Head) and the remainder of the list (Tail). When a list is handled in this way, it is symbolised by [Head|Tail] where the square brackets represent the boundaries of a list. Thus a predicate place_group, which attempts to place all names in the group, can be described in the following manner:

```
place_group([]).     /* terminating condition (empty list) */
place_group([Head_name|Tail]):-
   place_label(Head_name),     /* place first name in list */
   place_group(Tail).          /* place remaining names */
```

This predicate is defined recursively, such that when the first label has been taken from the head of the list of labels, it is passed to the predicate which attempts to place it (place_label) before passing on the list of remaining names to the original predicate (place_group). If place_label cannot find a suitable position, and hence fails, the place_group predicate backtracks to the last situation in which an alternative action was possible. Such a possibility would occur when another trial position was available for a previous name. Note that if a previous label cannot find another position, it will induce further backtracking to its predecessor in the list of names. The logic program will fail only when all combinations have been exhausted without success. Such complete failure can however be averted by deleting a name (such as the last one which failed to be placed, if all names have equal importance).

Reducing Search Time

When there are many names to be placed, program run times may be unacceptably long. Solution times can however be reduced by breaking the problem into sub-problems defined by groups of names which are in potential overlap with each other. This was done by Freeman and Ahn (1984), who referred to these groups as connected components. Further reduction in search time can be obtained by sorting trial label positions in order of decreasing difficulty, as also found by Freeman and Ahn. Possible measures of difficulty for a particular name include the number of trial positions available and the number of neighbouring names which are in potential overlap.

RULE-BASED SELECTION OF TRIAL LABELS

The techniques for conflict avoidance, as described in the previous sections, are based on the assumption that a mechanism exists whereby trial label positions are generated, either in advance or in the course of seeking non-conflicting positions. The process of selecting trial positions is one which has a major bearing on the style and, in many cases, the legibility of the resulting map. Trial positions are selected according to factors such as orientation, the relationship between the label and the feature which it annotates, and its relationship to other map features. The relationship between a label and its associated feature depends upon the class of feature concerned. For points, the label is typically placed immediately adjacent, to the right or left, above or below or at intermediate positions to these. Line labels might be placed on the line or to one side or the other of the feature, or perhaps lie across it. Area labels may be positioned inside a region at some orientation which could be related to the shape of the area, or they might be outside, adjacent to the edge, such as when the area is too small to accommodate the label.

The combination of factors which determine trial positions, along with issues of whether or not particular features should be labelled at all, provide enormous scope for controlling the design of a map. In an automated system, these factors should therefore be subject to rules which can be adjusted by the cartographer. In the NAMEX name placement system (Cook, 1988), an attempt has been made to demonstrate how, with the aid of logic programming, such rules can be implemented in a manner which allows them to be changed relatively easily. In the present implementation, such changes still depend in many cases upon a knowledge of Prolog programming but, in the ideal situation, it is possible to envisage a user interface which allowed rules to be changed via a natural language dialogue.

Before looking at examples of Prolog rules in NAMEX, it should be noted that the system is a hybrid one in terms of programming language, in that a number of low level functions, concerned with tasks such as accessing a spatial database and detecting overlap, have been implemented in Fortran using subroutines which can be called from Prolog. Of particular significance for the implementation of spatially defined rules are functions which determine the presence of specified map features within a given vicinity (e.g. a circle or rectangle). It should also be remarked that the NAMEX conflict resolution strategy differs somewhat from that which was described in the previous section.

Label Selection in NAMEX

Rules for generating trial labels in NAMEX fall into three categories. The first is concerned with the selection of names for which label positions are sought, the second defines label configuration, which relates to text size, orientation and position relative to the named feature, while the third is concerned with validation of the selected configurations. Validation ensures that selected configurations obey rules of association between labels and adjacent features. Thus there may, for example, be priorities and limits controlling what features may be overlapped, and what distances there may be between labels and nearby features.

The extent to which rules must be provided for the initial selection of names depends upon the application. In some situations, it may be desirable to attempt to plot all names, in which case selection may depend upon either an initial analysis of the map space and name density (as in Langran and Poiker, 1986), or perhaps on a system of priorities or ranks which must be considered when the conflict resolution strategy encounters difficulties (at present, the NAMEX conflict resolution strategy can delete names irrespective of their cultural importance). There are also situations in which it is appropriate to specify rules which govern selection of features to be named according to the theme of the map and on the basis on map scale.

At a very simple level of selecting major feature types, a predicate name_select(Fsn,Ftype) has been used, in which Fsn is a unique feature serial number, and Ftype is the feature type (point, line, area). In order to ensure that, say, all point features are to be selected, the predicate name_select(_,point) may be asserted. If the feature name under consideration has the corresponding value for Ftype then, when the predicate is called for the given name, it will succeed (i.e. be regarded as true). More sophisticated rules which take account of scale have been used in the creation of a series of Moon maps in which craters are labelled if their size exceeds a threshold which is determined by an empirical function dependent upon map scale. For example, a predicate for selecting primary craters on maps of scale smaller than 1:15,000,000 may be defined as follows:

```
select_crater(primary, Scale, Fsn):-
     Scale > 15000000,
     crater_diameter(Diameter, Fsn),
     generalise_crater_rule(Limit, Scale),
     Diameter >= Limit.
```

The predicate crater_diameter accesses the NAMEX database to determine the diameter of feature Fsn, while the generalise_crater_rule predicate calculates the diameter threshold Limit, on being given a value of Scale. The map in Figure 1 was created using such a generalisation rule.

Many of the rules of configuration selection may be implemented quite simply, since there may often be a direct and fixed relationship between feature type and the label's size, orientation and position relative to the feature. For example, control over the orientation of labels can be achieved with a predicate select_label_orientation(Fsn,Ftype,Fcode,Orientation), in which Fcode specifies a particular class of feature of specified type, and Orientation may be either horizontal or diagonal. Once a diagonal orientation has been selected, the angle employed is calculated as a function of the disposition of the feature. Normally the first three parameters would be predetermined for a given label. Calling the predicate will then result in Orientation being instantiated with an an appropriate value. In the case of a rule which specified that all point-referenced labels were to be plotted horizontally, it would be sufficient to use the predicate

```
select_label_orientation(Fsn,point,Fcode,horizontal).
```

Thus, on calling the predicate, all point features will be given a horizontal orientation. The orientation of line labels may often be subject to a variety of rules. If it was required to set labels to be horizontal for all roads with feature code 'a_road', it could be done with the predicate

```
select_label_orientation(Fsn,line,a_road,horizontal).
```

If however the orientation of 'a_road' labels was to be a function of direction, the predicate would have to access the database to examine the properties of the specified feature. The following rules would set the orientation to be horizontal if the line was orientated more steeply than 40 degrees, and otherwise diagonally.

```
select_label_orientation(Fsn,line,Fcode,Orientation):-
       select_line_label_orientation(Fsn,Fcode,Orientation).

select_line_label_orientation(Fsn,a_road,Orientation):-
       compute_angle_of_line(Fsn,Angle),
       select_orient_by_angle(Angle,Orientation).

select_orient_by_angle(Angle,diagonal):-
       Angle =< 40.

select_orient_by_angle(Angle,horizontal):-
       Angle > 40.
```

Rules of this sort were used in placing major ('A') road names on the road map in Figure 2. Similar rules have been implemented for controlling the orientation of area labels, as in Figure 3. Here it was required to determine the label orientation according to the degree of elongation and the orientation of the area itself, both of which were found by calls to standard NAMEX predicates.

An important aspect of name placement is that of maintaining a clear association between the label and the feature it annotates. As an example, the predicate valid_position, described below, ensures that a point label does not overlap too many other features or lie too close to any adjacent point symbols, other than its own.

```
valid_position(Fsn,Ftype,Fcode,Pos_number):-
/* get location of label at given position number          */
      read_label_data(Fsn,Ftype,Fcode,Pos_number,
                          X,Y,Angle,Length,Height,Prox),
/* examine underlying pixels in rectangle centred on X,Y  */
      raster_rectangle_totals(X,Y,Angle,Length,Height,
                     Number_of_pixels,Total_pixel_value),
      Ratio is realof(100 * Total_pixel_value /
                               Number_of_pixels),
/* find current acceptable threshold value for ratio      */
      dense_space_threshold(Thresh_dense),
/* test ratio against threshold                           */
      Ratio =< Thresh_dense,
/* enlarge label region by Prox to include buffer zone    */
      enlarge_label(Length,Height,Prox,Newlength,Newheight),
/* test for absence of illegal features in label region   */
      raster_rectangle_features_absent(X,Y,Angle,
                Newlength,Newheight,[city,town,village]).
```

The predicates called by valid_position are not defined here. Some of them make calls to lower level predicates which access the contents of the NAMEX database. Note that the total pixel value found by raster_rectangle_totals is obtained from the sum of priority weighted raster bit planes representing map features, and that the test for illegal pixel values in the extended region of the label depends upon the pixels of the label's feature having been temporarily masked. The logic described here was used in creating Figure 2.

Comparable techniques to these can also be used to good effect in validating on the basis of other cartographic criteria which require spatial search. An example occurs in the county map (Figure 3), in which settlement names away from the coast have been placed either inside or mostly inside the county to which they belong. The rule was formulated with reference to searches in the rectangular regions formed by the co-ordinates of the named settlement and the centre of the label itself. Substituting the feature type 'county boundary' into the list of features to search for in the raster_rectangle_features_absent predicate, it was possible to test whether a county boundary lay between the label centre and its point symbol. If the county boundary was also a coastline, then the rule was not applied. To keep a label entirely in its own county, the search rectangle could be enlarged appropriately.

CONCLUSIONS

The logic programming language Prolog has been used for writing rule-based programs which perform cartographic name placement. The built-in inference mechanism of the language makes it possible to produce relatively short and readable programs which find positions for groups of names by obeying simple rules of conflict avoidance. Rules for the selection of labels to be placed, and the selection of suitable label configurations, have also been implemented in a name placement system (NAMEX) which provides access to a spatial database for the purpose of evaluating potential label positions. NAMEX has been used to place names on a variety of maps subject to quite different rules. Useful future developments of the system include more sophisticated conflict resolution strategies governed by rules controlling the deletion of names in crowded areas, and a high level user interface allowing rules to be encoded via a natural language dialogue.

ACKNOWLEDGEMENTS

The authors are grateful for financial and material assistance from the Ordnance Survey, Southampton. ACC was supported by an SERC studentship.

REFERENCES

Imhof, E., 1975, "Positioning Names on Maps", The American Cartographer, Vol. 2, No. 2, pp. 128-144.

Clocksin, W.F. and C.S. Mellish, 1981, Programming in Prolog, Springer-Verlag.

Cook, A.C., 1988, "Automated Cartographic Name Placement Using Rule-Based Systems", PhD Thesis, Polytechnic of Wales.

Freeman, H. and J. Ahn, 1984, "AUTONAP - An Expert System for Automatic Map Name Placement", Proceedings International Symposium on Spatial Data Handling, Zurich, pp. 544-569.

Jones, C.B., 1987, "Cartographic Name Placement With Prolog", manuscript, Dept. Computer Studies, Polytechnic of Wales.

Langran, G.E. and T.K. Poiker, 1986, "Integration of Name Selection and Name Placement", Proceedings Second International Symposium on Spatial Data Handling, Seattle, pp. 50-64.

Pfefferkorn, C., D. Burr, D. Harrison, B. Heckman, C. Oresky and J. Rothermel, 1985, "ACES: A Cartographic Expert System", Proceedings Auto-Carto 7, Washington DC, pp. 399-407.

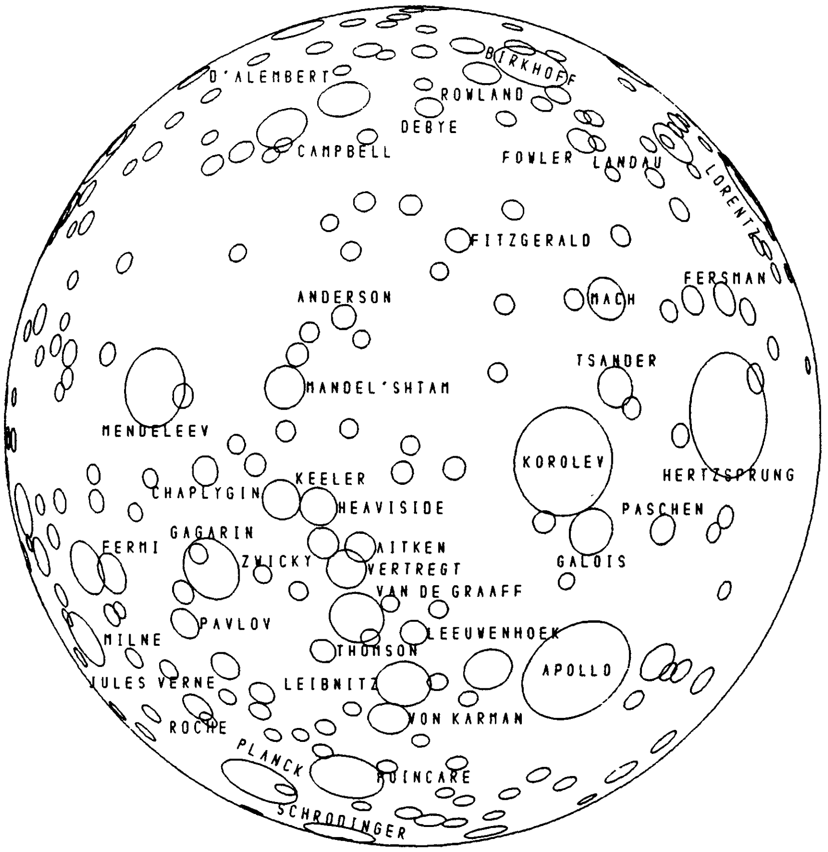

Figure 1. Far side of the Moon

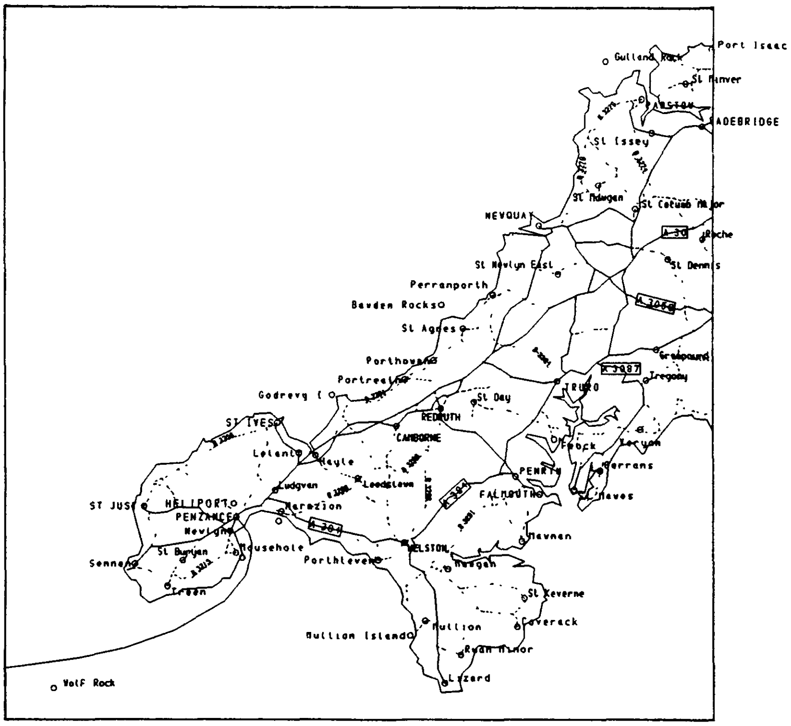
Figure 2. Road map (using Ordnance Survey data)

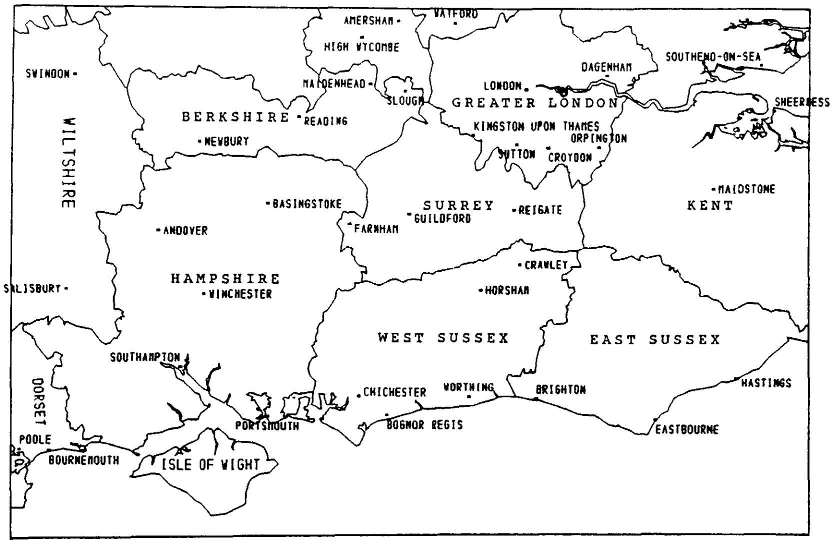

Figure 3. County map (using Ordnance Survey data)