CIAT

Centro Internacional de Agricultura Tropical
International Center for Tropical Agriculture

A Classifier System Prototype for the Exploratory Analysis of Spatial Data

Project Report

Euan Crawford and William Bell

July 1997

CGIAR Consultative Group on International Agricultural Research

A Classifier System Prototype for the Exploratory Analysis of Spatial Data

Project Report

Euan Crawford
CIAT / University of Edinburgh
July 1997

This report details work conducted as part of a CIAT funded project investigating exploratory analysis of spatial data using artificial intelligence techniques. The study was conducted in collaboration with Dr William Mackaness (Department of Geography, University of Edinburgh) and Dr William Bell (Chief Information Officer, CIAT).

1. Introduction

As GIS emerges as a mature technology for the management and manipulation of spatial data the amount of digital data to be serviced has exploded. As we generate greater quantities of data at increasingly higher resolutions the *opportunities* to analyse, model and understand environmental and socio-economic processes increase. However, it appears that our ability to make use of data has been swamped by our ability to create it. Current spatial analysis techniques, whether conceptual or widely used, are not powerful enough to allow us to 'release the empirical regularities that exist amongst the noise' in large spatial datasets (Openshaw, 1994, p123). As Holland (1986, p593) notes;

'Human experience indicates that real environments abound in regularities. The problem is to uncover and exploit them'.

Many of our current techniques for the analysis of spatial data date from an era of limited computing power where analysis was concepts, rather than data, driven. As a result most spatial models are either too abstract to allow operationalisation (environmental systems modelling, theories of urban social structure) or too crude (gravity interaction modelling, urban economic zoning) to produce useful results (Senior, 1979). Many techniques from exploratory statistical analysis (EDA) also prove to be of limited use. They assume *a priori* knowledge of all potential relationships in the data and the strict assumptions of parametric statistics, as well as the need to analyse categorical data, frequently make EDA techniques inappropriate.

This study details the development and implementation of a prototype classifier system that searches for rules and relationships among spatial data.

2. Machine Learning

Over two decades of work in artificial intelligence has produced a number of paradigms aimed at inductive and deductive learning of rules from experience (see Wilson, 1987, for a review of relevant work). Many of these paradigms deal only with high level, symbolic representations of data ('a small field of dark soil') (Mitchell, 1982) in broad problem domains, of the type more commonly associated with human experience. The most successful systems which are able to deal with lower level concepts ('a 1 hectare polygon of soil of pH 6.1') have been developed for restricted problem domains where the learning environment lends itself to straightforward classification e.g. gaming systems. Neural networks have proved able to handle both high and low level concepts with notable success (Barto, 1985). However, with a non-transparent learning process little insight is gained into the processes that produce regularity in the data. The classifier system, developed by Holland (1975, 1986), avoids many of these limitations and holds great potential for learning of relationships in spatial data.

2.1 Classifier Systems

A comprehensive review of classifier systems and their operation is outwith the scope of this report and thus only relevant elements will be treated in depth. A fuller account is available in Holland (1975, 1986).
Holland classifiers are essentially expert systems that incorporate elements of inductive learning and rule creation. They are particularly adept at producing solutions to highly complex problems that cannot be solved analytically (NP-complete).

Holland classifier systems comprise three main elements;

- standard classifier system: a rule base and message board
- learning and induction system: bucket brigade algorithm
- rule discovery system: genetic algorithm

2.1.1 Standard Classifier

The standard classifier system uses a ternary alphabet {0,1,#} to represent data. Real world, non-binary data can be transformed to a binary representation. The third element in the alphabet , '#', is a wildcard character, used to confer a degree of generality on rules. A '#' is used to signify either a '1' or a '0'.

The rule base consists of a fixed number of rules, each one a hypothesis about some aspect of the environment (in this case a description of the relationships between spatial data). Rules, of similar form to those used in expert systems, are of fixed length L and include a condition and outcome segment. A rule with L=12 takes the form {011001100101}, where bits 1-8 represent the condition and bits 9-12 represent the outcome. The condition might represent a set of variables (soil X, aspect Y, slope Z) that lead to a particular outcome (landuse Q). An analogy can be drawn with the way chromosomes are used for the storage of genetic information in biological systems. In this case each bit corresponds to an allele. Input from the environment is coded using the same scheme. In this way rules and environmental data can be directly compared.

The message board is a list, accessible by all rules. It is used for posting environmental input and to allow inter-rule linking and communication.

2.1.2 Learning and induction system

The bucket brigade algorithm, as proposed by Holland (1976), allows competing rules to bid for the right to become the system decision. Each rule in the rule base is compared with environmental input posted to the message board. Matching is performed only on the condition part of chromosomes. The system decision is the rule which most closely matches an input. Rules are then rewarded according to their level of success in predicting the outcome, that is their ability to match the outcome segment of the input chromosome. While successful rules become progressively stronger poor performers lose strength and run the risk of being eliminated from the population by the genetic algorithm (as explained below).

The bucket-brigade algorithm is particularly effective in task environments where success may be infrequent and requires the construction of long chains of rules to achieve a single goal e.g. recognising, and acting upon, input to navigate a robot around a maze.

2.1.3 Genetic algorithm

The genetic algorithm (GA) provides the rule discovery element of the system. GAs essentially mimic the way biological populations use genetic evolution to become progressively fitter through successive generations. The GA operates, primarily, by combining elements from the best performing rules to produce new, fitter offspring.

New rules are bred by randomly combining segments from the chromosome of each parent. In the example below (L=12) the segments consisting of bits 5-8 are swapped to produce two hybrid offspring.

parents	offspring
rule_26 {11110000 1111}	rule_101{11110001 1111}
rule_27 {11010001 1010}	rule_102{11010000 1011}

The secondary operator, random mutation, precludes the permanent loss of, and ensures the creation of new, 'genetic material' in the system. In the above example rule_102 was subject to this process. As is the case with biological populations, random mutation occurs far less frequently than crossover.

Chromosomes can be considered as consisting of 'building blocks' of genetic material. The key to the success of GAs lies in the way they search the solution space. The combination of historically successful building blocks moves through this space 'orders of magnitude more rapidly than would be indicated by the rate at which it is processing strings' (Holland, 1986). This is particularly important for searches in spatial datasets. A modest exploratory search might use five layers of spatial data. Assuming that all data in these layers can be adequately represented using a 3-bit binary alphabet there exists, potentially, 2^{15} unique rules. A more complex search might use 15 datasets and a 5-bit ternary alphabet. With a solution space of 2^{75} unique combinations conventional searches are clearly inadequate. The benefits of a GA which is guided through search space by a combination of experience and implicit parallelism are apparent.

3. System Prototype

A system prototype was developed to test the applicability of the classifier system to the exploration of spatial data. This section documents the system design. The system incorporates elements from both the standard Holland classifier and Stewart's (1987) Animat classifier, with a number of modifications.

3.1 Data Representation

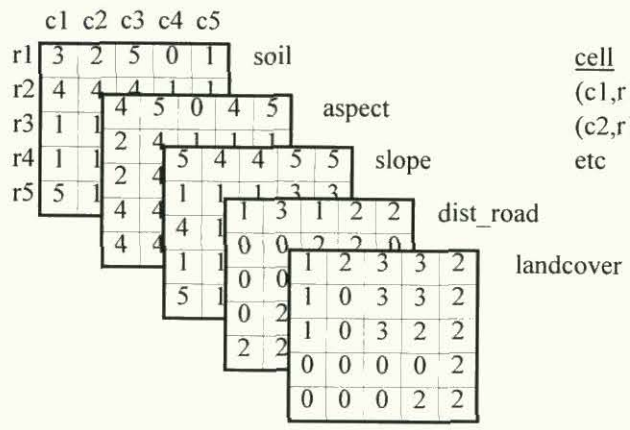
The use of a ternary alphabet (0,1,#) to represent rules and environmental input has the advantage of allowing the standardisation of many, potentially diverse, datasets. Coupled with the ability to use fixed length registers in the CPU for simple string matching rapid processing is possible. However, for ease of representation and interpretation of results, the system was designed for use with a decimal alphabet. System trials used rules and environmental input represented by the alphabet {0,1,2,3,4,5,6} with '6' acting as a wildcard (#).

3.2 Search

The system searches for rules by scanning the study area from upper left to bottom right. At each grid cell the values in the corresponding datasets are recorded and mapped to a single chromosome of the form described earlier (Figure 1). Comparisons are then made between

each rule in the rule base and the input string. The number of matches, including wildcard matches, is recorded for each rule. The remaining steps in the algorithm are carried out before preceding to the next cell.

Figure 1: Environmental Input (five 5*5 rasters)



As the search through the spatial data allows immediate evaluation of rule success, that is, success in providing a match, the full rule-chaining capabilities of the bucket-brigade algorithm are not required.

3.3 Rule Base Payoff

For any particular grid location, those rules which fully match the condition part of the input data are subset from the rule base [P] to form the condition set [C]. The rule with the greatest strength is then selected from [C], this becomes the system decision (*sd*) . The decision set [D] is formed as a subset of [C] and comprises the system decision and those rules with the same outcome as the system decision. The set NOT[D] is also formed. The outcome of [D] is tested against the input data to establish whether or not the system provided a correct decision. Depending on the decision, each member of [D] receives a proportion of a fixed quantity of payoff. The payoff received by individual rules is determined by the generality function defined below.

$$\text{payoff}_i = \frac{1+(\text{em}_i / \text{tm}_i)^g}{\sum_{1}^{n} 1+(\text{em} / \text{tm})^g} \cdot \text{tpayoff}$$

where payoff$_i$: payoff received by rule$_i$, em$_i$: number of non # matches in rule$_i$, tm$_i$: total number of condition matches in rule$_i$, g: system-wide generality control (scalar value) and tpayoff: total payoff available to [D].

Payoff to an individual rule is inversely proportional to the number of wildcard matches generated by it's condition. As g increases, those rules with a high number of wildcard matches receive proportionately less payoff. The generality control was introduced to control the production of overly general rules.

The system also includes the capability to levy taxes on rules. Taxes can be applied to [C], [D] and NOT[D] to control the way the system learns. Payoff with no taxation can be

considered a payoff-only regime while payoff with high taxes can be considered a payoff-penalty regime (Stewart, 1987).

3.4 Genetic Algorithm

After the rule base has been matched with the input string and rewarded accordingly, genetic operators are employed. The two operators, crossover and mutation, are controlled by two system wide probability functions; $1/cr$ and $1/mt$ respectively.

The algorithm selects the strongest rule from [P] and makes a copy. If crossover is invoked, the second strongest rule is also copied and a random segment is swapped between the two. The offspring are then inserted in place of the two weakest (lowest strength) rules in the population. If crossover is not invoked the copied rule is simply inserted in place of the weakest rule. In both cases the mutation operator is then applied to each allele in the offspring with probability $1/mt$.

4. System Trials

Testing was carried out using five artificially generated spatial datasets corresponding, arbitrarily, to soil, aspect, slope, distance from road and landcover data. The latter is treated as the outcome while the preceding four layers form the condition. Each dataset is stored as an ASCII raster.

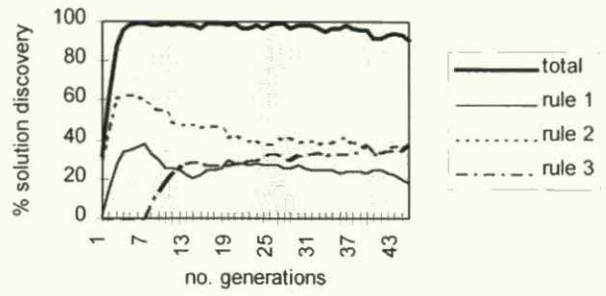
The datasets were generated containing known regularities, or rules. The system was tested by incorporating a performance algorithm to monitor the percentage of rules in the rule base which corresponded to one of the known rules. This, in effect, measures the ability of the system to learn by recognising regularities and adapting the rule base accordingly.

For each trial the rule base was populated with 250 randomly generated rules. Rules were generated by setting each allele (except those in the outcome part) to either # or a value from the set {0-5}, each with a probability of 0.5. Each trial was run for 45 generations (one generation is equivalent to every rule being matched once with every cell in the input data) over rasters of size 8*8. Payoff, g and $1/cr$ were set to 1000 , 2 and 0.5, respectively, after initial experimentation. All tax values were set to zero to limit the number of variables for testing.

4. Results

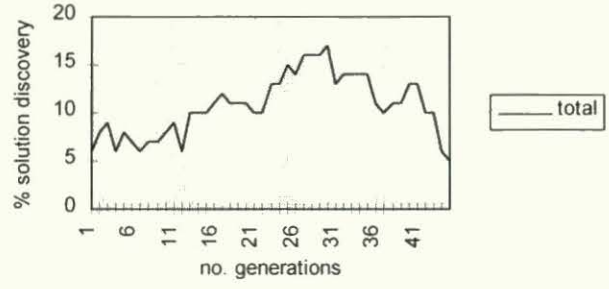
Figure 2a show the results from a simple trial where the input data contained only three randomly placed rules and no random noise ('non-rules') with $1/mt$ set to 0.015. The upper series (total) represents the percentage of the rule base that corresponds to any one of the three known rules while the other three series (rule 1-3) correspond to the discovery of individual rules, as a percentage of the entire rule base. As can be seen from the total the system rapidly recognises, learns and populates itself with know rules. Even though the total reaches close to 100% after six generations, the population is initially dominated by rule 1 and, to a lesser extent, rule 2. The system only 'discovers' rule 3 after 8 generations. Thereafter rule 3 becomes more widely represented, at the expense of rules 1 and 2. After twenty-five generations the system appears to reach a stage of equilibrium with only minor changes in the composition of the rule base.

Figure 2a.



It can be seen that the randomly populated rule base (zero generations) contains solution matches. Although the theoretical probability of random generation of any one of the known rules is low, the high percentage of wildcards in the initial population (50%) increases the probability greatly (the performance algorithm includes rules with up to three wildcard matches). The graph shows that the initial rule base did not contain any rules matching rule 3. The system required time to search the solution space and adjust itself to present a successful system decision. A second trial was run with the same system parameters as used in Figure 2a., the only difference being that the input data was randomly generated to contain no regularities (a regularity being defined as more than one occurrence of a particular combination of conditions and outcome) and, specifically, no instances of any of the three known rules. Six of the sixty-four unique combinations (a unique combination can be thought of as a single-occurrence rule) were fed into the performance algorithm for system testing. The system was able to learn only two of the six combinations. An interesting phenomena was discovered when the system was run a third time, this time only the performance algorithm was altered by feeding it with the three rules used in Figure 2a, the results of which are shown in Figure 2b. The system appears to be discovering regularities not existent in, in fact specifically excluded from, the data.

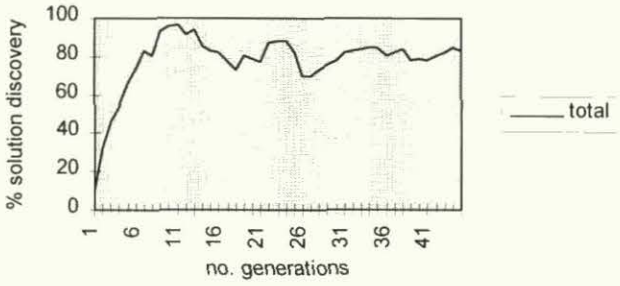
Figure 2b.



The explanation for this phenomenon is that many different, but similar, patterns in the data can be matched by one rule from the rule base. For example, the three input data strings {0,1,5,3,4}, {1,1,3,3,4}, {5,1,2,3,4} can all be matched by the rule {#,1,#,3,4}. This suggests that results produced by the learning algorithm must be interpreted by care. The percentage of the rule base that corresponds to any of the know rules can include multiple counting due to the possibility of multiple matching illustrated above (Σ individual rule recognition % > 100%) . However, none of the trials presented show evidence of this.

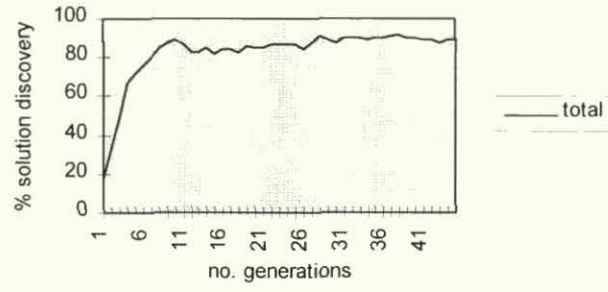
The role of mutation in rule discovery is demonstrated in Figure 2c. This plot shows total system performance on a trial with $1/mt$ set to 0.25. In all other respects the trial is identical to that presented in Figure 2a. As can be seen the higher probability of mutation has led to slower, less stable, learning and an equilibrium level lower than that achieved in Figure 2a. As the mutation rate increases, the more conservative processes of rules replication and crossover become less prevalent. Rule replication serves to strengthen the rule base while crossover produces new rules through minor changes, both based on the historical performance of the rule base. Mutation does not use the accumulated knowledge of the rule base and the effects of this more radical search strategy can be seen. Even when the system reaches equilibrium mutation ensures the continued insertion of random genetic material into the rule base.

Figure 2c.



A more realistic learning environment is presented in Figure 2d. In this case the input data contains both known rules and randomly generated noise.

Figure 2d.



The system is able to learn the known rules, though less rapidly than when 'clean' data is used. Solution discovery reaches a maximum value of around 90%, compared with the figure of 97% achieved for clean data. This is expected as the rule base must retain non-solution rules to be able to respond to frequent, random noise.

5. Discussion

The development and implementation of a successful classifier system prototype has been presented. The system works well with small datasets of restricted complexity where the existence of known relationships in the data allows an evaluation of performance. It has demonstrated a capability to recognise patterns in both clean and noisy data and a sensitivity

to variations in the mutation rate. Although a comprehensive testing of other system parameters, specifically taxes, was outwith the scope of this study, work by Holland (1986) and Stewart (1987) documents their role in the learning process.

A limitation of the system was discovered during the trial performed on the data containing sixty-four unique rules. Although the rule base was exposed to each combination forty-five times (the number of generations over which the trial was run) the system exhibited a low rule discovery rate. It is not clear if the limitation is due to the large number of rules to be learned and held, concurrently, by the rule base or to the infrequent exposure of the rule base to each rule. With respect to the latter, if the system were applied to real world data it is unlikely that a phenomenon that occurs so infrequently in such isolated groups (of one) would be considered an important regularity in the data.
It is suggested that the system provides, at the very least, the basis for a technique that can be used for the exploratory analysis of 'real world' spatial data. There are a large number of potential applications for the system. As the arbitrary naming of the test datasets implies, the system could be used to look for evidence regarding the causes of soil erosion, marginal cropping and other unsustainable practices. An immediate advantage of the technique over statistical methods is the ability to incorporate both interval and categorical data. This would allow analysis of the conditions giving rise to low income and poverty by incorporating biophysical (soil, slope etc.) and socio-economic (land tenure, farming practices etc.) data. Analysis should aim to incorporate as many factors as possible, with little *a priori* concern for which data might be relevant. If the system is run over a sufficiently large number of generations it will identify particular sets of conditions which give rise to particular outcomes with irrelevant data simply represented by wildcards in the rule base.

Where *a priori* theories or hypotheses about the nature of regularities in the data are held, they can be incorporated as rules, seeded in the rule base. Over a number of generations, the proliferation or extinction of these rules should help determine their validity.

A final note of caution is required. The classifier system presented here should only ever be considered an aid to directed, insightful exploratory analysis of spatial data. Any results produced by a classifier system should be subject to the highest scrutiny. It is quite conceivable that a large, rich spatial database will contain spurious regularities of little relevance. Results should be used as part of an ongoing process of analysis, aiding in the construction of hypotheses and guiding future research.

6. Future

Future work should aim to determine the scalability of such a system. It is essential to test the system on real world data, magnitudes of size and complexity greater than the very simple data used in this study. Scalability should be considered in terms of both system performance and the ability to handle large numbers of concurrent rules with, potentially, infrequent payoff. This might include measures to siphon rules out of the system once they have reached a particular threshold (in terms of dominance of the rule base etc.), as trials indicated that apparently strong rules can become extinct if not subject to frequent strengthening. Appendix A contains additional information on the architecture and performance of the system.

An issue requiring attention is the incorporation of spatial relations in the search process. At present the system can only handle those relations which are explicitly defined and recorded in a dataset e.g. distance to a particular road. It is unsatisfactory to include only spatial information that is considered *a priori* to have relevance. Dibble (1994) proposes the use of

Dutton's tesselation data structure as a means of incorporating spatial relations. An area of investigation would be the use of object-oriented data modelling to incorporate the full semantics of spatial relations that might be relevant in exploratory data analysis.

References

Aspinall R., 1994, 'Exploratory spatial analysis in GIS: generating geographical hypotheses from spatial data', in Worboys M. (ed), Innovations in GIS 1, Taylor Francis.

Barto A.G., 1985, Learning by statistical cooperation of self-interested neuron-like computing elements, COINS Technical Report 85-11, Amherst: University of Massachusetts.

Dibble C., 1994, 'Beyond Data: handling spatial and analytical context with genetics based machine learning ', Proceedings SDH '94, Edinburgh, 1041-1060.

Dibble C., Densham P.J., 1993, 'Generating interesting alternatives in GIS and SDSS using genetic algorithms', Proceedings GIS/LIS '93, 180-189.

Frey P.W.,1986, 'A bit-mapped classifier', Byte, 11(12), 161-173.

Goldberg D.E, 1989, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley.

Holland J.H, 1975, Adaptation in Natural and Artificial Systems, MIT Press.

Holland J.H., 1986, Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems, in R.S. Michalski, J.G. Campbell & T.M. Mitchell (eds), Machine Learning: An artificial intelligence approach (Vol. 2), Morgan Kaufmann (Los Altos, CA).

Holland J.H, 1992, 'Genetic algorithms', Scientific American, July, 66-72.

Koza J.R., 1992, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press.

Mitchell T.M., 1982, 'Generalisation as search', Artificial Intelligence, 18, 203-226.

Openshaw S., 1993, 'Some suggestions concerning the development of artificial intelligence tools for spatial modelling and analysis in GIS', in Fischer M.M, Nijkamp P. (eds), Geographic Information Systems, Spatial Modelling and Policy Evaluation, Springer-Verlag.

Openshaw S., 1994, 'A concepts-rich approach to spatial analysis, theory generation, and scientific discovery in GIS using massively parallel computing', in Worboys M. (ed), Innovations in GIS 1, Taylor Francis.

Schrodt P.A., 1986, 'Predicting International Events', Byte, 11(12), 177-192.

Senior M.L., 1979, 'From gravity modelling to entropy maximising: a pedagogic guide', Progress in Human Geography, 3(2), 176-210.

Wilson S. W., 1987, 'Classifier Systems and the Animat Problem', Machine Learning, 2, 199-229

Appendix A

System Implementation

The system was coded in C using the Borland C++ compiler (V 4.5) under Windows 3.1. Although written for a 16-bit OS it is anticipated that few changes would be required for porting to a different OS. A fully documented code listing is included in Appendix B.

The system was designed to explore the applicability of the classifier system rather than achieve high performance levels. With limited attention paid to program optimisation processing speed can be slow. For a trial with a rule base of size 300 and five 8*8 input raster processing times on an RM PC-466 (486) and RM PC-575 (75 Mhz Pentium) are 34 and 21 minutes, respectively. For trials with real, significantly larger, datasets some form of optimisation would clearly be required. The use of a more restricted alphabet to represent rules and data input may help in this respect.

Datasets must be stored as ASCII rasters of equal dimensions. The system can, theoretically, handle any number of datasets, of any size, however, limitations are imposed by the memory capacity of the hardware running the system.

The algorithm includes a bounds checking function. The rules are coded as integer arrays and this function makes period checks for rules with strengths approaching INT_MAX to avoid system crashes. Although unnecessary for most trials it avoids problems where large datasets are processed over many generations.

Appendix B

```
/*****************************************************************************
****          CLASSIFIER SYSTEM PROTOTYPE                                  ****
****          (machine learning of relationships in spatial data)            ****
****                                                                       ****
****              file: csp.cpp (code)                                     ****
****              Euan Crawford, July 1997                                 ****
*****************************************************************************/

#include "csp.h" // libraries and algorithm parameters

/* Function declarations - descriptions included with code after main() *
FILE * diagf(FILE *, int);
void filetoarray(int [][NROW][NCOL]);
void randfill(int **, FILE *);
int reinforce(int [][NROW][NCOL], int, int, int **, FILE *);
void ga(int **, FILE *);
void bounds(int **);
void stats(int **, FILE *, int);
void sort(int **);
void prn_data(int [][NROW][NCOL], FILE *);
void prn_pop(int **, FILE *);
void header(FILE *);
void mem(int **);

void main() {

int h,i,j,**pop;
int dstore[NUMDSETS][NROW][NCOL];
time_t st,ft;   // for calculation of processing time
FILE *df;

 pop=new int *[POPSIZE];     // allocate memory for rule base
 for(i=0;i<POPSIZE;++i)
  pop[i]=new int[W];

 randomize();
 st=time(NULL);
 filetoarray(dstore);        // environmental (input) data to array
 df=diagf(df,1);
 #if V
  prn_data(dstore,df);
 #endif
 randfill(pop,df);

 printf("\n\n\n\t\tprocessing:      ");
 for(h=1;h<=MAXGEN;++h){     // generation counter
    for(i=0;i<=AOIR-1;++i){
        for(j=0;j<=AOIC-1;++j){   // raster AOI controls
//            bounds(pop);
            while(!reinforce(dstore,i,j,pop,df)); // reinforce() while [C] set empty
            ga(pop,df);
            #if V
                if(TSOL) stats(pop,df,h);
            #endif
        }
    }
```

```
    printf("\b\b\b\b\b%5d%",(h*100)/MAXGEN);  // percentage processed counter
    if(TSOL) stats(pop,df,h);
}

 ft=time(NULL); ft-=st;
 printf("\n\n"); if(TSOL) stats(pop,df,h);
 sort(pop);
 fprintf(df,"\n\nfinished after %d iterations (%d seconds)\n\n",h,ft);
 if(TSOL) stats(pop,df,h);
 prn_pop(pop,df); df=diagf(df,0);

 mem(pop);                          // deallocate memory

}

//-----------------------------------------FUNCTIONS-------------------------------------------------------

//---------------------------------LEARNING ROUTINES-----------------------------------------------------
int reinforce(int dstore[][NROW][NCOL], int r, int c, int **pop, FILE *df) {
/* Function has two roles:   i)matches rules in rule base with input strings
                             ii)strengthens rules according to matches */

 int i,j,maxbid=INT_MIN,mset=0,aset=0,ecnt=0,lsysdec=INT_MIN,sysdec=0,taxa,taxna,taxsd;
 double dt=0,tmp;

/*    rule matching - counts input string-classifier matches   */
 for(i=0;i<=POPSIZE-1;++i){
    pop[i][G]=0; pop[i][C]=0;     // reset rule match scores to 0
        for(j=0;j<=NUMDSETS-2;++j)    // count condition matches only
        if(pop[i][j]==dstore[j][r][c] || pop[i][j]==MAXVALUE){
            ++pop[i][C];
        if(pop[i][j]==MAXVALUE)
            ++pop[i][G];              // count # matches
        }
 }

/* strengthen rules */
 for(i=0;i<=POPSIZE-1;++i)
    if(pop[i][C]==NUMDSETS-1)       // determine size of [C]
        ++mset;

 if(mset>0){                    // if [C] not empty
    for(i=1;i<=NUMDSETS-1;++i){      // locates system decision (full matches)
        for(j=0;j<=POPSIZE-1;++j)       // with lowest no. #
            if(pop[j][C]==NUMDSETS-1 && pop[j][G]<=i && pop[j][S]>maxbid ){
                lsysdec=j; maxbid=pop[j][S];
            }
        if(lsysdec>INT_MIN)
            break;
    }

 if(pop[lsysdec][A]==dstore[NUMDSETS-1][r][c])
    sysdec=1;                   // test if system decision correct

 for(i=0;i<=POPSIZE-1;++i)        // determine size of [D]
    if(pop[i][C]==NUMDSETS-1 && pop[i][A]==pop[lsysdec][A]){
        ++aset; ecnt+=(NUMDSETS-1)-pop[i][G];                  // count non-general (#) matches
    }
```

```
#if V  // progress report
   fprintf(df,"mset:%d  aset:%d  lsysdec:%d  sysdec:%d  ecnt%d\n\n",
           mset,aset,lsysdec,sysdec,ecnt);
   fprintf(df,"compdatsol(): flag classifier and input string matches\n");
   prn_pop(pop,df);
#endif

// calculate generality function for [D}, excluding full # matches
 for(i=0;i<=POPSIZE-1;++i)
   if(pop[i][C]==NUMDSETS-1 && pop[i][A]==pop[lsysdec][A] && pop[i][G]<NUMDSETS-1){
      tmp=NUMDSETS-1;tmp=(tmp-pop[i][G])/tmp; tmp=pow(tmp,GEN); dt+=tmp;
   }

 for(i=0;i<=POPSIZE-1;++i) {
   if(pop[i][C]==NUMDSETS-1){
      if(pop[i][A]==pop[lsysdec][A] && pop[i][G]<NUMDSETS-1){
         taxa=TAXD/10;                        // select members of [D]
         pop[i][S]=pop[i][S]-((pop[i][S]*taxa)/10);  // tax [D]
         if(sysdec==1){                       // if sys. decision correct
            tmp=NUMDSETS-1; tmp=(tmp-pop[i][G])/tmp;  tmp=pow(tmp,GEN);
            pop[i][S]+=(tmp/dt)*TPAYOFF;             // payoff to [D]
         }
         else if(sysdec==0) {                 // if sys. decision wrong
            taxsd=TAXSD/10;
            pop[i][S]=pop[i][S]-((pop[i][S]*taxsd)/10); // tax [D]
         }
      }//[D]
      else {                                  // select members of NOT[D]
         taxna=TAXND/10;                                 // tax NOT[D]
         pop[i][S]=pop[i][S]-((pop[i][S]*taxna)/10);
      }//NOT[D]
   }//NUMDSETS-1
 }//i

 #if V
   fprintf(df,"reinforce(): classifiers taxed and rewarded\n");
   prn_pop(pop,df);
   sort(pop); fprintf(df,"reinforce(): classifiers taxed, rewarded  and sorted\n");
   prn_pop(pop,df);
 #endif

}//mset==1
else if(mset==0){                          // if [D] empty
   sort(pop);
   #if V
      fprintf(df,"[M] empty - pop sorted and rule created\n\n");
   #endif
   for(i=0;i<=A;++i){                      // create new rule
      if(i<A){
         if(random(2)==1)
            pop[POPSIZE-1][i]=MAXVALUE;           // insert random #s
         else
            pop[POPSIZE-1][i]=dstore[i][r][c];    // else copy input string
      }
      else
         pop[POPSIZE-1][i]=random(MAXVALUE);      // rule outcome=random value
   }
```

```
    pop[POPSIZE-1][S]=(pop[0][S]+pop[POPSIZE-2][S])/2; // rule strength set to median of rulebase

    #if D
        fprintf(df,"rule created + inserted\n");
        prn_pop(pop,df);
    #endif

}//mset=0;

return mset; // return value indicating if [C] empty or not
}


void ga(int **pop, FILE *df)  {          /* genetic algorithm */
 int i,j,tmp[W],spt=0,cprob=0,ps1,ps2,rv=1;

 sort(pop);

 if(random(CROSSPROB)==CROSSPROB/2){
    cprob=1;              // invoke crossover with probability 1/CROSSPROB
    while(!(spt=random(NUMDSETS)));   // define split point (0 < spt < NUMDSETS)
 }
 #if V
    fprintf(df,"crossover switch:%d  crossover point:%d\n\n",cprob,spt);
 #endif

 for(i=0;i<W;++i){             // strongest rules replace weakest
    pop[POPSIZE-1][i]=pop[0][i];     //copy strongest over weakest
    if(cprob==1){
        pop[POPSIZE-2][i]=pop[1][i];     //copy 2nd strongest over 2nd weakest
            if(i>=spt){                  // crossover at spt
                tmp[i]=pop[POPSIZE-2][i];
                pop[POPSIZE-2][i]=pop[POPSIZE-1][i];
                pop[POPSIZE-1][i]=tmp[i];
            }
    }

 /* invoke mutation with probability 1/MUTPROB equally over chromosome */
 if(i==A) rv=0;                  // # not permitted for action mutation
 if(random(MUTPROB)==MUTPROB/2){
    pop[POPSIZE-1][i]=random(MAXVALUE+rv);
    if(cprob==1 && random(MUTPROB)==MUTPROB/2)
        pop[POPSIZE-2][i]=random(MAXVALUE+rv);
 }
 // alter rule strengths
 if(cprob==1){                       // if crossover invoked
    ps1=pop[0][S]/3; ps2=pop[1][S]/3;
    pop[0][S]=pop[0][S]-ps1; pop[1][S]=pop[1][S]-ps2;
    pop[POPSIZE-1][S]=pop[POPSIZE-2][S]=ps1+ps2;
 } // parent strengths reduced by 1/3 and shared equally between offspring
 else {                              // crossover not invoked
    pop[0][S]=pop[POPSIZE-1][S]=pop[0][S]/2;
 } // parent strength halved and given to child strength

 #if V
    fprintf(df,"ga(): new rules bred and mutated\n"); prn_pop(pop,df);
 #endif
```

```
}

//---------------------------------BACKGROUND ROUTINES-----------------------------------------------
void stats(int **pop, FILE *df, int h){
/* calculate learning performance statistics (when rules in data known) */
 int i,j,k,tp=0,maxv,max=0,sol[TSOL][NUMDSETS+3]={{0,1,2,3,4,0,0,0},
                                                  {1,2,3,4,5,0,0,0},{2,3,4,5,0,0,0,0}};

 for(i=0;i<=POPSIZE-1;++i){  // count no. rules from solution set in rulebase
    for(k=0;k<=TSOL-1;++k){
    sol[k][NUMDSETS]=0; maxv=0;
    for(j=0;j<=A;++j){
        if(pop[i][j]==sol[k][j]) // count individual taxa/action matches
            ++sol[k][NUMDSETS];
        else if(pop[i][j]==MAXVALUE){
            ++sol[k][NUMDSETS];
            ++maxv;
        }
    }
    if(sol[k][NUMDSETS]==NUMDSETS && maxv<NUMDSETS-1){
        ++sol[k][NUMDSETS+1];   // count full (non #) matches
        k=TSOL;                 // skip to next classifier if match found
    }                   // note: may miscount non-unique rules (esp #)
 }
}
for(k=0;k<=TSOL-1;++k)      // calc % correct classifiers
    if(sol[k][NUMDSETS+1]>0){
        sol[k][NUMDSETS+2]=(sol[k][NUMDSETS+1]*100)/POPSIZE;
        tp+=sol[k][NUMDSETS+2];
        if(sol[k][NUMDSETS+2]>max)
            max=sol[k][NUMDSETS+2];
 }

 fpri tf(df,"soln[%d]\t ",h); // output stats to file/screen
 for(i=0;i<=TSOL-1;++i)
    fprintf(df,"%d:%3d  ",i+1,sol[i][NUMDSETS+2]);
 fprintf(df,"total:%3d",tp);
}


void sort(int **pop) { /* bubble sort rule base */
 int i,j,k,tmp[W];

 for(j=0;j<=POPSIZE-2;++j)
  for(i=0;i<=POPSIZE-2;++i)     // sort by (in order): strength, no. condition matches, generality
    if( pop[i+1][S]>pop[i][S] ||
        (pop[i+1][S]==pop[i][S] && pop[i+1][C]>pop[i][C]) ||
        (pop[i+1][S]==pop[i][S] && pop[i+1][C]==pop[i][C] && pop[i+1][G]<pop[i][G]) ){
        for(k=0;k<=S;++k)
        tmp[k]=pop[i][k];
        for(k=0;k<=S;++k)
            pop[i][k]=pop[i+1][k];
        for(k=0;k<=S;++k)
            pop[i+1][k]=tmp[k];
 }
}
```

```
void bounds(int **pop){
/* checks for rule strengths approaching INT_MAX when tax calcs performed  */
 int i,f=0,tmp=max(TAXSD,max(TAXD,TAXND));

 if(tmp>0) tmp=INT_MAX/(tmp/10); // for integer % tax calculations
 for(i=0;i<=POPSIZE-1;++i)
    if(pop[i][S]>tmp) {
        f=1;
        break;                  // flag high strength
    }

 if(f)
    for(i=0;i<=POPSIZE-1;++i)     // divide all rule strengths by 100
        if(pop[i][S]>100)
            pop[i][S]=pop[i][S]/100;
}


//----------------------------------------OUTPUT TO FILE-----------------------------------------------
void prn_pop(int **pop, FILE *df){
/* print rule base to report/diagnostics file */
 int i,j;

 for(i=0;i<=POPSIZE-1;++i) {
    fprintf(df,"%3d: ",I);
    for(j=0;j<=W-1;++j)
        if(j==S)
            fprintf(df,"%6d",pop[i][j]);
        else if(pop[i][j]==6)
            fprintf(df,"%3c ",'#');
        else
            fprintf(df,"%3d ",pop[i][j]);
        putc('\n',df);
  }
  putc('\n',df);
}


void prn_data(int dstore[][NROW][NCOL], FILE *df){
/* print array holding raster data to report/diagnostic file */
 int h,i,j;

 fprintf(df,"prn_data(): rasters read into arrays\n");
 for(h=0;h<=NUMDSETS-1;++h){
    fprintf(df,"dstore[%d]\n",h);
        for(i=0;i<=NROW-1;++i){
            for(j=0;j<=NCOL-1;++j)
                fprintf(df,"%d ",dstore[h][i][j]);
            putc('\n',df);
  }
 }
 putc('\n',df);
}


void header(FILE *df){
/* output learning parameters to report/diagnostics file */
 fprintf(df,"\t\t\t**********Report**********\n\n");
```

```
 fprintf(df,"No. datasets:%d  Max. value:%d  Pop size:%d  No. gen:%d\n",
  NUMDSETS,MAXVALUE,POPSIZE,MAXGEN);
 fprintf(df,"Crossover prob:%d  Mutation prob:%d\n",CROSSPROB,MUTPROB);
 fprintf(df,"TPAYOFF:%d  tax[D]:%d  tax[D](sd=0):%d  tax NOT[D]:%d\n\n",
  TPAYOFF,TAXD,TAXSD,TAXND);
}
//--------------------------------INITIALISATION ROUTINES-----------------------------------------
void randfill(int **pop, FILE *df){
/* randomly populate rule base */
 int i,j,tmp;

 for(i=0;i<POPSIZE;++i)
    for(j=0;j<W;++j){
        if(j==G || j==C)
            pop[i][j]=0;
        else if(j==S)
            pop[i][j]=100;
        else if(j==A)
            pop[i][j]=random(MAXVALUE); // '#' not permitted as action value
        else {
            tmp=random(MAXVALUE+1)%RG;
            if(tmp==0)
                pop[i][j]=MAXVALUE;
            else
             pop[i][j]=random(MAXVALUE);
        }
    }

/*seed rules can be inserted here for diagnostics and hypothesis testing
 pop[1][0]=0; pop[1][1]=1; pop[1][2]=2; pop[1][3]=3; pop[1][4]=4; pop[1][5]=5;  */

 #if V // prints randomly filled rule base to diagnostics file
    sort(pop); fprintf(df,"randfill(): random population generated\n");
    prn_pop(pop,df);
 #endif
}


void filetoarray(int dstore[][NROW][NCOL]){
/* reads data from ascii rasters and stores in array*/
 int i,j,k;
 char fnames[NUMDSETS][20]={"r-soil.dat","r-aspect3.dat",
                            "r-slope.dat","r-dist.dat",
                            "r-lcover.dat"};   // ASCII raster filenames
FILE *ipf[NUMDSETS];

 for(i=0;i<=NUMDSETS-1;++i)          // open files
    ipf[i]=fopen(fnames[i],"r");

 for(i=0;i<=NROW-1;++i)              // read data into array
    for(j=0;j<=NCOL-1;++j)
        for(k=0;k<=NUMDSETS-1;++k)
            fscanf(ipf[k],"%d",&dstore[k][i][j]);
        for(i=0;i<=NUMDSETS-1;++i)          // close files
            fclose(ipf[i]);
}
```

```
void mem(int **pop) {
/* deallocate rulebase memory */
 for(int i=0;i<POPSIZE;++i)
    delete[] pop[i];
    delete[] pop;
}

FILE * diagf(FILE *df, int sw){
/* open/close diagnostics/report file */
 if(sw==1){
    if((df=fopen("diagf.dat","w"))==NULL){
        printf("file error\n");
        exit(1);
    }
    header(df);
 }
 else
  fclose(df);

 return df;
}
```

```
/*****************************************************************************
****          CLASSIFIER SYSTEM PROTOTYPE                                  ****
****          (machine learning of relationships in spatial data)            ****
****                                                                       ****
****             file: csp.h (header)                                      ****
****             Euan Crawford, July 1997                                  ****
*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <limits.h>
#include <time.h>

/* input data parameters */
#define NUMDSETS 5          // number of datasets/rasters
#define W NUMDSETS+3        //
#define A NUMDSETS-1        //
#define G NUMDSETS          // alter (W-S) only via NUMDSETS
#define C NUMDSETS+1        //
#define S NUMDSETS+2        //
#define NCOL 8              // no. columns in rasters
#define NROW 8              // no. rows in rasters
#define MAXVALUE 6          /* 0-MAXVALUE = range of permissible values in input data,
                               where wildcard (#) = MAXVALUE */

/* learning parameters */
#define POPSIZE 250         // size of rulebase
#define MAXGEN 75           // no. generations
#define AOIC 8              // define segment of raster to be included in learning -
#define AOIR 8              // no. of columns/rows from upper left
#define TSOL 3              // no. of solution sets included for performance testing
#define RG 2                    // 1/RG =probability of alleles as # in randomly populated rulebase
#define TPAYOFF 1000        // payoff to [D]
#define GEN 2               // generality payoff control
#define TAXD 0              // % tax on [D]
#define TAXSD 0             // % tax on [D] (sd=0)
#define TAXND 0             // % tax on NOT[D}
#define CROSSPROB 2         // 1/CROSSPROB = GA crossover probability (even nos only)
#define MUTPROB 60          // 1/MUTPROB = allele mutation probability (even nos only)

/* performance reporting */
#define V 0                 // verbose report on/off (small trials only)
```