Coursework Proposal

Genetic Algorithm

I have decided to use a genetic algorithm with a hyperrectangle classifier to help solve the problem. A genetic algorithm is useful because of its ability to work with varying sized problem sets. The algorithm makes discrete changes to the population without making any assumptions, and given our solution should make no assumptions, this algorithm seemed most appropriate to me. The algorithm can remain unchanged despite the addition of extra columns/dimensions.

Knowledge Representation

As stated above, I will use a hyperrectangle as my knowledge representation. The classifier will be defined as two points diagonally across from each other that defines the boundary of the hyperrectangle. It will then be a trivial operation to check whether a value falls within these two values/the boundary of the hyperrectangle.
When computing the fitness function we can use a simple selection process that checks whether the instance is within the bounds of the rectangle - greater than one point and less than the other point, where both points are as stated above (diagonally across) as to say that the instance falls within the bounds of the hyperrectangle. This is computationally cheap, which is ideal for any computing task.

Implementation

I will implement the genetic algorithm using the JGAP framework. This framework has excellent classes that will allow me to model genotypes, chromosomes, genes, and run selection and mutation processes.

It will also allow me to produce a simple model for my hyperrectangle. I will store the classifier as an array of lower and upper bounds, the size of this array will correlate to the size of the problem set (number of attributes), doubled as explained below. This will allow me to use the InstanceSet and Instance classes to easily calculate when an instance falls within my hyperrectangle by checking if the real value for the instance is within the classifiers boundary.
I will use a tournament selection process to attain the best candidates, where the size of the tournament will be set to five.

I will create my hyperrectangle as a two dimensional array, with the size [n][2].
This will allow me to store n upper and lower bounds, where n is the number of attributes (dimensions). The bounds will be stored as a DoubleGene.
When the Classifier class calls my own implemented classifier class, it will check each dimension *n* for an instance that is outside the *n-hyperrectangle* bounds.

Ant Colony Optimisation (Swarm Intelligence)

The second method I will use to solve the problem is an ant colony optimisation algorithm.
The idea behind this is to mimic the way ants build paths to food by creating weighted edges (trails) on a graph. The ant releases pheromones (corresponding to the weight of the edge) to let other ants know of a successful trail to food. More ants use this trail, leading to more pheromones (height weight).

This algorithm has the advantage of avoiding premature convergence, having a better chance of producing the optimal outcome. This is because of the idea behind pheromone evaporation; if the pheromone took longer to evaporate or didn't at all, then the solution space would be constrained as the trail (edge) would remain used (highly weighted) even if it was a bad solution. So a good rate of evaporation lets us find a nice balance between exploration (finding new solutions) and exploitation (finding the optimal solution).

Knowledge Representation

I will use the Max-Min Ant System (MMAS) proposed by Stutzle and Hoos[1] which has the following characteristics, leading to my reasoning as to why I chose it.

After each iteration, only the best ant is allowed to add pheromone to its trail. This allows for a better exploitation (to find the optimal solution faster). Next, the range of possible pheromone trails is limited to the interval $[T_{max}, T_{min}]$ to avoid premature convergence (where a suboptimal solution is found early). Lastly, to allow for high exploration early (and to find a balance between that and exploitation) the initial pheromone value for each trail is set to T_{max}.

Implementation

The algorithm consists of constructing rules, reducing these rules, and updating pheromones. The Ant-Miner does a flexible (varying size) sequential covering of the training set to discover a list of classification rules - a feature that led me to choose this method, as we are given varying sets of files.

When the test file is looked at, the rules developed by the test case will be used to classify. The Classifier class will then call my rule class to determine the instance class (non-programming class type). With the rules created in training, we can prune the rules with negative matches – which can be displayed as error rate and percentage covered and not covered as the framework does currently.

References

[1] https://svn-d1.mpi-inf.mpg.de/AG1/MultiCoreLab/papers/StuetzleHoos00%20-%20MMAS.pdf