Using Genetic Algorithms to Inductively Reason with Cases in the Legal Domain

Anandeep S. Pannu
Intelligent Systems Program
University of Pittsburgh

Abstract

Reasoning pragmatically (rather than using theories of jurisprudence) from cases has been established as a viable model of legal reasoning. Cases are recognized as encapsulating specific knowledge tied to a context. The reasoning done with cases lead to solutions that are tied to situations. To generalize these solutions, the use of machine learning techniques is often necessary. Induction from cases, seen as 'examples' with a 'classification' in a domain, is one of these techniques. A Genetic Algorithm based approach to inductively learn features of a case in a legal casebase that leads to specified classifications is described in this paper. This knowledge can then be used to reason about cases that are 'interesting' by virtue of their features and classification, and to predict classifications of other cases. This method is contrasted with other well known techniques in machine learning. The claim is made that *prototype exemplars* can be generated efficiently and that operational information from any domain (ie. cases) can be used to guide the generation, using variants of this technique.

1 Introduction

Case-based reasoning (CBR) is accepted as a viable legal modeling technique in AI. The law is an ill-structured domain and the case-based approach attempts to make up for the lack of an authoritative strong model by modeling weaker methods for drawing and supporting legal conclusions (Ashley 1992). CBR avoids the problems inherent in a heuristic rule-based approach because it confines itself to *symbolic* comparison, inference and justification rather than trying to find the 'right' answer (as rule-based experts systems purport to). Cases are 'operationalizations of domains' (Kolodner 1993) that are anchored to specific circumstances or situations and any comparison, inference and justification in CBR is also anchored to these circumstances.

Every CBR program assumes the existence of a partial domain theory for analyzing problems by comparing cases (Ashley 1992). The 'operationalization' of the domain is interpreted through the (implicit or explicit) partial domain theory and representation implemented by a CBR program. The usefulness of a CBR program is determined by the user's assessment of the reliability of the advice given by the program. This is easy to do in a case-based reasoning program, because the explanations in a CBR are anchored in specific circumstances allowing the user to draw connections between 'abstract explanatory concepts whose definitions he may not know and concrete facts with which he may be more familiar' (Ashley 1992).

HYPO (Ashley 1990) is a CBR program which uses a partial domain theory of precedents based on *Factors*, but presents the results to the user (assumed to be familiar with the law) in a manner such that the user need not understand the underlying theory. HYPO justifies a conclusion about a problem by drawing an analogy to a similar past case and arguing that the problem should be decided the same way. Employing examples to focus a listener on important features, like HYPO does, helps the listener to assess the reasonableness of advice.

While learning from cases, even when cases are considered as just clusters of features, this property of transparency should be preserved. Cases have a set of facts representing the 'circumstances' of the case and have a 'result' which is the *classification* that the circumstances of the case lead to. In this paper the problem of classifying cases using the circumstances of the cases is addressed and a technique for inductive learning from cases using Genetic Algorithms is described. Cases are seen as examples tied to particular situations and the examples provide evidence about features that lead to particular classifications. The technique is used to generalize from the examples and extract features that lead to particular classifications. This technique used in a program called GAINC (*G*enetic *A*lgorithm for *IN*duction from *C*ases) is a generative rather than deductive or statistical technique. Information is kept in the same form as a case in the case-base, ie. in uninterpreted symbolic form rather than numeric or

(say) decision tree form. The generation of a 'perfect' prototypical exemplar for a particular classification is the goal of the program. The fitness function used by the GA is based on a general notion of 'distance' and 'similarity' which is usable across case-bases and domains.

In the next section *factors* are described and the problem of inducing knowledge about attributes of factors is introduced. Two well known machine learning techniques and their weaknesses *vis a vis* the problem at hand are analyzed in the third section. The GA technique is introduced in the section following.The performance of the GA is analyzed in the fifth section. Inferences that can be made from the learned knowledge are explained and the technique's generalizability to other legal reasoning systems is explored. In subsequent sections comparisons are made with statistical learning programs in the law, and the advantages of the GA based technique are enumerated.

2 Factors and the Case-Base

HYPO (Ashley 1990) is a program that exemplifies the adversarial case-based reasoning approach. HYPO's task is to analyze legal disputes involving trade secret laws. In such a dispute the plaintiff claims that the defendant has gained an unfair competitive advantage by gaining access to the plaintiff's confidential product-related information. In every case there are some *factors* that favor the defendant and some factors that favor the plaintiff. Factors are a kind of expert knowledge of the commonly observed collections of fact that strengthen or weaken the plaintiff's arguments. Generally, factors that strengthen the plaintiff's arguments are favorable to the plaintiff and factors that weaken the plaintiff's arguments favor the defendant.

In HYPO, *dimensions* represent a factor and also whether a factor favors the legal claim of a plaintiff (Ashley 1990). This knowledge is provided by the domain expert (in this case the creator of HYPO, Kevin Ashley) to the HYPO system.

CATO (Aleven & Ashley 1992) is a pedagogical program which is based on HYPO and uses it's underlying partial domain theory to teach students how to reason with cases. It has a larger case-base than HYPO and identifies more factors. CATO's knowledge sources consist of a Case Knowledge Base of forty six legal cases indexed by twenty dimensions in the trade secrets domain of the law.

As mentioned above, dimensions represent the factors that experts judge are important collections of facts for winning legal claims in the trade secrets domain of law. HYPO uses cases from its CKB as precedents to provide a basis for how to resolve competing factors in a *new* case. Typically a trade secret dispute is likely to involve a collection of competing factors, some favoring the plaintiff and some favoring the defendant. According to (Ashley 1992), 'Legal experts are likely to agree that these factors are important in deciding the case but they often disagree on how important each factor is or how the conflict should be resolved.' Note that legal experts agree on what the factors are and which side they favor but disagree on the relative importance of these factors and their effect on the outcome.

How did this agreement on which factor favors the plaintiff and which favors the defendant come about? In this paper is it is claimed that collection of factors that favor the plaintiff or the defendant can be inductively generated from our Case Knowledge Base, specifically from knowledge of the factors present in each case and its outcome. As is hinted at by the disagreement about the effect of factors, there is an assumption that factors are not linearly separable - ie. the effect of a factor on winning/losing a case cannot be calculated in isolation from other factors present in the case. Since each factor has an effect dependent on what other factors are present in a case; non-linear interactions make it meaningless to consider each factor in isolation, resulting in a need for search in the space of factors.

The objective of this paper is to address the question of whether knowledge can be induced from a set of cases represented as a set of features with a classification. The relative merits of using dimensions versus factual predicates will not be addressed. The fact that factors are constructs is not material since the technique does not have the knowledge that factors are constructs and considers each one of them as just a feature in a feature vector (explained in the next section). However, the use of easily understood constructs such as factors makes it easier to judge the merits of the technique for humans.

3 What Machine Learning techniques could be applied?

The task therefore, is to find out what factors are pro-defendant and which factors are pro-plaintiff from cases which have information about outcomes and the factors leading to an outcome, in a domain where there is no authoritative deductive model. Given the circumstances, induction - defined as 'all inferential processes that expand knowledge in the face of uncertainty' (Holland *et al* 1986), is a promising avenue to explore.

A look at some of the major paradigms in symbolic machine learning that could be candidates for learning the effect of factors given the information that is available, would now be in order. First we look at what information is available in the cases.

To simplify matters a case from CATO's case-base is represented as a vector of attribute values (each representing a factor) and a indication of an outcome. Each attribute position can have one of two binary values ie. either a '1' indicating that the factor represented by the attribute is present or a '0' indicating that it is absent in that particular case. The first attribute in every case represents the outcome and is also a binary value. A '0' represents the fact that the plaintiff won the case and a '1' indicates that the defendant won the case.

Since there are twenty dimensions representing factors in cases, the vector is twenty one bits long (including the outcome bit). The factor that each attribute represents and a case from the database are

Bit	Represents
1	Outcome (Win Plaintiff= 0 / Defendant =1)
2	Disclosure-In-Negotiations
3	Info-Reverse-Engineerable
4	Info-Independently-Generated
5	Bribe-Employee
6	Employee-Sole-Developer
7	Identical-Products
8	Agreed-Not-To-Disclose
9	Agreement-Not-Specific
10	Security-Measures
11	Secrets-Disclosed-Outsiders
12	Info-Known-To-Competitors
13	No-Security-Measures
14	Unique-Product
15	Brought-Tools
16	Competitive-Advantage
17	Knew-Info-Confidential
18	Vertical-Knowledge
19	Outsider-Disclosures-Restricted
20	Restricted-Materials-Used
21	Invasive-Techniques

Table 1: Bit positions and Representations

Speedry	111100000000000000000
Case Name	Speedry
Won by	Defendant
Factors Present	Disclosure-In-Negotiations Info-Reverse-Engineerable Info-Independently-Generated

Table 2: A case in CATO's case base

shown in Table 1 and Table 2 respectively.

The problem could now be viewed in two ways; which correspond to two major sub-divisions of inductive (or empirical) machine learning (Dietterich & Shavlik 1990).

One way to look at the problem is to say that the concepts 'factors-favoring-plaintiff' and 'factors-favoring-defendant' are to be learned. Supervised concept learning involves inducing concept descriptions to be learned from a set of positive and negative examples of the target concepts. Examples are represented as points in an n-dimensional feature space which is defined *a priori* and for which all the legal values of the features are known (DeJong & Spears 1991). In the representation used above, each case is an example. An outcome in which a plaintiff wins is a positive example for the concept 'factors-favoring-plaintiff' and a negative example for the concept 'factors-favoring-defendant'. Similarly an outcome in which a defendant wins is a negative example for the concept 'factors-favoring-plaintiff' and a positive example for 'factors-favoring-defendant'.

Once the problem is formulated this way there are many standard learning algorithms (and representations) to chose from; for instance decision trees (Quinlan 1986) used by the ID3 algorithms. Unfortunately the domain considered is not so amenable to these well known techniques. The reasons for these are enumerated below.

1. The problem considered resembles the *multiplexor problem*. Multiplexor problems fall into the general area of trying to induce a description of a k-input boolean function from input/output examples (DeJong & Spears 1991). Here the case is expressed as a boolean string and the output is a binary digit (outcome). Because no single individual input 'line' is useful in distinguishing class membership, information-theoretic approaches like ID3 have a particularly hard time inducing decision trees for multiplexor problems.

2. There is uncertainty (in the form of noise) involved in the examples in the database. An example with the same factors present may be won by the plaintiff in one case and won by the defendant in another case. Decision trees have no easy way to represent this uncertainty and cannot 'smooth out' the noise resulting from contradictory examples.

3. Even if there were a decision tree that was generated from the examples, the resulting decision tree is a predictor of whether a case can be won / lost when a test example is presented to it. That is, the decision tree representation has to be post-processed in some way to get the representation of *all* factors that predict a plaintiff/defendant win (it that were possible at all).

Unsupervised inductive learning does not depend on explicit classification of examples by an expert. Instead the learning systems try to 'look for regularities' in the examples presented to them. Most unsupervised learning programs search for regularities that take the form of "clusters" of values (examples are usually presented as vectors of values) and so is called *clustering*. An example in these methods is represented as a vector of feature values plus a classification label. In the formulation of a case, the vectors would be formed by representing the factors as the vector of features and the outcome of the case as the classification label.

All exemplar-based methods learn by storing examples as points in feature space (Cost & Salzberg 1993). The feature vector of a case in our problem can be clustered into one of the clusters "plaintiff-wins" or "defendant-wins". A model can then be developed that specifies which factors predict that the plaintiff wins and which factors predict that the defendant wins. Unfortunately this technique also does not fit the problem domain well, for the following reasons.

1. The classifications of the examples are "case won by plaintiff" or "case won by defendant" and not "factors favoring plaintiff" or "factors favoring defendant". The information being sought is an aggregate property of the examples. This

means even if a classification model of the domain is constructed with this technique, post processing would be required to extract the information needed.

2. In the domain being considered it is quite possible to have 'contradictory' categorization ie. a feature vector with exactly the same features (factors) and opposite categorizations (outcomes). It is not clear how the exemplar-based learning can handle inconsistency in the instances presented to it.

3. The exemplar being sought is the 'perfect exemplar'. That means that it must contain *all* the factors that result in a plaintiff/defendant win and *not* contain the factors that weaken the case for the plaintiff/defendant. This exemplar (or one near perfect) is not likely to be found in any case base. So an exemplar-based learning system would not be able to learn this (except maybe when unusual combination of examples are presented to a *best-example* learning system (Kibler & Aha 1987))

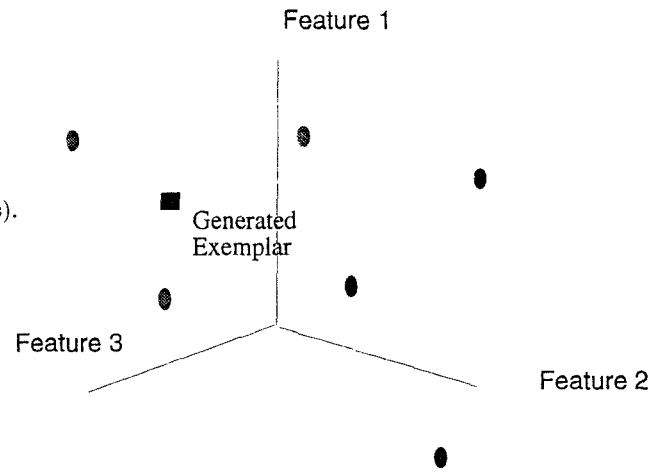
4 A new technique for inducing knowledge

An alternative formulation of the problem that is a combination of strengths from supervised learning and clustering, and uses a *generative mechanism* to generate exemplars (instead of just classifying examples presented to it) is introduced in this section.

Assume as before that examples are presented as vectors of features with an outcome and an exemplar (or *prototype*) of a certain classification needs to be found. This exemplar should be 'closest' to all the exemplars that provide support for the outcome that the prototype represents and 'furthest' from the exemplars that do not have the same classification. In addition the prototype should be an 'average' of all the examples that have the outcome that the prototype represents. To understand what this means in feature space, look at Figure 1. Here a vector of three features is considered (excluding the outcome bit - which is not considered as a part of feature space but is represented separately).

The shaded circles represent the examples (eg. cases from the Case Knowledge Base) that have the same outcome as the prototype we are trying to find (eg. cases won by the defendant). The dark circles are the examples with the opposite outcome (ie. cases won by the plaintiff). The prototype itself is represented as a filled in rectangle. As can be seen from the figure, the prototype is close to the examples that provide support to it and furthest from examples that do not. In addition it is in some way an "average" of the cases that support it.

Advantage is also taken of the *single representation trick* (Barr *et al* 1981) used in machine learning, in which instances and instance generalizations are expressed in the same language. This trick is used in reverse, here the concepts induced are expressed in the language of cases, rather than expressing instances in the generalization language.



Exemplar Generation in Feature Space

Figure 1:

Obviously there is a need to define the concept of *distance* in feature space so that concepts such as 'closest' and 'furthest' can have some meaning. Use is made of the concept of distance used in instance-based learning programs (Cost & Salzberg 1993). When features are numeric, normalized Euclidean distance can be used to compare examples. However when feature values have symbolic, unordered values (eg. the letters of the alphabet, which have no natural inter-letter "distance") resorting to simpler metrics (like counting the number of matches) is required. The distance metric used is an important part of our technique and will be described in a later section.

The problem of finding the prototype according to a measure of distance is exponential in complexity, since an exhaustive search would possibly have to consider the powerset of the features time the number of cases in the case-base.

What method can be used that generates this prototype making full use of the information already available in the cases and which does not have a worst case exponential runtime? The answer lies in Genetic Algorithms (GAs). GAs are search algorithms based on the mechanics of natural selection and natural genetics (Goldberg 1989). GAs maintain a population of members, usually called "genotypes" and classically represented as a binary string, which can be mutated and combined according to a measure of their worth or "fitness" as measured by a *task dependent evaluation function*.

GAs perform *adaptive* search by evolution of strings. There is a result by Holland (1992) that states 'despite the processing of only n structures each generation of a GA, there is useful processing of something of the order of n^3 structures in parallel with no

special bookkeeping or memory other than the population itself'(*intrinsic parallelism*). Another result from Holland (Goldberg 1989) states that an exponential number of trials will be assigned to the observed best building blocks.

This is the crux of the technique - avoiding exponential complexity but getting near-optimal results by generation of feature vectors in feature space. The other important parts are the coding and the fitness function.

GAs require the natural parameter set of the optimization or search problem to be coded as a finite-length string over some finite alphabet. So it is assumed that a potential solution may be represented as a set of parameters. The parameters (known as *genes*) are joined together to form a string of values (known as a *chromosomes*).

This scheme fits well with the representation of cases as vectors of features. The cases can now be 'strings' over the alphabet '0' and '1'. The potential solution can also be represented in the same form.

The coding described above is for the cases in the Case Knowledge Base. The question to be considered next is of how the evolving population of strings will be coded and what the population represents. Since it is not known if a crossover of an arbitrary string with other strings will result in a string representing win for the plaintiff or a win for the defendant, the least significant bit in the string (representing the outcome) cannot be set. This problem is overcome by formulating the problem as either evaluating a best fit instance for a defendant win or a plaintiff win each time the GA is run. If an exemplar is required that induces the factors for a plaintiff win, the least significant bit is uniformly set to 0 ; and if an exemplar with factors responsible for a defendant win is needed the least significant bit is set to 1 uniformly *(only in the evolving population).*

This means we have two 'populations' - one for the actual cases represented as strings and the other, the evolving population, which is used in the search for a superfit instance (see Figure 2).

The fitness evaluation function chosen is based on the distance measure used by the *k-nearest neighbor algorithm*(KNN). The KNN algorithm classifies feature vectors according to the vectors k closest neighbors. Each feature is taken to be a dimension in the search space. The samples (cases/feature vectors/strings) from the case-base are placed in this dimensional space and are labeled with their *classification* (eg. whether the case was won by the defendant or the plaintiff). An unknown sample can be placed in the space based on its feature values and then can be classified based on its k *nearest neighbors* where k is set to some integer value.

The distance between feature vectors is defined as $d_{ij} = \{\sum_{a=1}^{n}(X_{ia} - X_{ja})^2\}^{\frac{1}{2}}$ ((Kelly & Davis 1991)

Where i and j are the i^{th} and j^{th} vectors of features being compared. X_{ia} is the value of the a^{th} attribute for the i^{th} vector X_{ja} is the value of the a^{th} attribute for the j^{th} vector. This gives a Euclidean measure of distance, even though the features are symbolic. Since only binary strings are being considered, the distance between each feature is either 0 or 1.

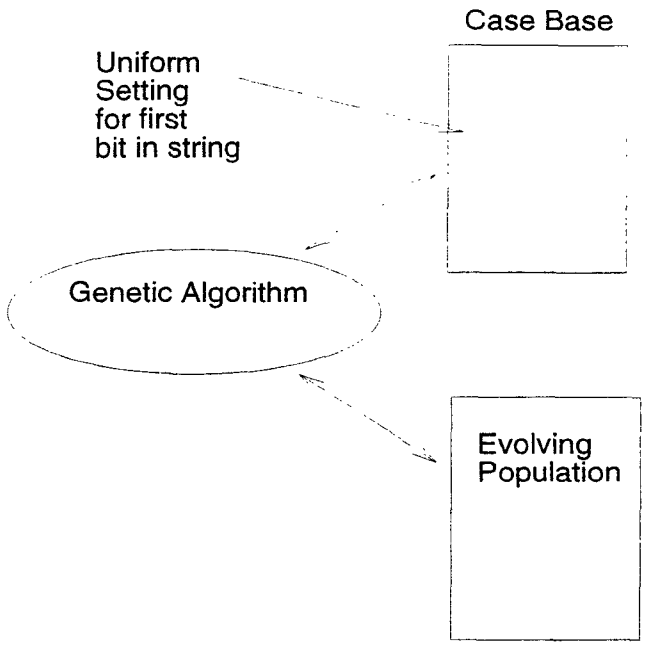


Figure 2:

The k-nearest neighbor algorithm just classifies new examples that are presented to it. It has to be adapted to generate the *perfect prototypical exemplar* that is being sought. The objective of the GA is to find an optimum point in feature space so as

- to *minimize* the distance for all such feature vectors (or strings as feature vectors are referred to in GAs) with known classifications (ie. the strings representing the cases) for which the classifications agree with our generated strings classification (pro-defendant or pro-plaintiff).
- to *maximize* the distance for all such feature vectors with known classifications for which the classifications *do not* agree with our generated strings classification.

The following fitness function accomplishes this

$$Fitness = \frac{\sum^{x}\{\sum_{a=1}^{s}(X_{xa}-X_{ga})^2\}^{\frac{1}{2}} - \sum^{y}\{\sum_{a=1}^{s}(X_{ya}-X_{ga})^2\}^{\frac{1}{2}}}{\sum_{i=1}^{n}\{\sum_{a=1}^{s}(X_{ia}-X_{ga})^2\}^{\frac{1}{2}}} \quad (1)$$

Where

x ranges over the strings that *do not* have the same classification as the generated string. In case the string is generated to identify the pro-defendant factors; x would contain the strings that have pro-plaintiff results.

y is the complement of the strings (in the universe of strings from the case-base) comprising the set that x ranges over. That is the strings with the same "outcome" or classification as the generated string.

n is the total number of cases in the case-base.

X_{ga} is the a^{th} attribute of the GA generated feature vector.

X_{ya} and X_{xa} are the a^{th} attributes of the feature vectors in the set over which y and x respectively range. X_{ia}, in the denominator, is the a^{th} attribute of the entire set of strings (i ranges from 1 to n.

s is the size of the string or chromosome - the attributes in each X_i are $a_{i1} \ldots, a_{is}$. In the strings from CATO's case-base the s is twenty (excluding the classification bit).

The essence of the fitness function is that it maximizes the sum of the distances from the generated string of the non-compatible classifications and minimizes the sum of the distances from the generated string of the compatible classifications. The first fraction is the ratio of the sum of the distances of known classifications that are not the same as the classification being searched for, to the total distance from *all* strings . Since this is additive it will be maximized. The second fraction is the ratio of the sum of the distances of known classifications that share the same classification as the GA generated string to the total distance from all strings and will be minimized due the negative sign in front of it. The sum of the two ratios is optimized to get the prototypical exemplar.

The above fitness function however takes into account only the *differences* between the strings. There needs to be a measure which factors in the *similarity* between features - this time *maximizing* the similarity between the generated string cases with similar classifications and *minimizing* the similarity between cases with dissimilar classifications.

The new measure is incorporated into the fitness by adding

$$\frac{\sum^{sim}\sum_{a=1}^{s} F_{sim}(a) - \sum^{dsim}\sum_{a=1}^{s} F_{dsim}(a)}{\sum_{i=1}^{n}\sum_{a=1}^{s} F(ia)}$$

to the fitness calculated by (1) above.

The function $F(a)$ returns 1 if the a^{th} attribute of a feature vector is the same as the a^{th} attribute of the generated string and returns 0 otherwise. s is the size of the feature vector and n is the total number of feature vectors. *sim* constitutes the set of feature vectors that have the same classification as the string being generated and *dsim* is the set of feature vectors with different classifications. As in (1) above the first sum is to be is maximized and the second minimized (due to the presence of a preceding negative sign).

5 Experimental setup and results

The algorithms and associated utility programs were written in *C*. The algorithm for the GA was the one given by Goldberg (Goldberg 1989) (sometimes called the SGA - Simple Genetic Algorithm). Some of the GA specific techniques that were used are described in this section.

The initial population is the strings that make up the 'zero-eth' generation. These strings are used as the 'seed' to start the evolution process. It is necessary that the strings cover the entire search space uniformly and are not biased towards a region in the space. In the experiment the population is generated by first taking in the population size desired as a parameter. Then *each bit* of each of the strings is generated randomly. This increases the probability that the strings so generated represent every region

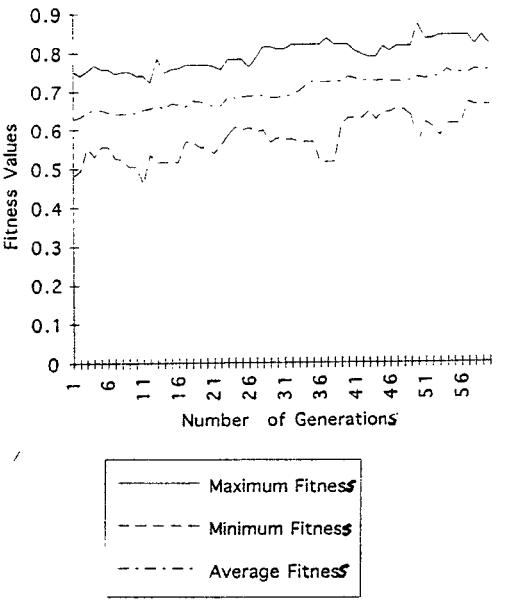


Figure 3: GA Run for Plaintiff Exemplar

in the search space ie. a fair initial distribution. The least significant bit of each of the strings is then set to the value that represents the side for which the string is to be generated.

The GA used has the usual reproduction, crossover and mutation operators. The parameters for the probabilities of crossover and mutation were set at the rate found to work for most GA applications (Goldberg 1989). They were varied to get the best results but proved robust to in the face of variation. Single point crossover was used and stochastic remainder without replacement was the selection procedure (Goldberg 1989). Since the algorithm described by Goldberg handled only objective functions that returned a positive value; and the objective function that was used could result in negative values the algorithm had to be modified to incorporate a *translation function*. This function made sure that the fitness value returned by the objective function was properly translated so that no negative fitness values were ever assigned to a string. The scaling function was called once in every 'generation' for each member of the population.

Figures 3 and 4 show graphs of the results of running GAINC with maximum fitness, average fitness and minimum fitness for each generation. The factors identified by GAINC as pro-plaintiff are given in Table 3 and the factors identified by GAINC as pro-defendant are given in Table 4.

The relative importance of any single feature is assumed to be the same as that of any other in the string, but in the domain of law this is not always true. Some features are more influential than others in deciding the outcome of the case. How can the features relative importance be induced from the case-base using the technique outlined above?

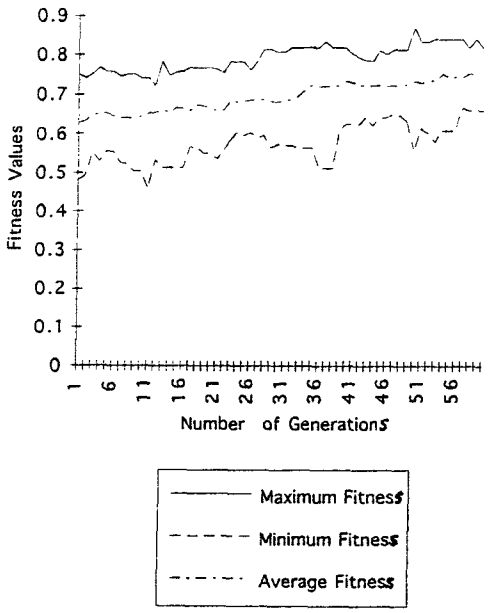


Figure 4: GA Run for Defendant Exemplar

Global max	0.868980
Fittest Chrom	D11111001011101001001
1	Disclosure-In-Negotiations (d)
2	Info-Reverse-Engineerable (d)
3	Info-Independently-Generated (d)
4	Bribe-Employee (p)
5	Employee-Sole-Developer (d)
8	Agreement-Not-Specific (d)
10	Secrets-Disclosed-Outsiders (d)
11	Info-Known-To-Competitors (d)
12	No-Security-Measures (d)
14	Brought-Tools (p)
17	Vertical-Knowledge (d)
20	Invasive-Techniques (p)

Table 3: The factors identified as pro-defendant

Global max	1.125633
Fittest Chrom	P00000110100011110111
6	Identical-Products (p)
7	Agreed-Not-To-Disclose (p)
9	Security-Measures (p)
13	Unique-Product (p)
14	Brought-Tools (p)
15	Competitive-Advantage (p)
16	Knew-Info-Confidential (p)
18	Outsider-Disclosures-Restricted (p)
19	Restricted-Materials-Used (p)
20	Invasive-Techniques (p)

Table 4: The factors identified as pro-plaintiff

Global max	1.056203
Fittest Chrom	8.16 7.18 9.53 4.71 8.35 6.27 8.90 5.06 9.61 8.08 9.10 5.49 0.00 0.08 0.04 0.00 5.06 0.00 0.00 2.43
1	Disclosure-In-Negotiations (d) weight 8.16
2	Info-Reverse-Engineerable (d) weight 7.18
3	Info-Independently-Generated (d) weight 9.53
5	Employee-Sole-Developer (d) weight 8.35
6	Identical-Products (p) weight 6.27
7	Agreed-Not-To-Disclose (p) weight 8.90
8	Agreement-Not-Specific (d) weight 5.06
10	Secrets-Disclosed-Outsiders (d) weight 8.08
11	Info-Known-To-Competitors (d) weight 9.10
12	No-Security-Measures (d) weight 5.49
17	Vertical-Knowledge (d) weight 5.06

Table 5: The weighted factors identified as pro-defendant

Global max	1.418576
Fittest Chrom	6.75 1.84 0.78 0.67 5.29 0.00 0.04 2.47 4.55 1.14 0.71 0.08 9.22 5.73 8.00 9.96 0.16 8.31 9.57 3.76
1	Disclosure-In-Negotiations (d) weight 6.75
5	Employee-Sole-Developer (d) weight 5.29
13	Unique-Product (p) weight 9.22
14	Brought-Tools (p)weight 5.73
15	Competitive-Advantage (p) weight 8.00
16	Knew-Info-Confidential (p) weight 9.96
18	Outsider-Disclosures-Restricted (p) weight 8.31
19	Restricted-Materials-Used (p) weight 9.57

Table 6: The weighted factors identified as pro-plaintiff

The KNN algorithm (more specifically the weighted KNN algorithm) provides clues in this regard. Consider the distance measurement to be multiplied by a weight w_{ga}, which has been set to the value 1.0 in the technique considered so far. Therefore the modified fitness function would be

$$\frac{\sum_{sim}\sum_{a=1}^{s} w_{ga}*F_{sim}(a) - \sum_{dsim}\sum_{a=1}^{s} w_{ga}*F_{dsim}(a)}{\sum_{i=1}^{n}\sum F(ia)} + \frac{\sum^{x}\{\sum_{a=1}^{s} w_{ga}*(X_{xa}-X_{ga})^2\}^{\frac{1}{2}} - \sum^{y}\{\sum_{a=1}^{s} w_{ga}*(X_{ya}-X_{ga})^2\}^{\frac{1}{2}}}{\sum_{i=1}^{n}\{\sum_{a=1}^{s} w_{ga}*(X_{ia}-X_{ga})^2\}^{\frac{1}{2}}}$$

Setting w_{ga} ,where w_{ga} represents the weight of the a^{th} feature in the GA generated string, to 1.0 for every factor means that each factor has the same relative importance. However the weight can be made fine grained by assigning more than one bit to every factor in the string. The values chosen for weighted GAINC are eight bits for each factor, allowing the factor to take 256 values normalized between 0 and 10. Arbitrarily it was decided that a factor which had a weight value below 5.0 was 'absent' in the string whereas a factor with a weight value above 5.0 was 'present' wherever the decision needed to be made. The 256 values were *Gray coded* to avoid *Hamming Cliffs* (Whitley & Shaner 1988). The results of run-

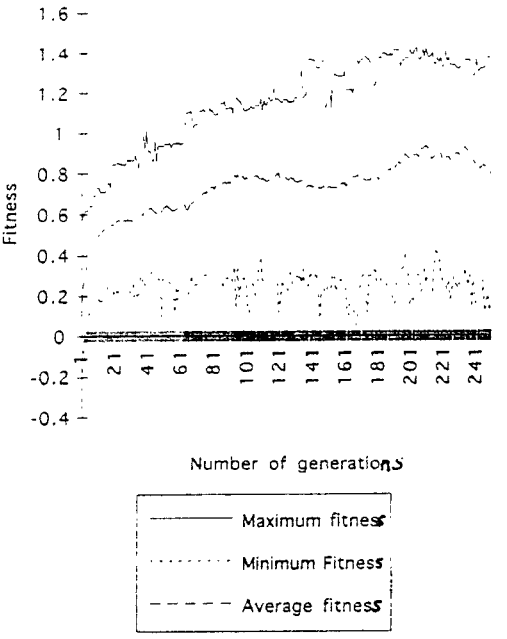


Figure 5: Weighted GA Run for Plaintiff Exemplar

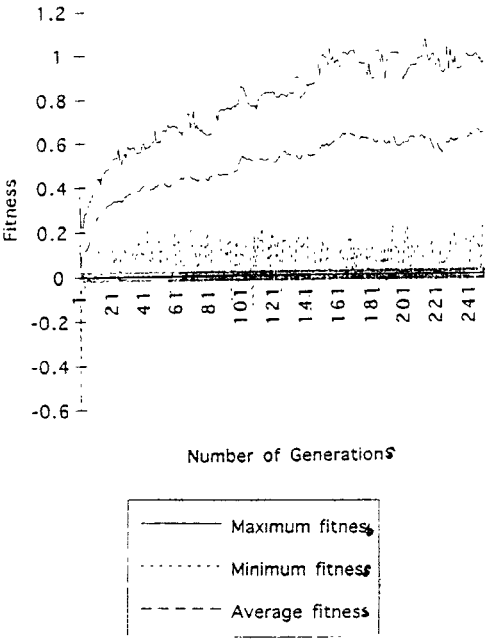


Figure 6: Weighted GA Run for Defendant Exemplar

ning weighted GAINC are given in Table 5 and Table 6. Figures 5 and 6 show graphs of the results of running weighted GAINC with maximum fitness, average fitness and minimum fitness for each generation.

The results of running the GA have been encouraging. In Table 4, the GA correctly classified 10 of the 11 pro-plaintiff factors and did not mis-classify any of them. In Table 3 *all* the pro-defendant factors were identified - but the GA mis-classified three pro-plaintiff factors as pro-defendant. (A (d) following a factor name means that the factor is judged to be pro-defendant and a (p) following a factor indicates that the factor is judged to be pro-plaintiff by a human expert.) This means that the GA has a 91% success rate at correctly classifying pro-plaintiff factors and 83% success rate at correctly classifying pro-defendant factors. The misclassified factors could be seen to be dependent on the particular case-base (which has a relatively small number of cases) since for example, the factor *Bribe-Employee (p)* which is misclassified as a pro-defendant factor occurs in 3 cases which are decided for the defendant but only 2 which are classified as pro-plaintiff (assuming of course the other factors contributing to the result have minimal effect). This is reflected in Table 5 (for the weighted GA run) which shows that the weight for the factor *Bribe-Employee (p)* is just below the threshold to be considered to be pro-defendant. (It is 4.71 whereas the threshold is 5.0).

The results of the weighted GA are a mixed bag - it mis-classifies only 2 pro-plaintiff factors and classifies all pro-defendant factors correctly. Only 6 of 11 pro-plaintiff factors are identified as pro-plaintiff however. The results for the weighted and non-weighted GA are reasonably consistent with each other. The analysis of a factors weight as generated by the weighted exemplar would provide the user an idea of the relative importance of a factor in deciding the classifications.

6 Using the induced knowledge

What is the utility of the prototype exemplars of classifications, generated by the GA? The answer lies in using the similarity and distance measures that have been used to induce the knowledge. In doing so the pragmatic view is taken that a legal model can be 'based upon a simplified model of legal reasoning (Popple 1994).' The GA has a simple and transparent knowledge representation structure in which the user is able to see the solution 'evolving' and assess the reliability of the advice generated. These advantages are crucial to machine based legal reasoning and it is not necessary to build complex systems to achieve useful results.

Consider Figure 7, in which two dimensional feature space is being considered. The prototypical exemplars for the two classifications A and B have been generated. Looking at Case A it is seen that it is 'closer' (in terms of the inter string distance that has been defined) to Case B that has a *different* classification than it is to it's own exemplar. What does this mean? An analysis of the situation provides some answers. The factors present in Case A which caused it to be 'pushed' across the boundary to a different classification may give useful information about the domain. This was implemented in GAINC by measuring the distance of each case from its exemplar and then comparing these cases to cases which had opposite classifications but a distance (measured using

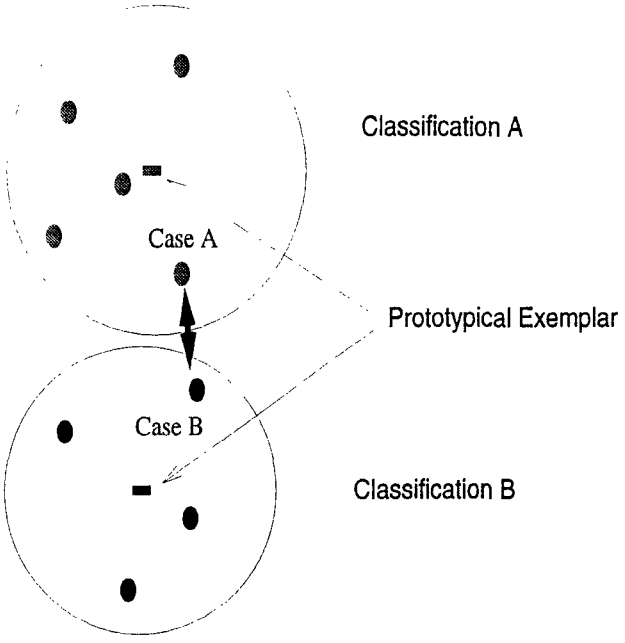


Figure 7:

the Euclidean distance metric as used in the nearest neighbor algorithms) less than that from the exemplar. The factors in each case were then flagged for inspection by the user.

Though GAINC is not developed as a predictive program, unclassified (ie. unknown outcome) cases could be classified using the fitness function developed for GAINC and distance measurement from the exemplars. The fitness function (either the weighted or unweighted fitness function) could be used to compare the fitness of the unclassified case by

1. setting the bit to indicate that the case was won by the defendant (as for the GA string in a defendant run) and evaluating the fitness.
2. Repeating step 1 above but this time setting the bit to indicate that the case was won by the plaintiff.
3. Assigning the classification that has the higher score to the case.

The distance of the case from the exemplar of different classifications can also be measured and the case can then be assigned the classification of the nearest exemplar.

Note that none of the methods should be considered to be predictive, but GAINC is set up so that the user can explore various hypotheses and see how they are supported by the case-base.

7 Comparison with other pragmatic approaches

SHYSTER (Popple 1993) is a pragmatic legal expert system which deals with case law and is designed to be used by lawyers. SHYSTER represents the state of the art in statistical legal reasoning and is used a basis for comparison. Like GAINC, SHYSTER uses a representational structure that is simple, since according to Popple (1993) this simplifies knowledge acquisition. Like GAINC, SHYSTER also represents it's cases as 'points in space, the dimensionality of which is the number of attributes.' It is, like GAINC, generalizable to more than one legal domain.

SHYSTER, however, is designed to make *predictions* about the likely result of a case, where the prediction is based upon previously decided cases. GAINC is more of an exploratory tool, it is best used to check the user's hypotheses about the cases in the case-base presented to it; to see what knowledge can be induced from features and classifications dependent on those features. The knowledge so induced is an aggregate property of the case-base and can be used to analyze particular cases. SHYSTER compares an *instant case* to all preceding cases that apply in it's model of legal reasoning. GAINC uses information available in the case-base, whereas SHYSTER depends on the user for input about properties of its cases. Once such input is based on the concept of an *ideal point.* An ideal point represents the best case for a given result. In constructing the the perfect prototypical exemplar GAINC would automatically see the emergence of these ideal points. SHYSTER's weights it's attributes (which is controversial see Ashley (1990)) using a statistical measure. Weighted GAINC generates weights that are supported by the case-base, to measure relative importance of factors. GAINC uses Ashley & Rissland's (1988) advice to postpone weighting as long as possible - that is to the point when it is evaluating the fitness during the run. SHYSTER calculates attribute weights *before* reasoning begins, since each attribute is calculated by taking the inverse of its variance across the case-base. Since the attributes in SHYSTER are decided by the user it has to make sure that the attributes are stochastically independent (so that, in the worst case, the user does not define the same attributes to SHYSTER under a different name). In GAINC, it is expected that there will be stochastic dependence between attributes and the distance measures are sensitive to this (Punch *et al* 1993). The most significant difference is that the calculations which SHYSTER uses to reach it conclusions and construct its legal arguments are not visible to the user. In GAINC the string evolution can be seen and its fitness assessed for reasonableness by the user, since the string is in easily interpretable symbolic form. There exist techniques and packages for graphical visualization (Clark 1994) and explanation (Louis *et al* 1993) of GAs that would make this task even easier.

8 Conclusions

A new technique for induction from cases was described in this paper. This technique is of use in pragmatic legal expert systems (Popple 1993) and a few inferences that could be made from knowledge induced by the technique were discussed. This technique is generally applicable, not only in the legal domain but also in any domain where the primitives

can be represented as feature vectors with classifications. For instance if there exists a case-base that contains factors pertaining to awards made in damage claims with the classifications being *damages-less-than-20000, damages-between-20000-and-50000, damages-between-50000-and-100000* and so on; GAINC could be run with the factors represented as feature vectors. The factors (one of which might be whether major injuries were sustained) leading to particular classifications could be induced. The distance and similarity measures could be changed from the ones used so far and as long as they retained the ability to quantify differences and similarities - the techniques would be useful. The advantages *vis a vis* runtime complexity and perspicuity was compared with a state of the art statistical legal expert system. Further work in this area should concentrate on evaluation of classifications and the generation of 'interesting' cases ; and on integrating GA visualization and explanation techniques into GAINC.

References

Aleven, V & Ashley, K. D. (1992). Automated Generation of Examples for a Tutorial in Case-Based Argumentation. In *Proceedings of the Second International Conference on Intelligent Tutoring System.* Montreal.

Ashley, K. D. (1990). *Modeling legal argument : Reasoning with Cases and Hypotheticals.* MIT Press, Cambridge, MA.

Ashley, K. D. (1992). Case-Based Reasoning and its implications for Legal Expert Systems. *Artificial Intelligence and Law:1* pp. 113-208. Kluwer Academic Publishers, Netherlands.

Ashley, K. D. & Rissland, E. L. (1988). Waiting on Weighting: A symbolic least commitment approach. *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88).* St. Paul, MN.

Barr, A. , Feigenbaum, E. A. & Cohen, P. (1981). *The Handbook of Artificial Intelligence.* Reading, MA: Addison-Wesley.

Clark, G. (1994). Visualisation of Genetic Algorithms. *Technical Report EPCC-SS94-07.* Edinburgh Parallel Computing Center. University of Edinburgh. Edinburgh, UK.

Cost, S. & Salzberg, S. (1993). A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features. *Machine Learning,* 10, pp. 57-78.

DeJong, K & Spears, W. M. (1991). Learning Concept Classification Rules using Genetic Algorithm. *Proceedings, 12th International Joint Conference on Artificial Intelligence,* 651-656. Sydney, Australia. IJCAI.

Dietterich, T. G. & Shavlik, J. (1990). *Readings in Machine Learning.* Morgan Kaufmann, San Mateo, CA.

Goldberg, D. (1989). *Genetic Algorithms in search, optimization and machine learning.* Addison-Wesley, Reading, MA.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems.* MIT Press, Cambridge, MA.

Holland, J. H. , Holyoak, K. J ., Nisbett, R. E. & Thagard, P. R. (1986). *Induction : Processes of Inference, Learning, and Discovery.* MIT Press, Cambridge, MA.

Kelly, J. & Davis, L. 1991. Hybridizing the Genetic Algorithm and the K Nearest Neighbors Classification Algorithm. *Proceedings of the 4th International Conference on Genetic Algorithms and their Applications.* Morgan Kaufmann:CA.

Kibler, D. & Aha, D. W. (1987). Learning Representative examples of concepts : An initial case study. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 24-30). Morgan Kaufmann, Irvine, CA.

Louis, S. , McGraw, G. & Wyckoff, R. 0. (1993). Case-based reasoning assisted explanation of genetic algorithm results. *Journal of Experimental and Theoretical Artificial Intelligence.* 5. pp21-37. Taylor & Francis.

Popple, J. D. (1993). *SHYSTER : A Pragmatic Legal Expert System.* Ph. D thesis , Australian National University, Faculty of Engineering and Information Technology, Dept of Computer Science.

Punch, W. F. , Goodman, E. D. , Pei, M. , Chia-Shun, L. , Hovland, P. & Enbody, R. 1993. Further Research on Feature Selection and Classification Using Genetic Algorithms. *Proceedings of the International Conference on Genetic Algorithms.* Champaign:IL.

Whitley, D. & Shaner, D. (1988). Representation Issues in Genetic Algorithms. *Technical Report CS-88-102.* Colorado State University, Fort Collins, CO.