

Scaling up Inductive Logic Programming: An Evolutionary Wrapper Approach

PHILIP G.K. REISER AND PATRICIA J. RIDDLE

{philip,pat}@cs.auckland.ac.nz

Department of Computer Science, University of Auckland, New Zealand.

Received ??; Revised ??

Editors: ??

Abstract.

Inductive logic programming (ILP) algorithms are classification algorithms that construct classifiers represented as logic programs. ILP algorithms have a number of attractive features, notably the ability to make use of declarative background (user-supplied) knowledge. However, ILP algorithms deal poorly with large data sets ($> 10^4$ examples) and their widespread use of the greedy set-covering algorithm renders them susceptible to local maxima in the space of logic programs.

This paper presents a novel approach to address these problems based on combining the local search properties of an inductive logic programming algorithm with the global search properties of an evolutionary algorithm. The proposed algorithm may be viewed as an evolutionary wrapper around a population of ILP algorithms.

The evolutionary wrapper approach is evaluated on two domains. The chess-endgame (KRK) problem is an artificial domain that is a widely used benchmark in inductive logic programming, and Part-of-Speech Tagging is a real-world problem from the field of Natural Language Processing. In the latter domain, data originates from excerpts of the Wall Street Journal. Results indicate that significant improvements in predictive accuracy can be achieved over a conventional ILP approach when data is plentiful and noisy.

Keywords: evolutionary algorithms, inductive logic programming, sampling, machine learning

1. Background

This work addresses the classification problem in machine learning. That is, given training examples of the form $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$ for some unknown function $y = f(\mathbf{x})$, where the \mathbf{x}_i values are vectors of the form $\langle x_{i,1}, x_{i,2}, \dots, x_{i,n} \rangle$, and y values are drawn from a discrete set of classes $\{1, \dots, K\}$; find a definition of function f such that the y value for any \mathbf{x}_i from the same distribution is accurately predicted [10]. Two approaches

to constructing classifiers are inductive logic programming and evolutionary algorithms.

1.1. Evolutionary Approaches to Classification

Evolutionary algorithms [2] are domain-independent search algorithms inspired by principles of population genetics. Using very simple mechanisms, evolutionary algorithms exhibit complex search behaviour that has been harnessed to solve some difficult problems, e.g. [7, 16].

Evolutionary algorithms (EA) have in the past successfully been used for the classification problem. GABIL [9] and REGAL [13] for example create a mapping between logical expressions and fixed-length binary strings. A genetic algorithm then searches the space of strings. To evaluate strings they are mapped back to the corresponding logical expression which may then be interpreted. Other work has been done on suitably modifying the operators in genetic algorithms to manipulate logical expressions directly, e.g. SAMUEL [39], GLPS [43].

However, conventional evolutionary algorithms have three weaknesses that hinder their application to the problem of discovering classifiers. Firstly, because of the difficulty of encoding problems, evolutionary operators can construct candidate solutions that either have no corresponding classifier (syntactically invalid) or are meaningless in the problem domain (semantically invalid). Secondly, EAs perform well in the absence of domain knowledge. Yet when such knowledge is available, it is difficult to exploit. Thirdly, while EAs are good at locating good solutions in complex search spaces, they are poor at refining these solutions.

Two of the above problems, namely representing the space of classifiers and the use of declarative domain knowledge, may be elegantly addressed by using a grammar to make the syntax of classifiers explicit. A classifier can be represented as a derivation tree with respect to a grammar and such trees can be manipulated by a crossover operation similar to the operator in genetic programming [26]. Previous approaches to constraining search with an explicit grammar include GA-Miner [38], STGP [29, 23], LOGENPRO [46, 45, 44, 47] and [41, 40, 21]. Successful applications have been reported in medicine [35] and ecological modelling [42].

This leaves the third problem: allowing evolutionary algorithms to refine the classifiers they construct. Previous approaches have supplemented the genetic operators with operators that add or remove propositions (or conditions) from a rule. This approach is adopted in GABIL [8, 9], GIL [24], SAMUEL [17, 39, 20, 18, 19], SIA01 [1], REGAL [13, 14, 34]. A second approach found in REGAL and GIL involves selecting two rules at

random. These rules are then replaced with either a (most specific) generalisation or (most general) specialisation.

There is accumulating evidence to indicate that the performance of evolutionary algorithms can be improved through the introduction of a local search method [6, 22]. A local method progresses by refining an existing solution; instead of considering the entire search space, a small subset – the solution’s neighbourhood – is examined. This can result in the rapid location of good solutions. However, if all the points in the neighbourhood are inferior, then the algorithm becomes trapped. Unless the local method is perturbed in some way, no further improvements can be made. Evolutionary algorithms are relatively robust against such local maxima, but are poor at local refinement [15]. Next, we describe a local search method for constructing classifiers.

1.2. Inductive Logic Programming

Inductive logic programming (ILP) [32] is an approach to constructing classifiers that draws on the foundations of logic programming. The task tackled by ILP is that of developing predicate descriptions given training examples and background knowledge. More specifically, given sets of positive E^+ and negative E^- examples and background knowledge B , an ILP algorithm searches for an hypothesis H that is consistent and complete with respect to the training data and background knowledge. B, E^+, E^- and H are each logic programs. A logic program is a set of definite clauses each having the form

$$h \leftarrow b_1, b_2, \dots$$

where h is an atom and b_1, b_2, \dots is a set of atoms. Usually, E^+ and E^- contain only ground clauses, with empty bodies. The following symbols are used below¹: \wedge (logical and), \vee (logical or), \models (logically proves), \square (falsity). The conditions for construction of H are [30]

Necessity: $B \not\models E^+$

Sufficiency: $B \wedge H \models E^+$

Weak consistency: $B \wedge H \not\models \square$

Strong consistency: $B \wedge H \wedge E^- \not\models \square$

ILP algorithms perform a structured search through the space of hypothesis clauses. For instance, in the ILP algorithm Progol [33], a partial ordering is imposed on the space of clauses and this structures hypotheses by generality allowing large parts of the search space to be pruned. For example, if a clause C does not cover a positive example, then none of the specialisations of C need be considered. In practice, however, sufficiency and strong consistency conditions must be relaxed in order to tolerate noise in training data.

ILP algorithms are typically based on the set covering algorithm which is a greedy search algorithm. An optimisation algorithm is greedy if (1) it has to construct a solution in a sequence of stages, i.e. incrementally; and (2) at each stage the best possible local choice is made. The aim of a greedy algorithm is to construct a global optimum by a succession of local optimisations. But greedy algorithms are susceptible to local maxima in the search space, and an ILP algorithm based on a greedy algorithm may return suboptimal classifiers. For instance, to address this greediness some approaches perform a stochastic search for clauses as in SFOIL [36] and MILP [25] which both use a heuristic based on a Markovian neural network.

Inductive logic programming is an appealing local search method as it allows the easy incorporation of domain-specific knowledge.

2. Evolutionary Inductive Logic Programming

Inductive logic programming and evolutionary algorithms have appealing properties which appear to be complementary. Evolutionary algorithms have good global search properties, whereas inductive logic programming algorithms have good local refinement characteristics. This provokes the question can a classification algorithm be constructed that captures both of these properties?

2.1. An Overview

In many real-world applications, there is too much data to use in one batch. One approach is to sample the training data and supply only a subset to the learning algorithm. But as the sample be-

comes smaller it may be less representative of the underlying distribution and as a result less accurate rules may be induced. There is therefore a trade-off between run time and variance in accuracy. This is the basis for ensemble methods such as *bagging* [3]. As will be seen EVIL_1 exploits this property.

EVIL_1 is a hybrid algorithm consisting of an evolutionary algorithm and an ILP algorithm. The evolutionary component corresponds to a global strategy and maintains a population of clausal theories (logic programs). Data is partitioned into training, validation and test sets as shown in Figure 1. In EVIL_1, small samples of training data are supplied to an ILP algorithm and the risk of inaccurate rules is high. But to compensate for this, a theory's survival from one generation to the next is based on an estimate of its predictive accuracy. To compute this estimate logic programs are tested on a validation set.

Conventional mutation and crossover operators are replaced and mutation is performed by an inductive logic programming algorithm. Crossover is an operator which in some respects is similar to crossover in Genetic Programming [26]. The *EVIL_1* algorithm may be informally described as follows:

1. Repeat for n generations:
 - (a) Repeat for each individual in population
 - i. sample training set
 - ii. apply ILP algorithm to induce theory
 - iii. add theory to population
 - iv. if clause-exchange condition satisfied:
 - A. select two parent individuals
 - B. construct two new theories by exchanging parts of theory
 - C. add to the population
 - v. compute accuracy on validation set
 - (b) stochastically select population for next generation based on validation set accuracy.
2. Return theory with highest accuracy.

2.2. The EVIL_1 algorithm

EVIL_1 may be defined as follows. An *individual* is a clausal theory T which consists of background knowledge clauses B and hypothesis clauses H such that $T = B \wedge H$. The *population* P is a

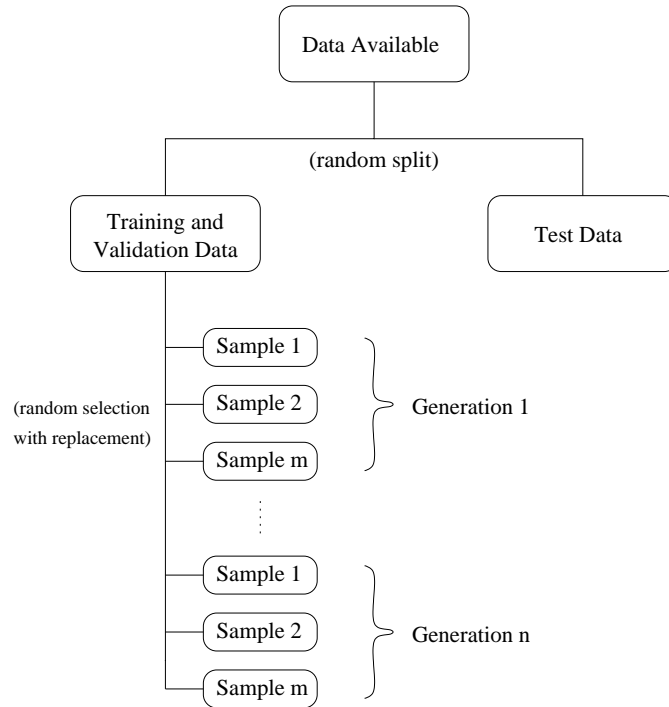


Fig. 1. Separation of data.

multiset of theories $\{T_1, T_2, \dots, T_n\}$. *Fitness* is a function $f : T \mapsto \mathfrak{R}$. The fitness of a theory $f(T)$ is its accuracy on the validation set. *Selection* is a function $S : P \mapsto P$. Let P_t be the population of theories at time t . The population at time $t + 1$ is $P_{t+1} = S(P_t)$ s.t. for all $T \in P_t$ the probability of $T \in P_{t+1}$ is given by

$$p(T \in P_{t+1}) = \frac{f(T)}{\sum_{T \in P} f(T)}.$$

The algorithm is detailed in Appendix A.1 and proceeds as follows. Initially, each theory in the population consists of the background knowledge and the null hypothesis (an empty set of induced clauses). In each of the following generations, an individual takes as input a small random sample of the training data and induces zero or more clauses using the Progol ([33]) inductive logic programming algorithm. The risk of inaccurate rules is high. But to compensate for this, the survival of a theory to the next generation is (probabilistically) determined by an estimate of its predictive accuracy.

Since an individual is a clausal theory, the population represents a sample over the space of the-

ories. The selection process manipulates this sample with respect to an estimate of predictive accuracy. To compute this estimate theories are tested on the validation set. The entire training set is used as a validation set—as some of the data will already have been used as training data, the estimate will be optimistic, but this is not important as the bias is the same for all theories and it is the accuracy relative to the population average that is used to determine survival. The estimate of predictive accuracy is used as the fitness of a theory. Consequently, those theories with poor predictive accuracy risk extinction, while those with high predictive accuracy are likely to occupy a larger proportion of the population in the next generation. In subsequent generations, a theory begins with the clauses induced in the previous generation. Learning is therefore incremental, with zero or more clauses being added each generation.

2.3. Clause Crossover

Crossover operators (as in genetic algorithms) manipulate fixed-length strings. However, logic programs do not map naturally to this setting; in-

stead logic programs are of variable length and are better represented by a tree. Genetic Programming (GP) [26] refers to a class of evolutionary algorithms that manipulate tree structures and consequently a variant of the GP crossover operation is used.

In EVIL₁, a *clause tree* $\mathcal{T} = (V, E)$ is a rooted binary tree where the vertices V are either the induced clauses in hypothesis H or null (\perp), i.e. $V \subset H \cup \{\perp\}$. As a result, a theory is not only represented as a set of clauses, but additional information is stored that captures relationships between them. Note that the clause tree does not include background knowledge, although this is an interesting direction that could be pursued.

The tree-structure specifies the interrelationships between clauses in the theory. Given any two clauses in the clause tree $c_i, c_j \in V$, there exists a unique path c_i, c_{i+1}, \dots, c_j . We define the *clause distance* $\Delta(c_i, c_j)$ to be the path length $j - i$. Initially, clauses are randomly placed in the tree and so clause distance is arbitrary. However, in practice, over a number of generations, certain clauses are observed to group together, reducing their clause distance.

Clause crossover is a function

$$\mathcal{C} : \mathcal{T} \times \mathcal{T} \mapsto \mathcal{T} \times \mathcal{T}$$

that involves choosing a random node on each tree and exchanging the subtrees below the chosen points. Since the crossover operator cannot cause a split within a clause, only syntactically valid theories can be generated; and because of the modular semantics of clauses, the addition and removal of clauses from a theory has a well-defined effect on the theory's semantics.

Given a tree T , the probability of any two clauses $c_i, c_j \in V$ being separated by a crossover operation is given by

$$\frac{\Delta(c_1, c_2)}{|V|}.$$

Consequently, the tree implicitly encodes the probability of clauses being separated by a crossover operation. The clause crossover algorithm is shown in Appendix A.1 and an example of a crossover operation is shown in Figure 2.

The frequency with which crossover is applied is governed by a parameter ω_c such that crossover operations occur exactly every ω_c generations.

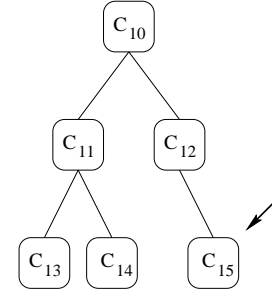
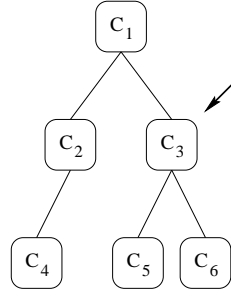
The Clause-Tree/Logic Program Mapping

It remains only to define the mapping between logic programs and trees.

1. *Creating a clause-tree from a logic program.*
 - (a) *If no clause-tree exists.* A tree is initially constructed by taking clauses in the logic program at random (without replacement) and creating either a root node or, if the root node already exists, attaching it at a random position in the tree. This results in a random mapping between clauses and their position in the clause tree.
 - (b) *If a clause-tree exists.* From generation to generation, Progol may add, remove or replace clauses in a theory. In order to preserve the properties of crossover the structure of clause trees should be disrupted as little as possible. This is achieved as follows. Copy the corresponding clause tree from the previous generation and replace all clauses with a null (\perp) symbol. Then for each clause in the logic program, check if the clause tree of the previous generation contained that clause. If so, it is placed in the position it held previously. If the clause was not in the tree before, place it in a random position on the tree. Thus clauses keep their previous position and removed clauses are replaced with null (\perp) placeholders thereby maintaining the tree structure. (This procedure is called `map2tree` in Appendix A.1).
2. *Creating a logic program from a clause-tree.* Construct an empty logic program; then perform an in-order traversal of the clause tree and for each node append the corresponding clause, thus building up a logic program. (This procedure is called `map2LP` in Appendix A.1).

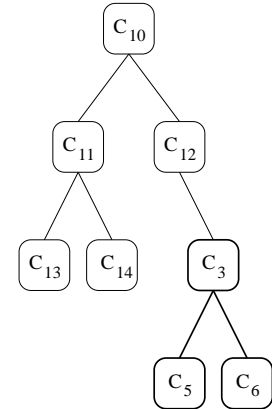
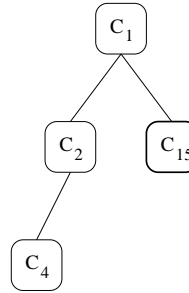
The following sections describe empirical analyses of the EVIL₁ algorithm.

C1: class(A,mammal) :- has_milk(A).
 C2: class(A,reptile) :- has_covering(A,scales), habitat(A,land).
 C3: class(A,bird) :- has_covering(A,feathers).
 C4: class(A,fish) :- has_gills(A).
 C5: class(A,reptile) :- has_covering(A,scales), has_legs(A,4).
 C6: class(A,fish) :- has_eggs(A).



C10: class(A,bird) :- has_legs(A,2), has_eggs(A).
 C11: class(A,mammal) :- has_legs(A,2), homeothermic(A).
 C12: class(A,reptile) :- has_legs(A,4).
 C13: class(A,fish) :- has_gills(A), habitat(A,water).
 C14: class(A,bird) :- habitat(A,land).
 C15: class(A,mammal) :- habitat(A,caves).

C1: class(A,mammal) :- has_milk(A).
 C2: class(A,reptile) :- has_covering(A,scales), habitat(A,land).
 C15: class(A,mammal) :- habitat(A,caves).
 C4: class(A,fish) :- has_gills(A).



C10: class(A,bird) :- has_legs(A,2), has_eggs(A).
 C11: class(A,mammal) :- has_legs(A,2), homeothermic(A).
 C12: class(A,reptile) :- has_legs(A,4).
 C13: class(A,fish) :- has_gills(A), habitat(A,water).
 C14: class(A,bird) :- habitat(A,land).
 C3: class(A,bird) :- has_covering(A,feathers).
 C5: class(A,reptile) :- has_covering(A,scales), has_legs(A,4).
 C6: class(A,fish) :- has_eggs(A).

Fig. 2. An example of clause crossover. Each clause in the logic program has a corresponding node in the clause tree. Subtrees are exchanged under the indicated crossover points. The resulting offspring trees correspond to two new logic programs.

3. Design of Experiments

The aim of this investigation is to examine the effect of introducing the evolutionary elements of EVIL1 and to examine the implications for problems where conventional Progol is weak. In particular, the objectives are:

1. to examine the relationship between classification accuracy and fitness-proportionate selection on the population of theories.
2. to examine the relationship between classification accuracy and the *type* of clause exchange mechanism used.
3. to examine the relationship between classification accuracy and the *frequency* of clause exchange.
4. to examine how the algorithm scales up to real world problems which involve:

- noisy data, such as class and attribute errors;
 - larger datasets;
 - more background knowledge.
5. and finally, to compare predictive accuracy with Progol.

Given these aims, the follow criteria were used in the choice of problem domains:

Large training sets. Most problems tackled using ILP algorithms deal with small datasets, typically under 10^4 training instances. Datasets should therefore exceed these limits.

Multiple-clause hypotheses. In EVIL1, the clause-crossover operator exchanges clauses between multiple instances of the ILP algorithm. However, if the hypothesis to be learned consists of only one clause, then the effect of clause crossover will not be seen. The

hypothesis should therefore consist of several clauses.

Of course, one may not know in advance whether a hypothesis requires several clauses. However, at this evaluation stage we address problems where this information is available.

Background knowledge. A forte of ILP algorithms is their ability to make use of explicit background knowledge. It is therefore desirable to consider problems where this knowledge can be exploited.

In the remaining sections, the evaluation of EVIL1 is described for two domains. The first problem concerns characterising illegal chess endgame positions. This problem is widely used as a benchmark for ILP systems. Secondly, a natural language processing problem is tackled, where the data comes from Wall Street Journal.

4. Empirical Study I: The Chess Endgame Domain

4.1. Problem description

The Chess Endgame (KRK) problem [31] is a widely used test problem for ILP systems. The problem may be characterised as follows. There are three pieces left on a chess board: the white king, white rook, and the black king. The objective of the learning algorithm is to discover rules to describe illegal positions when it is white’s turn to move, given a set of positive and negative training instances. For example, an illegal position occurs when the black king is in check with white to move. The Progol algorithm, given all the data in one batch, constructs a classifier with accuracy 99.42%.

4.2. Method

The predicate `illegal/6` is the target to be learned, and the attributes of the target predicate are file and rank for white king, white rook and black king respectively (Table 1). Examples, therefore, take the form `illegal(e,3,a,1,e,1)` and `:- illegal(d,4,g,3,b,5)` (where `:-` denotes negation). These examples correspond to

board positions as illustrated in Figure 3. The data consists of 20,000 examples which are split into 10,000 training and validation instances and 10,000 test instances.

The following background knowledge is also supplied. The `adj/2` predicate defines cases where the rank or file represented by the left argument is adjacent to that represented by the right argument. The `lt/2` predicate defines pairs of ranks or files, where the left argument is less than the right. Consequently, rules considered might include:

```
illegal(A,B,C,D,E,F) :- adj(A,E).
illegal(A,A,B,C,D,A) :- lt(A,B), lt(D,B).
```

It should be made clear that training and validation data are distinct from the test set, which is only used for evaluation independent of any learning.

The EVIL_1 algorithm was supplied with the following parameters: population size 10, sample size is 0.2% of the training set², 100 generations, 50 repetitions³. The following experiments were conducted.

Experiment 1. Selection The first experiment considers the effect of fitness proportionate selection. Two cases were considered: (1) the population for the next generation was chosen deterministically: if a new theory was superior to its parent, then it replaced the parent, otherwise the parent was carried forward; and (2) the members of the next generation were chosen stochastically with elitist fitness proportionate selection. In order to avoid confusion over which operator was responsible for phenomena, a crossover period of $\omega_c = \infty$ was used to disable crossover.

Figure 4 shows a scatter plot for the test set accuracy of the fitness-proportionate selection case (y -axis) against the no fitness proportionate selection case (x -axis). Points above the line $y = x$ reflect an improvement in predictive accuracy for the introduction of fitness-proportionate selection. However, the distribution of points indicates that while the introduction of fitness-proportionate selection is not advantageous, it is also not disadvantageous.

Experiment 2. Clause crossover The second experiment examined the effect of rule exchange between learners. At certain intervals denoted by the

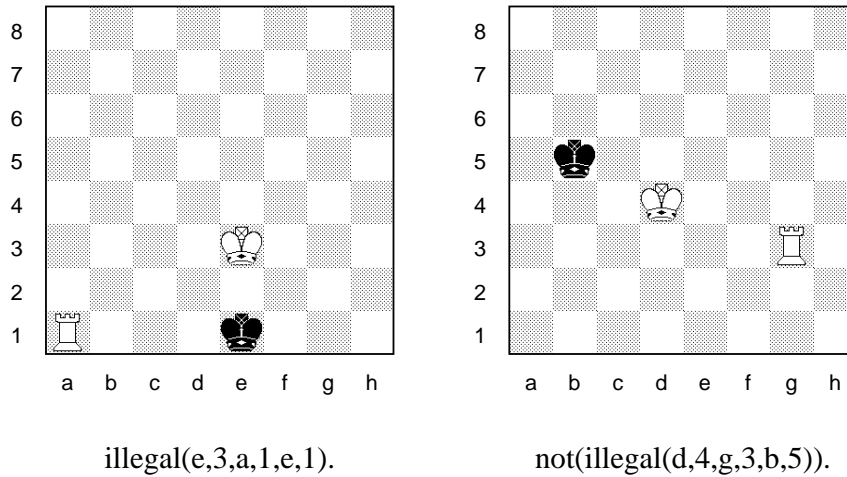


Fig. 3. Examples of legal and illegal positions.

Table 1. Attributes in the Target Predicate

| Attribute | Description | Values |
|-----------|--------------------|-------------|
| 1 | File of white king | $a \dots h$ |
| 2 | Rank of white king | $1 \dots 8$ |
| 3 | File of white rook | $a \dots h$ |
| 4 | Rank of white rook | $1 \dots 8$ |
| 5 | File of black king | $a \dots h$ |
| 6 | Rank of black king | $1 \dots 8$ |

communication period ω_c theories are selected to exchange parts of their clause tree. In the control case $\omega_c = \infty$, no rule exchange occurs. In the treatment cases, $\omega_c = 3, 5$ or 10 generations. The following types of rule-exchange were considered. (1) Union: new theories are constructed by taking the union of two parent theories; (2) Crossover: new theories are constructed by ex-

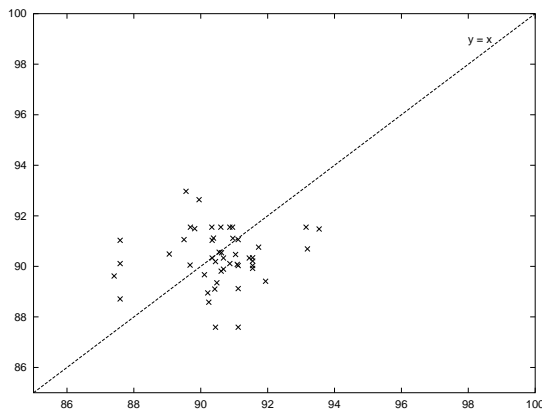


Fig. 4. Fitness-proportionate selection versus no fitness-proportionate selection.

Table 2. The statistical significance of introducing crossover for rule exchange at different crossover frequencies.

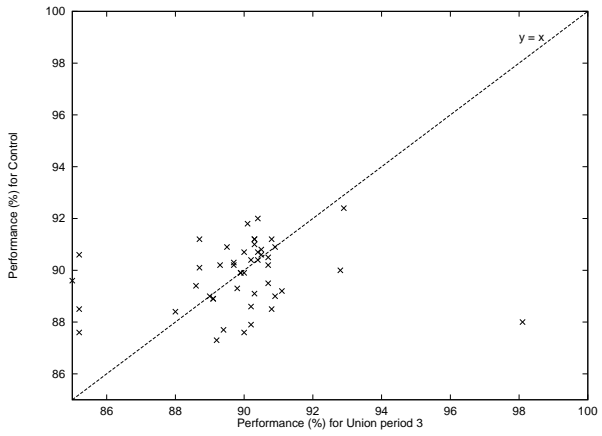
| Crossover period ω_c | Significance |
|-----------------------------|--------------|
| 3 | $P < 0.0005$ |
| 5 | $P < 0.05$ |
| 10 | $P < 0.1$ |

changing rules using the crossover operator described in Section 2.3.

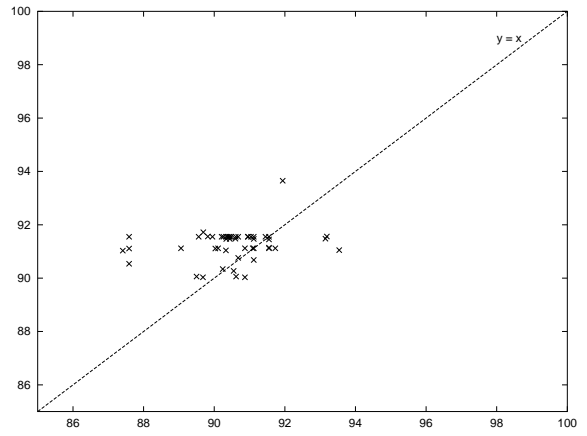
Figures 5(a),(c),(e) show scatter plots for the test set accuracy of rule exchange using union, and Figures 5(b),(d),(f) the performance distributions for crossover. The graphs indicate that (1) the union operation increases the variance of predictive accuracies but has little effect on the mean; and (2) crossover has little effect on variance but increases mean predictive accuracy. The overall statistical significance of these results was examined using the paired t -test. For union periods 3, 5 and 10, the introduction of union does not result in a statistically significant increase in predictive accuracy and therefore it may not be reasonably asserted that union improves performance in the chess endgame problem. However, the exchange of rules through crossover does result in a statistically significant increase. See Table 2.

4.3. A Closer Look at Crossover

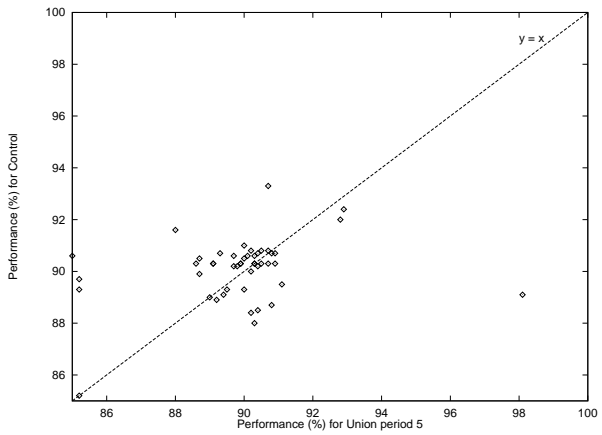
To examine the effect of crossover in greater detail, learning curves are plotted for runs with and without crossover. Figures 6 and 7 show the training set accuracy of eight representative runs selec-



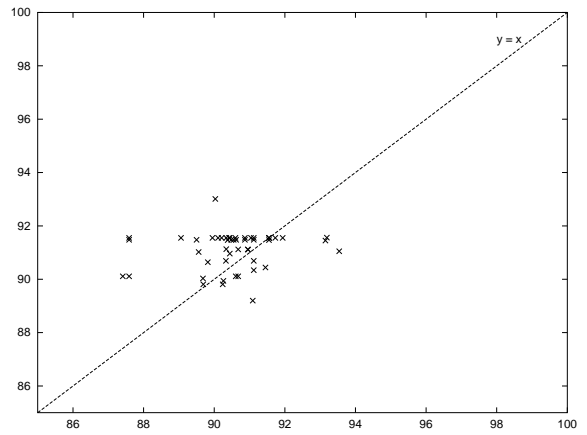
(a) Union $\omega_u = 3$



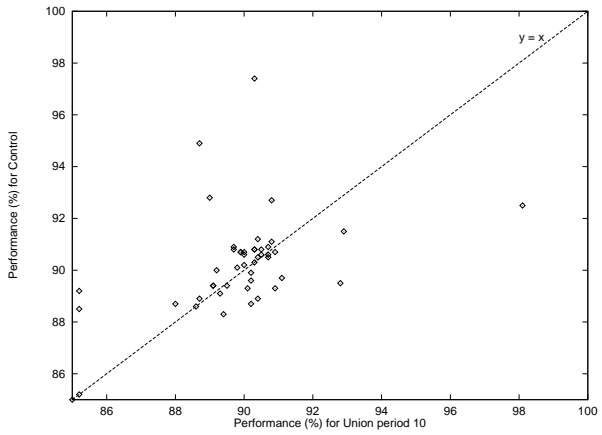
(b) Crossover $\omega_c = 3$



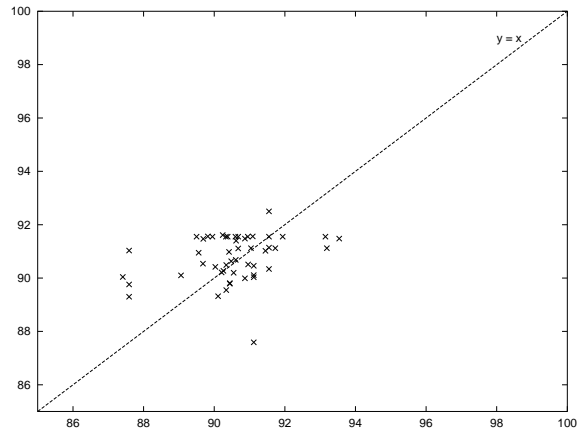
(c) Union $\omega_u = 5$



(d) Crossover $\omega_c = 5$



(e) Union $\omega_u = 10$



(f) Crossover $\omega_c = 10$

Fig. 5. Distributions of generalisation accuracy for crossover/union (y) against no crossover/union (x).

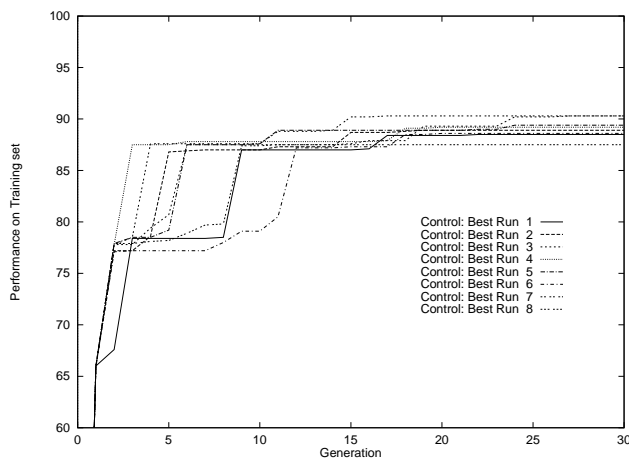


Fig. 6. Training set accuracy plotted against number of generations with no crossover.

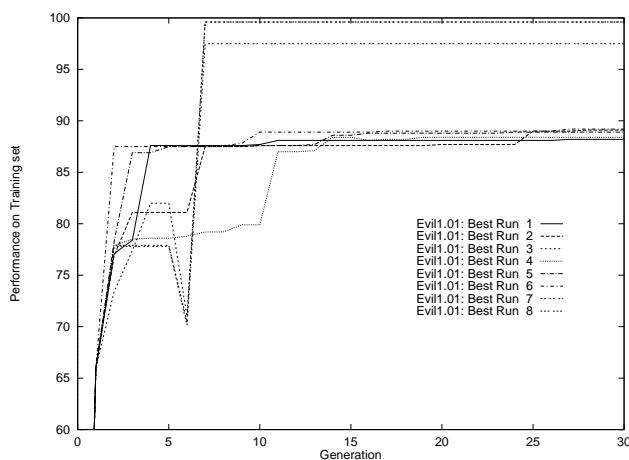


Fig. 7. Training set accuracy plotted against number of generations with crossover.

tion of runs. With no crossover (Figure 6) it can be seen that in each case accuracy converges at around 88%. However, when crossover is introduced with $\omega_c = 5$ the outcome is different (Figure 7). While in many runs the accuracy still converges at around 88%, there are cases where accuracy is dramatically increased to 97% and 99%. Furthermore, it is interesting to note that in these cases the accuracy dips after the crossover operation, shortly before reaching a maximum value. This indicates that crossover is having a disruptive effect that subsequently allows the rule induction algorithm to construct a more accurate set of rules. This is clearly a promising result. However, it should be stressed that high training set accuracy does not necessarily lead to high test set accuracy.

5. Empirical Study II: Part-of-Speech Tagging

The chess endgame problem used only 10,000 training examples and employed little background knowledge (only 2 predicates) and the data for this artificial domain was noiseless. In the following problem, the data is noisy and far more background knowledge is required.

5.1. Problem description

The problem addressed is a natural language processing problem. In analysing the syntactic structure of sentences in a text it is necessary to group words into classes or categories. Part-of-Speech

Tagging is a linguistic procedure which attaches word category information to the words in a text. Rules define how, given an input word sequence, each word is assigned its corresponding part-of-speech tag.

More formally, given a sentence S , which can be defined as a string of words w_1, w_2, \dots, w_n , part-of-speech tagging is the process of assigning each word w_i of S a corresponding tag t_i from the set T of tags. These tags are defined *a priori* by the user. Consequently for each sentence S we get an *alignment*

$$(S, T) = w_1 t_1, w_2 t_2, \dots, w_n t_n.$$

Note, however, that sentences can have more than one alignment because there exist more than one tag for each word. But, only one should be considered correct, and the tagging process should select this correct alignment.

A *tagged corpus* comprises of word and tag pairs that have been identified manually. However, developing a tagged corpus is a laborious task and consequently there is significant interest in automating this process. The learning task is to discover rules from a tagged corpus to disambiguate sentences by eliminating sentence alignments.

Cussens [5] has used inductive logic programming to tackle this problem. When only a simple grammar of English was supplied as background knowledge, ILP proved competitive at this task. However, the algorithm employed (P-Progol) could only use a fraction of the available data. For instance, for one tag (nn) there are in excess of 100,000 examples yet only 6000 were used for training. Even to deal with this small proportion of the data the ILP algorithm had to be extended by adding a caching facility [5].

5.2. Method

The tagged corpus used is the Wall Street Journal, which is part of the Penn Treebank [28]. The corpus contains sentences such as “*Rudolf Agnew, 55 years old and former chairman of Consolidated Gold Fields PLC, was named a nonexecutive director of this British industrial conglomerate*”. Each word in the corpus has been manually assigned one of 38 part-of-speech tags. For instance,

cd is a cardinal number, dt is a determiner and nn is a singular noun. (A list of tags is given in Table 4). The Wall Street Journal data was prepared for use with Progol by Cussens; details may be found in [4, 5]. The data was partitioned into 2/3 training and 1/3 test data. Following Cussens, the same simple grammar of English was used as background knowledge. This grammar comprised of some 120 Prolog clauses, an excerpt of which is shown in Appendix A.2.

The EVIL_1 algorithm was supplied with the following parameters: population size 10, sample size is 50 examples, 100 generations, 30 repetitions. The following three experiments were conducted.

Experiment 1. Selection The first examined the effect of fitness proportionate selection. Two cases were considered: (1) the population for the next generation was chosen deterministically: if a new theory was superior to its parent, then it replaced the parent, otherwise the parent was carried forward; and (2) the members of the next generation were chosen stochastically with elitist fitness proportionate selection. In order to avoid confusion over which operator was responsible for phenomena, a crossover period of $\omega_c = \infty$ was used to disable crossover. The predictive accuracy on the test set was recorded in each case. This procedure was followed for one tag, nn, using only the first 6000 training examples. This methodology allows comparison with the results obtained by Cussens for the nn tag⁴

Examples of input and output may be found in Appendix A.2. Figure 9 shows a scatter plot for the test set accuracy of the fitness-proportionate selection case (y -axis) against the deterministic selection case (x -axis). Points above the line $y = x$ reflect an improvement in predictive accuracy for fitness-proportionate selection. The distribution of points indicates that the use of fitness-proportionate selection is not advantageous, if anything it is slightly disadvantageous. However, when the accuracy is compared to that obtained using a greedy ILP strategy (73.8%), the accuracies are consistently over 76% regardless of whether deterministic or fitness-proportional stochastic selection was used. This would indicate that for the nn tag merely sampling the data into subsets is advantageous.

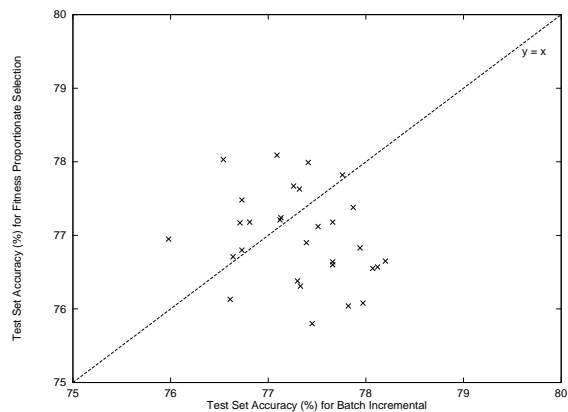


Fig. 9. Comparison of accuracies for the nn tag between stochastic fitness-proportionate selection and deterministic selection.

Experiment 2. Clause crossover The second experiment examined the effect of rule exchange between learners. Once again elimination rules are learned for the nn tag using only the first 6000 training examples, but at certain intervals, governed by the communication period ω_c , theories are selected for clause exchange. In the control case $\omega_c = \infty$, no clause exchange occurs. In the

Table 3. The statistical significance of introducing crossover for rule exchange at different crossover frequencies.

| Crossover period ω_c | Significance |
|-----------------------------|--------------|
| 3 | $P < 0.05$ |
| 5 | $P < 0.01$ |
| 10 | $P < 0.2$ |

treatment cases, ($\omega_c = 3, 5$ or 10 generations), new theories are constructed using clause crossover.

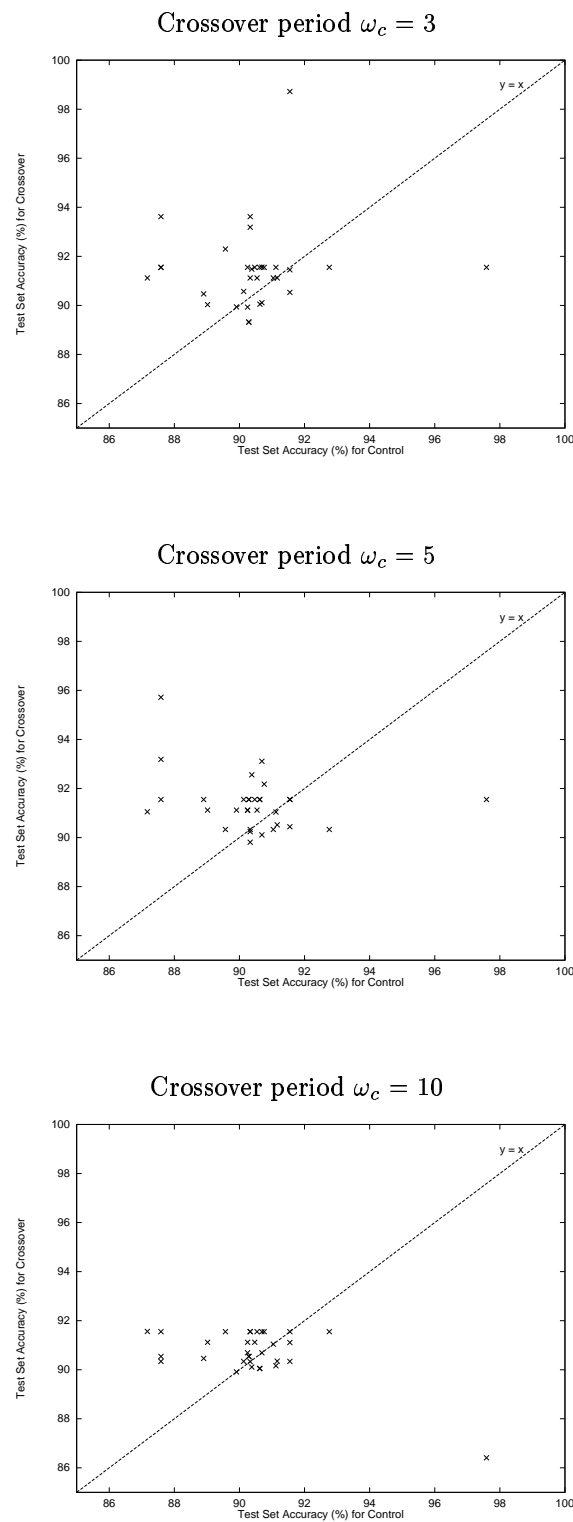


Fig. 8. Test set accuracies for the nn tag for crossover periods $\omega_c = 3, 5, 10$ plotted against no crossover, $\omega_c = \infty$.

Figure 8 shows scatter plots for the test set accuracy of rule exchange using crossover. The graphs indicate that there are a number of instances where the introduction of crossover increases predictive accuracy. Furthermore, the improvement is more significant with increasingly frequent crossover. This would suggest that crossover is directly contributing to an increase in test set accuracy. The statistical significance of introducing crossover was examined using the paired t -test⁵. For crossover periods 3 and 5, the introduction of crossover results in a statistically significant increase in predictive accuracy at the 95% confidence level (Table 3).

Experiment 3. Large data-sets The third and final experiment examined the effect of sampling from the entire training set instead of only the first 6000 examples. The EVIL_1 algorithm was run once for each tag with crossover periods $\infty, 3, 5, 10$.

The test set accuracies are shown in Table 4. Results indicate that when crossover is used with period $\omega_c = 5$ the greatest increase in accuracy is achieved. Compared with no crossover this increase is statistically significant ($P < 0.002$).

Finally, the performance of the greedy ILP strategy may be compared with EVIL_1. Results indicate that of the 34 tag concepts learned, EVIL_1 with crossover period $\omega_c = 5$ resulted in an improvement for 28 tags, statistical significance ($P < 0.005$). While these results appear very promising, there is a caveat. Because of the properties of the ILP algorithm he used, Cussens placed an upper bound on the size of training and test data. In the method reported here, no such upper bound was used; all the data was used (in some cases over 70,000 training examples). The increased accuracies in the last experiment could therefore be accounted for by changes in the underlying distribution in the data, i.e. Cussens' 6000 training and 3000 test examples were unrepresentative of the true distribution. However, it is perhaps unlikely that this should occur in so many cases.

6. Discussion and Future Directions

There is a price to pay for sampling. Regularities that occur in a dataset may not be present

in samples taken from it. For the chess endgame, the accuracy of the evolutionary approaches never exceed 94%, while Progol using all the data at once obtains 99.42%. One can conclude (1) that the KRK domain is sensitive to sampling and (2) that this domain does not suffer from local optima. The same is not true for the Wall Street Journal data. For the majority of tags, the rules learned had higher accuracy through sampling. But the use of less training data to construct better classifiers is distinctly counter-intuitive and warrants further consideration. The clause-crossover operation produced only small improvements in accuracy in both domains. These were regular enough to be statistically significant.

There exist a number of further directions that are being pursued:

Automatic Parameterisation. Experiments are necessary to investigate how EVIL1 behaves as parameters are varied. At present, the parameters that must be supplied include training set sample size, population size, and crossover frequency. The automatic variation of parameters from generation to generation has previously been studied in evolutionary algorithms [2]. This may provide a way of avoiding the problem of supplying implicit bias.

Sampling Strategy. Another interesting area that should be analysed more closely relates to the sampling strategy used. At present EVIL1 takes a random sample with replacement. However, Progol's set-covering algorithm immediately removes all covered examples. Consequently, new clauses are learned only on the remaining examples, which may be considerably less than the original sample. Alternative strategies to study include 1) sample only from uncovered examples; 2) let the training set of generation $n + 1$ be the training set at generation n plus a sample of the complete data; 3) Windowing [37].

Crossover. EVIL1 constructs new clausal theories by exchanging clauses of the target concept only. The effect of allowing crossover to exchange background knowledge clauses between classifiers has not been investigated. This would have the effect of allowing the hypothesis language to be manipulated too.

Table 4. Part-of-Speech tags; the distribution of examples; accuracy using ILP by Cussens [5]; accuracy using EVIL_1 with different crossover frequencies. n/a indicates insufficient data to learn rules.

| Tag | Data | | | Accuracy | | | | |
|-----|-------|-------|--------|----------|---------------------|----------------|----------------|-----------------|
| | Pos | Neg | Total | ILP | EVIL_1 | | | |
| | | | | | $\omega_c = \infty$ | $\omega_c = 3$ | $\omega_c = 5$ | $\omega_c = 10$ |
| cc | 1192 | 338 | 1530 | 59.1 | 77.65 | 78.24 | 77.65 | 77.65 |
| cd | 797 | 2327 | 3124 | 75.1 | 79.65 | 78.98 | 80.13 | 81.77 |
| cln | 8 | 0 | 8 | n/a | n/a | n/a | n/a | n/a |
| cma | 0 | 0 | 0 | n/a | n/a | n/a | n/a | n/a |
| dt | 11778 | 6770 | 18548 | 82.9 | 85.77 | 86.01 | 85.93 | 85.71 |
| dlr | 0 | 0 | 0 | n/a | n/a | n/a | n/a | n/a |
| ex | 182 | 660 | 842 | 88.1 | 94.68 | 95.04 | 94.68 | 94.33 |
| fw | 362 | 163 | 525 | 28.4 | 75.43 | 77.71 | 77.71 | 75.43 |
| in | 16448 | 25100 | 41548 | 82.0 | 77.60 | 77.51 | 77.40 | 77.90 |
| jj | 40041 | 35149 | 75190 | 62.7 | 74.67 | 76.13 | 75.88 | 74.66 |
| jjr | 3255 | 4182 | 7437 | 76.6 | 77.89 | 80.19 | 79.83 | 78.30 |
| jjs | 879 | 1068 | 1947 | 71.4 | 80.59 | 79.82 | 81.51 | 80.43 |
| lpn | 0 | 0 | 0 | n/a | n/a | n/a | n/a | n/a |
| lqt | 0 | 0 | 0 | n/a | n/a | n/a | n/a | n/a |
| ls | 5 | 4 | 9 | n/a | n/a | n/a | n/a | n/a |
| md | 209 | 102 | 311 | 94.1 | 78.10 | 79.05 | 78.10 | 78.10 |
| nn | 54837 | 50394 | 105231 | 73.8 | 78.34 | 78.81 | 78.81 | 78.81 |
| nns | 7826 | 13460 | 21286 | 79.4 | 80.90 | 82.20 | 82.00 | 80.95 |
| np | 16526 | 21672 | 38198 | 62.8 | 79.02 | 79.09 | 79.05 | 78.97 |
| nps | 5817 | 3483 | 9300 | 65.0 | 63.77 | 63.29 | 62.94 | 63.90 |
| pdt | 3231 | 470 | 3701 | 90.8 | 90.93 | 91.98 | 91.98 | 92.47 |
| pnd | 0 | 0 | 0 | n/a | n/a | n/a | n/a | n/a |
| pos | 1185 | 14318 | 15503 | 98.8 | 99.15 | 99.15 | 99.15 | 99.15 |
| pp | 511 | 168 | 679 | 83.7 | 91.19 | 90.31 | 90.75 | 91.19 |
| ppz | 152 | 511 | 663 | 82.1 | 91.86 | 91.86 | 92.31 | 92.31 |
| rb | 29352 | 13483 | 42835 | 64.7 | 71.99 | 73.92 | 73.80 | 74.51 |
| rbr | 5150 | 2855 | 8005 | 56.8 | 74.07 | 73.96 | 75.46 | 74.86 |
| rbs | 1064 | 720 | 1784 | 78.3 | 87.75 | 87.75 | 87.75 | 87.75 |
| rp | 6757 | 2414 | 9171 | 61.7 | 71.48 | 71.08 | 72.78 | 71.48 |
| rpn | 0 | 0 | 0 | n/a | n/a | n/a | n/a | n/a |
| rqt | 1043 | 0 | 1043 | 100.0 | 0.00 | 0.00 | 0.00 | 0.00 |
| stp | 11 | 1 | 12 | 0.0 | n/a | n/a | n/a | n/a |
| sym | 11 | 18 | 29 | 100.0 | 81.82 | 81.82 | 81.82 | 81.82 |
| to | 0 | 0 | 0 | n/a | n/a | n/a | n/a | n/a |
| uh | 195 | 33 | 228 | 82.4 | 89.47 | 81.58 | 88.16 | 81.58 |
| vb | 42673 | 29464 | 72137 | 92.4 | 95.06 | 95.35 | 95.35 | 95.32 |
| vbd | 27664 | 22701 | 50365 | 84.9 | 84.25 | 85.68 | 85.30 | 85.88 |
| vbg | 9701 | 8738 | 18439 | 67.9 | 78.10 | 77.47 | 77.96 | 78.71 |
| vbn | 28046 | 26665 | 54711 | 78.2 | 78.36 | 81.38 | 81.38 | 80.23 |
| vbp | 29465 | 10875 | 40340 | 89.2 | 92.32 | 93.50 | 92.32 | 92.32 |
| vbz | 22611 | 4745 | 27356 | 78.7 | 89.22 | 91.43 | 91.57 | 90.20 |
| wdt | 8954 | 3287 | 12241 | 92.5 | 97.21 | 97.30 | 97.48 | 97.48 |
| wp | 10 | 5 | 15 | 20.0 | 20.00 | 20.00 | 20.00 | 20.00 |
| wpz | 0 | 0 | 0 | n/a | n/a | n/a | n/a | n/a |
| wrb | 0 | 0 | 0 | n/a | n/a | n/a | n/a | n/a |

This could be useful where the background knowledge is possibly erroneous.

Population-based Prediction. The output of EVIL1 is the classifier with the highest fitness in the final generation. However, results obtained with ensemble methods, e.g. Bagging [3], suggest that better predictions can be obtained using multiple classifiers. This may prove beneficial in EVIL1 providing the population has not converged to a unique classifier.

7. Conclusions

A new classification algorithm has been presented that uses an evolutionary algorithm as a wrapper around an inductive logic programming algorithm. A key feature of the approach is the use a population of competing classifiers constructed from small samples of the training set. The fitness of classifiers is defined as the accuracy on a validation set.

The algorithm's learning properties were examined on both artificial and real-world data. Re-

sults indicated that this approach was suboptimal with smaller noise-free data-sets. However, when data-sets are large and noisy, this approach is highly effective. One possible explanation for these results is that the ILP algorithm is based on the greedy algorithm which is susceptible to local minima. The evolutionary wrapper serves as a global strategy, which can redirect the ILP algorithm to other areas of the search space.

Acknowledgements

The authors would like to express their gratitude to Ross King for suggesting the tagging problem; members of Penn Treebank Project for supplying the linguistic data; James Cussens for providing the background knowledge and preprocessed data, and Stephen Muggleton for making available C-Progol. The authors would also like to thank James Cussens, Peter Whigham, Jan Zytkow and the anonymous reviewers for their valuable comments.

Appendix

A.1. The Evil_1 Algorithm

The EVIL_1 algorithm. The `induce` procedure refers to a call of the Progol inductive logic programming algorithm.

```

1. procedure EVIL_1 is
2. begin
3.   initialise(population);
4.   fitness = accuracy(population,validation_set);

5.   while not termination_criterion loop
6.     for each member of population loop
7.       theory = select_parent(population);
8.       subset = sample(training_set,sample_size);
9.       new_theory = induce(background_knowledge, theory, subset);
10.      fitness(new_theory) = accuracy(new_theory, validation_set);
11.    end loop

12.    if generation %  $\omega_c = 0$  then
13.      crossover(population);
14.    end if

15.    new_population = select(population);
16.    population = new_population;
17.    generation = generation + 1;
18.  end loop
19.  return fittest(population)
20. end EVIL_1;

1. procedure crossover(population) is
2. begin
3.   for each member of population / 2 loop
4.     parent_program_1 = select_parent(population);
5.     parent_program_2 = select_parent(population);
6.     parent_tree_1 = map2tree(parent_program_1);
7.     parent_tree_2 = map2tree(parent_program_2);
8.     <offspring_tree_1, offspring_tree_2> = exchange_random_subtree(parent_tree_1,parent_tree_2);
9.     offspring_program_1 = map2LP(offspring_tree_1);
10.    offspring_program_2 = map2LP(offspring_tree_2);
11.    fitness(offspring_program_1) = accuracy(offspring_program_1, validation_set);
12.    fitness(offspring_program_2) = accuracy(offspring_program_2, validation_set);
13.  end loop
14.  return population;
15. end crossover;

```

A.2. Examples of training data, background knowledge and induced theories

A sentence from the Wall Street Journal corpus together with two examples constructed for the highlighted word. The negative example is preceded by :-.

Rudolf Agnew, 55 years old and former chairman of Consolidated Gold Fields PLC, was named a nonexecutive director of this British industrial conglomerate.

```
:- rmv([in,nn,jj,cc,jj,nns,cd,cma,np,np],
      [np,np,np,cma,vbd,vbn,dt,jj,nn,in,dt,jj,jj,nn,stp],
      np).

rmv([in,nn,jj,cc,jj,nns,cd,cma,np,np],
     [np,np,np,cma,vbd,vbn,dt,jj,nn,in,dt,jj,jj,nn,stp],
     jj).
```

Excerpts of the grammar used as background knowledge (in Prolog).

```
%noun classes

%proper noun
prnoun([np|S],S) :- !.
prnoun([nps|S],S).

% noun phrase - used
nounp(L1,L2) :- noun(L1,X), !, (X=L2 ; snp(X,L2)). % "Pierre Vincken"
nounp([pp|S],S) :- !. % "us, him"
nounp(L1,L2) :- dtz(L1,L3), !, (snp(L3,L2) ; adjp(L3,L4), snp(L4,L2)).
nounp(L1,L2) :- cncy(L1,L3), !, cds(L3,L2). % "$ 20 billion "
nounp([cd|Y],L2) :- !, (X=Y ; cds(Y,X)), (X=L2 ; snp(X,L2)). % "20, 20%"
nounp(L1,L2) :- adjp(L1,L3), snp(L3,L2). % "green man"
```

A theory learned for the in tag.

```
rmv(A,B,in) :- sverb(B,C).
rmv(A,B,in) :- dt(A,C).
rmv(A,B,in) :- vpart(A,C), vp1(C,D).
rmv(A,B,in) :- noun(B,C), stp(C,D).
rmv(A,B,in) :- nounp1(A,C), cma(C,D).
```

Notes

1. A more complete set of logic programming definitions may be found in [27].
2. The choice of sample size was based on a trade-off between providing enough data for rules to be found while avoiding excessive run-times.
3. The number of repetitions was chosen for statistical significance.
4. Cussens also evaluated the tag theories as a complete tagging system together with statistical extensions. Although this provides a more accurate reflection of the value of the induced theories, this is not the aim in this work.
5. It should be noted that when comparing classifiers, the paired *t*-test can give over-optimistic results [11]. The statistical significance of these results should be treated with appropriate caution.

References

1. S. Augier, G. Venturini, and Y. Kodratoff. Learning first order logic rules with a genetic algorithm. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pages 21–26, 1995.
2. Thomas Bäck. *Evolutionary algorithms in theory and practice*. Oxford University Press, 1996.
3. L. Breiman. Bagging predictors. Technical Report No. 421, University of California, Berkeley, Department of Statistics, September 1994.
4. James Cussens. Part-of-speech disambiguation using ilp. Technical Report PRG-TR-25-96, Oxford University Computing laboratory, 1996.
5. James Cussens. Part-of-speech tagging using progol. In *Inductive Logic Programming: Proceedings of the 7th International Workshop (ILP-97), volume 1297 of Lecture Notes in Artificial Intelligence*, pages 93–108. Springer, 1997.
6. L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.
7. Kenneth A. DeJong and William M. Spears. Using genetic algorithms to solve NP-complete problems. In *International Conference on Genetic Algorithms*, pages 124–132, 1989.
8. Kenneth A. DeJong and William M. Spears. Learning concept classification rules using genetic algorithms. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 651–656, 1991.
9. Kenneth A. DeJong, William M. Spears, and Diana F. Gordon. Using genetic algorithms for concept learning. *Machine Learning*, 13:161–188, 1993.
10. T. G. Dietterich. Machine learning research: Four current directions. *AI Magazine*, 18(4):97–136, 1997.
11. T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1924, 1998.
12. David Fogel and Chris deSilva, editors. *Proceedings of the 1995 IEEE International Conference on Evolutionary Computation*, Perth, Western Australia, November 1995. IEEE, IEEE Computer Society Press.
13. A. Giordana and L. Saitta. Regal: an integrated system for learning relations using genetic algorithms. In *Proceedings of 2nd International Workshop on Multi-strategy Learning*, pages 234–249. Morgan Kaufmann, 1993.
14. Attilio Giordana, Lorenza Saitta, and Floriano Zini. Learning disjunctive concepts by means of genetic algorithms. In *Proceedings of the 11th International Conference on Machine Learning*, pages 96–104, 1994.
15. D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Reading, MA, 1989.
16. David E. Goldberg. Genetic and evolutionary algorithms come of age. *Communications of the ACM*, Vol. 37:113–119, March 1994.
17. John J. Grefenstette. A system for learning control strategies with genetic algorithms. pages 183–190, George Mason University, 1989. Naval Research Laboratory.
18. John J. Grefenstette. The evolution of strategies for multiagent environments. *Adaptive Behaviour*, 1(1):65–90, 1992.
19. John J. Grefenstette. Lamarckian learning in multi-agent environments. In *Proceedings of 4th International Conference of Genetic Algorithms*, pages 303–310. Morgan Kaufmann, 1992.
20. John J. Grefenstette. Learning decision strategies with genetic algorithms. *Proceedings of the International Workshop on Analogical and Inductive Inference, Lecture Notes in Artificial Intelligence 642*, pages 35–50, 1992.
21. Frédéric Gruau. On using syntactic constraints with genetic programming. In P.J. Angeline and Jr. Kenneth E. Kinneer, editors, *Advances in Genetic Programming III*, chapter 19, pages 377–394. MIT Press, Cambridge, MA, 1996.
22. William E. Hart and Richard K. Belew. Optimization with genetic algorithm hybrids that use local search. In Richard K. Belew and Melanie Mitchell, editors, *Adaptive Individuals in Evolving Populations: Models and Algorithms.*, volume 26, chapter 27, pages 483–496. SFI Studies in the Sciences of Complexity, 1996.
23. Thomas D. Haynes, Dale A. Schoenefeld, and Roger L. Wainwright. Type inheritance in strongly typed genetic programming. In Peter J. Angeline and K. E. Kinneer, Jr., editors, *Advances in Genetic Programming 2*, chapter 18, pages 359–376. MIT Press, Cambridge, MA, USA, 1996.
24. Cezary Z. Janikow. A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 13:189–228, 1993.
25. Matevž Kovačič. *Stochastic Inductive Logic Programming*. PhD thesis, University of Ljubljana, Slovenia, 1994.
26. John R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
27. J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, 1984.

28. M.P. Marcus, B. Santorini, and M.A. Marcinkiewicz. Building a large annotated corpus of english: the penn treebank. *Computational Linguistics*, 19, 1993.
29. David J. Montana. Strongly typed genetic programming. BBN Technical Report #7866, Bolt Beranek and Newman, Inc., 10 Moulton Street, Cambridge, MA 02138, USA, March 1994.
30. S. Muggleton. Inductive logic programming: derivations, successes and shortcomings. *SIGART Bulletin*, 5(1):5–11, 1994.
31. S. H. Muggleton, M. Bain, J. Hayes-Michie, and D. Michie. An experimental comparison of human and machine learning formalisms. In *Proc. Sixth International Workshop on Machine Learning*, pages 113–118, San Mateo, CA, 1989. Morgan Kaufmann.
32. Stephen Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.
33. Stephen Muggleton. Inverse Entailment and Progol. *New Generation Computing*, 13, 1995.
34. Filippo Neri. *First Order Logic Concept Learning by means of a Distributed Genetic Algorithm*. PhD dissertation, Università of Torino, Dipartimento di Informatica, 1997.
35. P.S. Ngan, M.L. Wong, W. Lam, K.S. Leung, and J.C.Y. Cheng. Medical data mining using evolutionary computation. *Artificial Intelligence in Medicine*, 16:73–96, 1999.
36. U. Pompe, M. Kovačič, and I. Kononenko. Sfoil: Stochastic approach to inductive logic programming. In *Proceedings of the 2nd Electrotechnical and Computer Science Conference*, page 31, Portoroz, Slovenija, 1993.
37. J. R. Quinlan. Discovering rules from large collections of examples: a case study. In D. Michie, editor, *Expert Systems in the Micro-electronic Age*, pages 168–201. Edinburgh University Press, Edinburgh, 1979.
38. N J Radcliffe and P D Surry. Cooperation through hierarchical competition in genetic data mining. Technical Report TR9409, University of Edinburgh, Parallel Computing Centre, Edinburgh, 1994.
39. A. C. Schultz and J. J. Grefenstette. Improving tactical plans with genetic algorithms. In *Proceedings of the 2nd International IEEE Conference on Tools for Artificial Intelligence*, number IEEE Cat. No. 90CH2915-7, pages 328–334, Herndon, VA, 6-9 Nov 1990. IEEE Computer Society Press, Los Alamitos, CA.
40. P. A. Whigham. Search bias, language bias, and genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996 : Proceedings of the First Conference*, Stanford University, CA, USA, 28–31 July 1996. MIT Press.
41. P.A. Whigham. Grammatically-based genetic programming. In J. Rosca, editor, *Proc. Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41. Morgan Kaufmann, 1995.
42. Peter A. Whigham and Peter F. Crapper. Time series modelling using genetic programming: An application to rainfall-runoff models. In Lee Spector, William B. Langdon, Una-May O'Reilly, and Peter J. Angeline, editors, *Advances in Genetic Programming* 3, chapter 5, pages 89–104. MIT Press, Cambridge, MA, USA, May 1999. Forthcoming.
43. Man Leung Wong and Kwong Sak Leung. Inductive logic programming using genetic algorithms. In J.W. Brahan and G.E. Lasker, editors, *Advances in Artificial Intelligence – Theory and Application II*, pages 119–124, 1994.
44. Man Leung Wong and Kwong Sak Leung. Applying logic grammars to induce sub-functions in genetic programming. In Fogel and deSilva [12], pages 737–740.
45. Man Leung Wong and Kwong Sak Leung. Combining genetic programming and inductive logic programming using logic grammars. In Fogel and deSilva [12], pages 733–736.
46. Man Leung Wong and Kwong Sak Leung. An induction system that learns programs in different programming languages using genetic programming and logic grammars. In *Seventh International Conference on Tools with Artificial Intelligence*, pages 380–387, Herndon, Virginia, November 1995. The Chinese University of Hong Kong, IEEE Computer Society Press.
47. Man Leung Wong and Kwong Sak Leung. Evolving recursive functions for the even-parity problem using genetic programming. In P.J. Angeline and Jr. K.E. Kinnear, editors, *Advances in Genetic Programming 2*, Massachusetts, 1996. MIT Press.

Philip Reiser is an honorary research fellow in the computer science department at the University of Auckland, New Zealand. He is currently working on a project funded by the Boeing Company. His research interests include evolutionary models of learning, inductive logic programming and knowledge discovery in databases.

Patricia Riddle is a senior lecturer in the computer science department at the University of Auckland, New Zealand. Previously, she was a senior principal scientist with the Boeing Company. Her research interests are machine learning with particular emphasis on practical applications of machine learning or datamining, and the process of data engineering and how it relates to the representation problem in general.