Inducing Diverse Decision Forests with Genetic Programming

Jan Suchý and Jiří Kubalík

Department of Cybernetics, CTU Prague,
Karlovo náměstí 13, 121 35, Praha 2, Czech Republic
suchyj1@fel.cvut.cz, kubalik@labe.felk.cvut.cz

Abstract. This paper presents an algorithm for induction of ensembles of decision trees, also referred to as decision forests. In order to achieve high expressiveness the trees induced are multivariate, with various, possibly user-defined tests in their internal nodes. Strongly typed genetic programming is utilized to evolve structure of the tests. Special attention is given to the problem of diversity of the forest constructed. An approach is proposed, which explicitly encourages the induction algorithm to produce a different tree each run, which represents an alternative description of the data. It is shown that forests constructed this way have significantly reduced classification error even for small forest size, compared to other ensemble methods. Classification accuracy is also compared to other recent methods on several real-world datasets.

1 Introduction

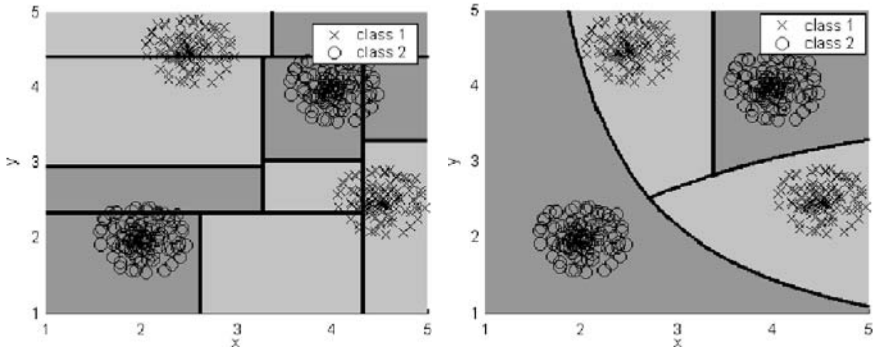
Classification is a task in which machine learning methods are commonly used. With knowledge of attributes $(x_1, x_2, \ldots, x_n) \in X_1 \times X_2 \times \ldots \times X_n$ of an object the task is to assign a correct class k to it, which is unknown, from a set of possible classes K. A program is sought, called classifier, that correctly describes dependence between the class and the attributes. Decision trees [16] are a popular paradigm for modelling such dependencies. This paper presents an algorithm for decision tree induction from data. Emphasis is put on two important aspects of the problem. First, highly expressive, possibly user-defined tests are allowed in decision tree nodes. This way problem specific knowledge can be incorporated into the algorithm. Strongly typed genetic programming is utilized in the hard task of searching for good such tests. Similar approach has been applied in [13]. Second, the algorithm is designed to induce a whole set of diverse trees that can be grouped together in order to improve classification accuracy. Such formation is often called an *ensemble* or, in the case of trees, a *forest* [10]. Decision of a forest is determined by majority vote of its individual trees. For this method to work, it is important that the forest is diverse [12]. In other words, the individual trees should represent alternative descriptions of the data. This is often achieved by introducing small changes in the data that is input to the induction algorithm [1]. In contrast the method presented in this paper explicitly encourages the

M. Keijzer et al. (Eds.): EuroGP 2005, LNCS 3447, pp. 301–310, 2005.

induction algorithm to produce a different decision tree in each run from the same, unchanged data.

2 Decision Trees

Decision trees divide the process of deciding about object's class into a sequence of tests. The tests are organized into a tree structure. In the tree the tests occupy the internal nodes, the edges determine order in which the tests are applied and the leaves represent final decisions: class labels. Classification of an unknown object starts with test in the root node. The edge that corresponds to the outcome of the test determines which test is applied next. This way the object "falls through" the tree down to a leaf which finally assigns a class label to it.



(a) axis-parralel

(b) multivariate

Fig. 1. An example of splitting attribute space with decision trees

Commonly used tests are in the form of conditions $x_i \leq c$ for continuous attributes and $x_i = k$ for discrete attributes. Here c and k are some constants produced by the tree induction algorithm. This kind of tests partition the attribute space with axis-parallel splits, as shown in figure 1(a). It is possible to enrich expressiveness of the trees by allowing more complicated tests in the tree nodes. An example could be $x_1 - \sin x_2 \leq c$. Trees with such tests can be more flexible in partitioning the attribute space, as shown in figure 1(b). They are commonly referred to as *multivariate decision trees* [15]. In this paper the tests are multivariate conditions represented by a tree structure so that they can be straightforwardly searched for with genetic programming.

3 The Tree Induction Algorithm

The algorithm follows the common top-down induction scheme, creating one node at a time. Starting with the root node, the algorithm constructs a test which splits the given set of training examples into two disjoint subsets. The test is constructed so that it maximizes a criterion of optimality, which measures the ability of the test to discriminate examples belonging to different classes. The same is then applied to both subsets obtained with the test. This way the original training set is recursively partitioned until a stopping condition is met.

The individual tests are evolved with genetic programming, using terminal and function sets as specified in section 4. The fitness function coincides with the criterion of test optimality, which is based on measuring the *information gain* [16]. The information gain is the amount of uncertainty (entropy) eliminated by the test from the set it splits. Let M be the training set containing n examples, and s the number of different classes. Let n_i be the number of examples belonging to class i. Then the amount of uncertainty in the set is given by the following equation:

$$H(M) = -\sum_{i=1}^{s} n_i \log_2 \frac{n_i}{n} \ . \tag{1}$$

Further it is defined $H(\emptyset) = 0$ and $0 \log_2 0 = 0$. Every test t splits a set M into M_P and M_N, so that $M = M_P \cup M_N$ and $M_P \cap M_N = \emptyset$. The information gain of test t is computed as:

$$I(t) = H(M) - H(M_P) - H(M_N) \ . \tag{2}$$

While using I as fitness is suitable for single decision trees, it is not appropriate when trees are sequentially induced that are to be combined into a forest. For this scheme a modified criterion is proposed in section 5, which assures that sequentially induced trees differ considerably from each other.

With growing depth of the tree there is an increasing risk of overfitting the data. Therefore at each node a condition is tested which stops the induction if either $\frac{H(M)}{n}$ or the size of M drops below a threshold specified by the user. When this condition is satisfied a leaf node is produced, which assigns to the objects label of the class most frequent in M.

4 The Tests

Higher expressiveness of the trees implies more complex structure of the tests. The tests are expressions in general, which in turn can be easily represented by trees in genetic programming. In this representation the terminal set T consists of attributes and perhaps other entities important for the classification, e.g. random constants. The function set F contains operators, functions and predicates, that express possibly meaningful properties and relations of the attributes. For an expression to be a properly formed test it must evaluate to either `true` or `false`. That is, the function in the root node of the expression must return a boolean value. Now it comes to the serious limitation of standard genetic programming that requires both function and terminal sets to have the closure property. Clearly, one would expect functions that (for example) add or

Table 1. Predefined terminals and functions

Type	**Terminals**	**Functions**
Nominal attributes	x, R	$=$
Ordered attributes	x, R	$=$, $>$
Numeric attributes	x, R	$=$, $>$, $+$, $-$, $*$, sin, (> 0)
Logical values		$\wedge$, $\vee$, $\neg$

multiply the attributes in the tests, but this is not possible due to the necessary closure property. In addition one has to often deal with both numeric and nominal attributes in a single classification task, which at last leads to the same problem. To resolve this problem strongly typed genetic programming [14] was used. It introduces *types* of functions and terminals similar to those found in higher programming languages. It also adds a type checking mechanism to the recombination operators so that only valid trees are constructed.

As said above, the function and terminal sets form a sort of language that is used to describe the data. Ideally the user of the algorithm supplies definitions of needed functions and terminals using his or her knowledge of the problem at hand. As this is not always possible a basic set of functions and terminals is predefined within the system. These allow the algorithm to handle nominal attributes, attributes whose domain is an ordered set, and numerical attributes. Table 1 shows a summary of the predefined functions and terminals. For each attribute type (nominal, ordered and numeric) there is a terminal called x in the table, which represents the value of the attribute, and a terminal called R, which represents an ephemeral random constant [11] from the attribute's domain. For each type there is the function $=$, i.e. comparison of values of that type, which evaluates to a logical value `true` or `false`. For ordered and numeric attributes there is also the relational operator $>$. Only for numeric attributes there are also arithmetic operators $+$,$-$,$*$ and the sine function sin. For logical (boolean) values there are functions $\wedge$, $\vee$ and $\neg$ (conjunction, disjunction and negation, respectively). With these functions it is possible to form tests that combine attributes of different types, as shown in an example in figure 2.

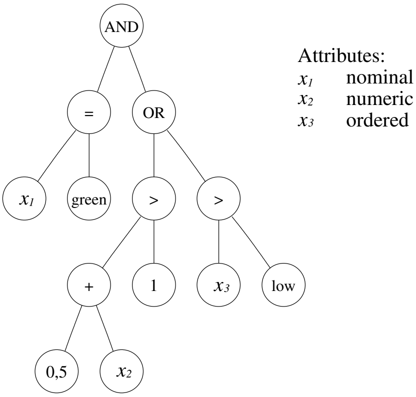


Fig. 2. Example of a test

5 The Forests

Classification accuracy can be often improved by the use of ensemble classifiers. The decision of an ensemble is determined by (weighted) majority vote of the individual classifiers. The underlying concept is that the accuracy of a *diverse* ensemble of classifiers whose accuracy is at least a little better than random guessing is generally better than accuracy of the individual classifiers. The requirement of diversity is important, although precise meaning of the term has not been given clearly yet [12]. A common way to construct different classifiers with an algorithm that outputs a single classifier is to choose different training data for each run of the algorithm. The widely known ensemble methods *bagging* [5] and *AdaBoost* [9] follow this scheme. The algorithm presented here is not deterministic, therefore it produces a different classifier each run even with the same training data. To further improve the resulting ensemble diversity the following approach is proposed.

In the top-down decision tree induction as described in section 3 the root test heavily influences how the other, lower layer tests are formed. Thus by changing the root node test one can substantially alter the whole tree being induced. This is achieved by using a fitness function that causes genetic programming to search for root tests that eliminate uncertainty other than that removed by root tests of previously evolved trees. Tests other than root are evolved with the standard information gain fitness given by equation 2.

It is done in the following way: The root node of the first tree in the forest is evolved with fitness given by equation 2. It should maximize information gain on the training set. The root node of the second tree is evolved with a modified fitness, which favors tests, which eliminate uncertainty not eliminated by the root node of the first tree. Suppose that the first root node test splits the training set M into M_P and M_N, then the fitness used in evolution of the second test is $I(M_P) + I(M_N)$. The root node test of the third tree is evolved so that it eliminates uncertainty not eliminated by either of the root tests of the previously evolved trees. The general formula for fitness of i-th root node test t_i is:

$$J(t_i) = \min_{j \leq i-1} \{I(M_{Pj}) + I(M_{Nj})\} \ , \quad i = 2, \ldots, R \ , \tag{3}$$

where M_{Pj}, M_{Nj} are sets, in which the root node test of j-th tree splits the original training set. Index j runs over all tests evolved before t_i.

6 Experiments

In all experiments in the following subsections, the algorithm was run in the following setup: Only predefined functions and terminals were used, as specified in section 4. For each rule genetic programming was allowed to run for 100 generations with population of 500 individuals. Maximal number of function and terminal symbols together was limited to 12 in all tests. The induction was

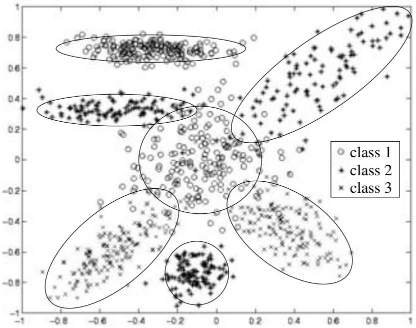


Fig. 3. Synthetic data used for experiment in section 6.1

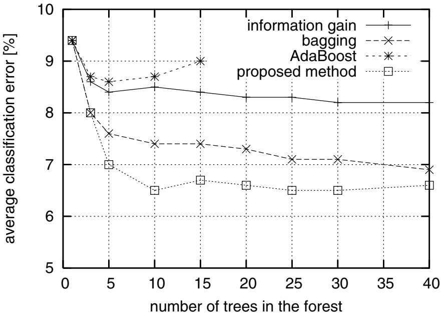


Fig. 4. Classification accuracy of the compared methods

stopped whenever $\frac{H(M)}{n}$ dropped below $0,65$ bits per example or when the size of M dropped below 5% of its original size.

6.1 Comparison to Other Ensemble Methods

In this section the proposed algorithm is compared to two popular ensemble construction methods bagging and AdaBoost. The comparison was carried out on synthetic data, which allowed to illustrate behavior of the proposed method. At the same time the dataset was meant to emulate an easy but typical classification task with two numerical attributes x, y and three classes c_1, c_2, c_3, sampled from a mixture of Gaussian distributions:

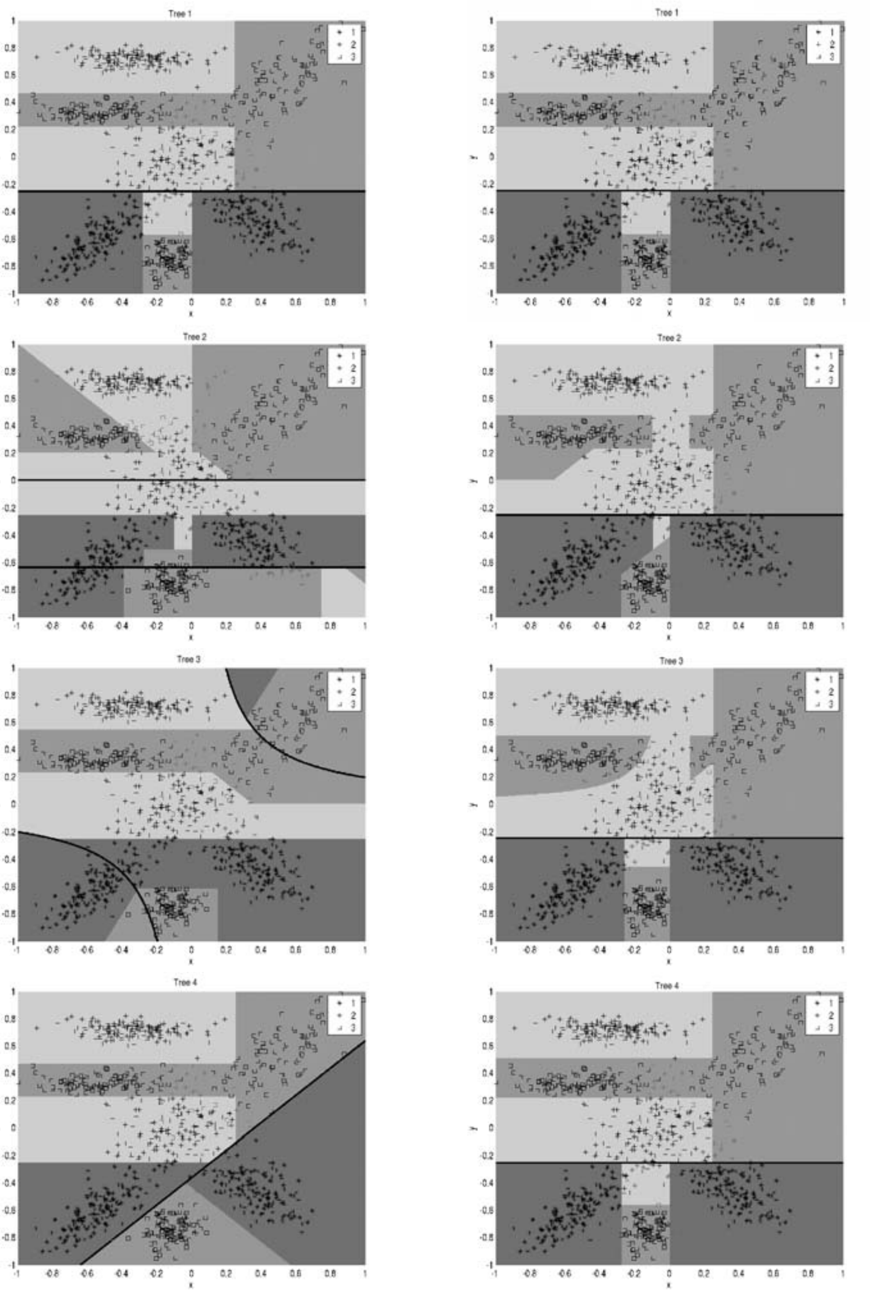
$$p(x, y|c_i) = \sum_j k_{ij} N(\mu_{ij}, \mathbf{\Sigma}_{ij}) \ .$$

The dataset is displayed in figure 3. It is clearly visible that the classes are not perfectly separable. One can achieve classification accuracy approx. 95%.

In the experiment four different methods were used to construct forests of different sizes. For each method and each forest size data consisting of 1000 examples were used for training and testing in 5-fold cross-validation. This procedure was repeated 10 times. All 50 results were then averaged.

First the forests were constructed with pure information gain as fitness, with no changes made in the training data for different trees. This serves as a reference to the other methods. Second, bagging was used as described in [5]. Third, the forests were constructed with AdaBoost.M1 setup exactly as described in [9], section 3. At last, the proposed method was used with fitness given by equation 3 for root node tests and pure information gain for the rest of nodes.

The results are shown in figure 4. The first observation is that when using pure information gain, the induced trees are not as diverse as in the other cases, which leads to lower classification accuracy when compared to the other methods. This is illustrated in figure 5, where trees induced by the proposed method and



(a) the proposed method

(b) pure information gain as fitness

Fig. 5. An example of forests constructed by different methods

Table 2. An overview of the datasets used in the experiment

dataset	# examples	# attributes	# classes	class distribution
Pima	768	8	2	65% / 35%
Heart	920	13	2	44% / 56%
Breast	699	10	2	65% / 35%

those induced using pure information gain can be visually compared. In the figure the black lines represent the root node splits. Another observation is that the proposed method works better compared to bagging when the forest size is small. With growing size of the forest it ceases to have a major advantage.

Although AdaBoost is more sophisticated method than bagging, it suffers from overfitting on this dataset as it tries to specialize on “hard” examples. These, however, are but noise in this case. The effect is especially noticeable for larger forest sizes. In contrast the proposed method searches for alternative descriptions of *all* examples, which makes it more resistant to overfitting.

6.2 Experiments on Real-World Datasets

In this section the results of experiments are presented, which were carried out on datasets from the UCI machine learning repository [3]. Here the proposed algorithm was compared to several other recent methods known from the literature. A short overview of datasets employed in the comparison is in table 2. For each dataset, the classification accuracy was tested in 10-fold cross-validation, and this procedure was 6 times repeated. The 60 resulting values were then averaged.

The results are summarized in table 3, which contains the values of classification error achieved by each of the algorithms. The results of the algorithms have been reported by their authors. A short description of the algorithms with references to papers where the results have been reported follows. It has to be noted that the actual experiment set-up differs for each algorithm and can be looked up in the corresponding paper.

Table 3. Classification error of the compared algorithms in %

Pima	Heart	Breast	algorithms
25,6	20,0		fuzzy decision trees
28,4	22,2		C4.5
25,7		3,3	C4.5 + Adaboost.M2
26,3		4,8	OC1
26,4		5,9	fuzzy rules
27,1	14,8	4,5	GP rules
24,4	**13,8**	3,3	NN
25,0	21,4	3,8	proposed algorithm – single tree
23,6	18,7	**3,1**	proposed algorithm – forest, 11 trees

Fuzzy decision trees. An evolutionary method for induction of fuzzy decision trees based on clustering [8].

C4.5. Well-known decision tree induction algorithm by J. R. Quinlan [16]. The reported results are taken from [8].

C4.5 + Adaboost.M2. C4.5 combined with the AdaBoost for ensemble construction [9].

OC1. A well-known algorithm for induction of oblique decision trees. In oblique decision trees the tests are in the form of inequalities: $a_1x_1 + a_2x_2 + ... + a_nx_n \leq c$. The reported results are taken from [6].

Fuzzy rules. An expert system based on fuzzy rules constructed with genetic programming. [2].

GP rules. A rule-based system, evolves IF-THEN rules with genetic programming. [7].

NN. A neural network (multi layer perceptron) learned by the back-propagation method. [4].

The results indicate that the proposed algorithm is competitive with the other algorithms. The error reduction achieved by the use of forests is significant even for small sized forests when compared to ensemble sizes used in [5].

7 Conclusions and Future Work

An algorithm was introduced, that induces decision trees with possibly highly expressive tests in their nodes. It allows the user of the algorithm to choose or define "building blocks" of the decision tree tests, appropriate for the problem at hand. As this feature is an advantage theoretically, its practical benefits for real-world problems is still to be investigated in the future. Strongly typed genetic programming is utilized for searching for the tests, which provides a straightforward way to construct tests with different, possibly complex structure.

A special fitness function is proposed, which allows the algorithm to sequentially induce a diverse group of trees, which can be then used as a decision forest. Decision forests constructed this way have considerably higher classification accuracy than the individual trees even for small forest sizes, compared to the general method Bagging. On the other hand they are not as susceptible to overfitting as the AdaBoost algorithm, which is also known to be able to improve classification accuracy even for small ensemble sizes.

Similarly to many other evolutionary algorithms the proposed algorithm has a number of variable parameters. The most important of them are the stopping condition of the tree induction, the maximum allowed size of the evolved tests, and the function and terminal sets used. The parameters used for the experiments conducted in this paper can serve as reasonable default values. As a rule, one should allow only simple tests to be evolved (i.e. tests consisting only of a small number of functions and terminals), when the training sample is small. Otherwise the algorithm is likely to overfit the data. Something similar holds for the stopping condition of the induction algorithm: For small training samples one should stop the induction earlier to avoid overfitting.

The future work will be concentrated on the problem of searching for simple descriptions of data, as they are likely to perform better than complex ones. The authors' observations suggest this, as well as several other studies [15].

References

1. Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1-2):105–139, 1999.
2. P. J. Bentley. Evolving fuzzy detectives: An investigation into the evolution of fuzzy rules. In *Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference*, pages 38–47, Orlando, Florida, USA, 1999.
3. C. L. Blake and C. J. Merz. UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`, 1998.
4. M. Brameier and W. Banzhaf. A comparison of linear genetic programming and neural networks in medical data mining. *IEEE Transactions on Evolutionary Computation*, 5(1):17–26, February 2001.
5. L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
6. E. Cant-Paz and C. Kamath. Inducing oblique decision trees with evolutionary algorithms. *IEEE Transactions on Evolutionary Computing*, 7(1):56–68, 2003.
7. I. De Falco, A. Della Cioppa, and E. Tarantino. Discovering interesting classification rules with genetic programming. *Applied Soft Computing*, 1(4F):257–269, May 2001.
8. J. Eggermont. Evolving fuzzy decision trees with genetic programming and clustering. In *Proceedings of the 4th European Conference on Genetic Programming, EuroGP 2002*, volume 2278, pages 71–82, Kinsale, Ireland, 3-5 2002. Springer-Verlag.
9. Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proc. 13th International Conference on Machine Learning*, pages 148–146. Morgan Kaufmann, 1996.
10. Tin Kam Ho. C4.5 decision forests. In *Proceedings of the 14th International Conference on Pattern Recognition-Volume 1*, page 545. IEEE Computer Society, 1998.
11. J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA, 1992.
12. L. Kuncheva and C. Whitaker. Measures of diversity in classifier ensembles, 2000.
13. Robert E. Marmelstein and Gary B. Lamont. Pattern classification using a hybrid genetic program decision tree approach. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 223–231, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.
14. D. J. Montana. Strongly typed genetic programming. BBN Technical Report #7866, Bolt Beranek and Newman, Inc., 10 Moulton Street, Cambridge, MA 02138, USA, 7 May 1993.
15. S. K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2(4):345–389, 1998.
16. J. R. Quinlan. *C4.5: programs for machine learning.* Morgan Kaufmann Publishers Inc., 1993.