A Genetic Programming Framework for Two Data Mining Tasks: Classification and Generalized Rule Induction.

Alex A. Freitas
University of Essex
Dept. of Computer Science
Colchester, CO4 3SQ, UK
freial@essex.ac.uk

ABSTRACT

This paper proposes a genetic programming (GP) framework for two major data mining tasks, namely classification and generalized rule induction. The framework emphasizes the integration between a GP algorithm and relational database systems. In particular, the fitness of individuals is computed by submitting SQL queries to a (parallel) database server. Some advantages of this integration from a data mining viewpoint are scalability, data-privacy control and automatic parallelization. The paper also proposes some genetic operators tailored for the two above data mining tasks.

1. Introduction

Data Mining (DM) consists of the extraction of interesting, novel knowledge from real-world databases [Fayyad et al. 96]. DM is an interdisciplinary subject, whose core lies at the intersection of machine learning and databases. Four desirable characteristics of a DM system are: (1) the discovery of comprehensible knowledge, typically expressed by high-level rules; (2) integration with databases [Han et al. 96], [Imielinski et al. 96]; (3) a high degree of autonomy, necessary to discover knowledge previously unknown by the user [Zytkow 93]; (4) the efficiency of the knowledge discovery process, necessary to cope with large databases.

This paper proposes a genetic programming (GP) framework for DM that addresses, to some extent, all of these four issues. GP individuals encode database queries that correspond to high-level rules, used to predict the value of some attribute. The fitness of GP individuals is computed by submitting SQL queries to a database server, so achieving a tight integration between GP and relational databases. The ability of GP to search many parts of the program (database query) space in parallel and the robustness of GP render the DM system more autonomous, minimizing the need for domain knowledge to guide the search. Finally, efficiency can be significantly improved by using a parallel SQL server to evaluate the fitness of an individual. Note that the use of this kind of database server does not require any modification in the GP framework - i.e. the SQL queries generated by the GP are automatically parallelized by the SQL server.

There are several kinds of DM tasks [Fayyad et al. 96], depending mainly on the application domain and the user interest. Some of the major DM tasks frequently discussed in the literature are listed in Figure 1,where the tasks are roughly ordered according to their amount of inductive inference and computational complexity. Deviation detection (i.e. detection of unexpected values) and summarization are in general the simplest kinds of DM task, since the amount of inductive inference is usually very limited and there is no prediction of unknown attribute values (the emphasis is on description, rather than on prediction). On the other extreme, clustering is in general the most complex kind of DM task, since it involves a kind of "many-to-many" prediction of attribute values, i.e. any attribute can be used to determine clusters of tuples and to predict the values of other attributes. This paper proposes a GP framework for the DM tasks of classification and generalized rule induction. The former is the most studied task in the literature, and the latter is a natural generalization of the former. Finally, this paper also proposes some genetic operators tailored for the tasks of classification and generalized rule induction.

complex
↕
- clustering
- generalized rule induction
- classification
- summarization
- deviation detection

simple

Figure 1 Some DM tasks ordered by complexity.

This paper is organized as follows. Section 2 discusses related work. Section 3 discusses the phenotype of GP individuals in our framework. Sections 4 and 5 discuss genotype-related issues for the tasks of classification and generalized rule induction, respectively. Section 6 mentions some advantages of the integration between GP and relational databases. Section 7 presents a discussion and summary.

2. Related Work

PLEASE [Knight & Sen 95] is a Genetic Algorithm (GA) where each individual encodes a set of class prototypes, and the learned prototypes are then used for classification - i.e. a tuple is assigned the class of the nearest prototype, according to a given distance metric. GIL uses several generalization/specialization operators proposed by [Michalski 83] to extend the genetic operators of conventional GA, creating a knowledge-intensive GA for classification tasks [Janikow 93].

REGAL learns first-order-logic (FOL) class descriptions [Neri & Giordana 95]. However, it assumes that the user provides a kind of template of the logical formula to be learned. This reduces the autonomy of the system, which is a serious drawback in the context of DM. SIAO1 also learns FOL class descriptions, but it does not need a user-specified formula template [Augier et al. 95]. Instead, it adapts the technique of generalizing a seed example [Michalski 83] to learn FOL classification rules. From a DM viewpoint, the main drawback of SIAO1 is that this kind of generalization technique is quite computationally expensive, making it difficult to apply SIAO1 to large databases.

Masson is a Genetic Programming system designed for a new kind of knowledge discovery task, here referred to as commonality description [Ryu & Eick 96]. This task consists of searching for a database query that describes the commonalities (e.g. common attribute values) of a user-specified tuple set. From the viewpoint of inductive learning, this task is significantly simpler than the tasks of classification and generalized rule induction - the two tasks addressed in this paper. (With respect to Figure 1, commonality description would be somewhere between summarization and classification.) The reason is that in commonality description the system knows a priori which tuples the description to be discovered should cover and which it should not, and it has access to all task-relevant tuples. There is no separate test set, where the system has to predict the class of unseen tuples, which is the ultimate challenge of classification and generalized rule induction. However, Masson was an important inspiration for the framework proposed in this paper.

GP-Knn is a hybrid Genetic-Programming/K-nearest-neighbors algorithm, where the GP searches for a good set of attribute weights for the K-nn [Raymer et al. 96]. Each individual in the GP population is represented by m trees, where m is the number of attributes. Hence, each tree encodes a function for the weight of a given attribute. Once attribute weights are determined by the GP, the classification task itself (the core of the DM process) is performed by the K-nn. Hence, there is no discovery of explicit, high-level classification rules. This is a disadvantage in the context of DM, where the comprehensibility of the discovered knowledge is crucial [Fayyad et al. 96].

To summarize, several GAs have been proposed for classification tasks, including PLEASE, GIL, REGAL and SIAO1. However, these systems are mainly machine learning ones, ignoring the important DM issue of integration with databases. Two GPs proposed for DM are GP-Knn and Masson. The former was designed for classification tasks. However, from a DM viewpoint, it has important limitations concerning discovered-knowledge comprehensibility and integration with databases. Masson was designed for the commonality description task in object-oriented databases. The framework described in this paper is inspired in Masson, but it addresses the more challenging (from an inductive learning viewpoint) DM tasks of classification and generalized rule induction, in relational (rather than object-oriented) database systems.

3. The Phenotype of Genetic Programming Individuals for DM

In this Section we discuss the phenotype of individuals (i.e. the meaning of the genetic material for the user) in our Genetic Programming (GP) framework for DM. This phenotype is the same for both the classification and the generalized rule induction tasks. However, the genotype of the GP individuals are somewhat different in these two tasks. Actually, the genotype of individuals in the generalized rule induction task is an extension of the genotype in the classification task. Hence, the discussion of the genotype of individuals in our framework will be done in the next two Sections, which discuss each of these tasks separately.

Turning to phenotype issues, in our framework the GP individuals encode SQL queries following the query template shown in Figure 2. To simplify our discussion, the specification of that query assumes, without loss of generality, that the data to be mined is stored in a single relation - called the Mine relation.

Each query (individual) generated by the GP is evaluated against the database, in order to compute the kind of contingency table shown in Figure 3. This is a 2 x n matrix extended with totals of rows and columns, where n is the number of values of the goal attribute (or classes, in the case of the classification task). Each of the cells in the first row of this matrix, denoted C_{1j} - j=1,...,n (where C stands for a Count value) - contains the number of tuples satisfying the Tuple-Set Descriptor (TSD) and having goal-attribute value G_j. On the other hand, each of the cells in the second row of this matrix, denoted C_{2j} - j=1,...,n - contains the number of tuples *not* satisfying the TSD and having the goal-attribute value G_j. The row and column totals are denoted respectively by C_{i+}, i=1,2, and C_{+j}, j=1,...,n. The total number of tuples being mined is denoted by C_{++}.

These Count values are used to compute the fitness of a GP individual, by taking into account the number of correct and incorrect predictions made by a rule. Note that the fitness of an individual can also consider other factors, such as measures of syntactic rule simplicity and rule similarity. However, these extra factors do not require access to the database, and their computational cost is dominated by the time to compute the Count values shown in Figure 3. Our goal in designing the query of Figure 2 was to encapsulate all the time-consuming fitness-computation procedures requiring access to the database into a single database query, so that this query can be optimized by a high-performance database server. In particular, this query can be speeded up by automatically exploiting data parallelism on parallel SQL servers. (For empirical evidence that parallel SQL servers can be effectively used to speed up queries like the one shown in Figure 2, the reader is referred to [Freitas & Lavington 96].)

```
Select Goal-Attribute, Count(*)
From   Mine-Relation
Where  Tuple-Set-Descriptor
Group  By Goal-Attribute
```

Figure 2 SQL query underlying the phenotype of GP individuals.

	G_1 . . . . G_n	Total
satisfy TSD	C_{11} . . . . C_{1n}	C_{1+}
not satisfy TSD	C_{21} . . . . C_{2n}	C_{2+}
Total	C_{+1} . . . . C_{+n}	C_{++}

Figure 3 Contingency table produced by evaluating the query of Figure 2.

The query shown in Figure 2 corresponds to a rule of the form:

if (Tuple-Set Descriptor) then (Goal-Attribute = V_g) ,

where the TSD is a logical formula containing attribute values and comparison operators (see Section 4 for details) and V_g is a value belonging to the domain of the goal attribute. To give a simple example, consider a company's database with three attributes, namely Age, Job-Status and Training, where the latter is the goal attribute. Assume that Training can take on two values, "yes" or "no", and assume that the TSD is: (Age < 25 and Job-Status = full-time). Then a rule predicting that an employee needs training would be:

if (Age < 25 and Job-Status = "full-time")
then (Training = "yes").

As discussed above, the TSD of the query shown in Figure 2 corresponds to the antecedent of a rule encoded by a GP individual. However, we still have to decide which value of the goal attribute should be predicted by the consequent of the rule (since all the values of the goal attribute appear in the table of Figure 3). We choose the goal attribute value with the largest Count value in the first row of the table of Figure 3, i.e. we choose the goal attribute value whose Count value C_{max} is given by

$$C_{max} = \max_{j=1}^{n} C_{1j} .$$

It is interesting to note that in our framework the mapping between an individual's genotype and its ultimate phenotype (a rule) occurs in two phases. First, the individual's genotype (described in Section 4) is mapped into an SQL query of the form shown in Figure 2. However, the actual mapping between the generated SQL query and the rule only occurs in a second phase, where the SQL query is evaluated against the database and the value of the goal attribute to be predicted by the consequent of the rule is determined.

This feature of our framework can be regarded as a form of lazy learning, where the actual generation of the full rule is *delayed until more information is available* to produce a better rule[1]. In passing we remark that the basic idea of lazy learning has been applied to other (nongenetic-based) DM paradigms as well - see e.g. [Friedman et al. 96].

Moreover, this approach also leads to gains in computational efficiency, since by evaluating a single database query we can efficiently compute the fitness associated with several possible rules. Hence, we avoid the repeated client/server communication overhead associated with the separate evaluation of those possible rules.

4. Genetic Programming for the Task of Classification

The classification task consists of predicting the value of a user-specified goal attribute given the values of other attributes, called predicting attributes. The goal attribute's domain consists of a small set of discrete values, called classes.

Note that the goal attribute is fixed during the execution of the DM algorithm. Hence, the genotype of a GP individual consists only of the tree representing the Tuple-Set-Descriptor (TSD) of the query shown in Figure 2. Note also that, although the goal attribute does not need to be represented in the genotype of an individual, it is still specified as part of the query submitted to the database, to compute an individual's fitness. The terminal set for the tree encoding a TSD consists of the names of the predicting attributes and of the values in their corresponding domains. The function set consists of the logical connectives {AND, OR, NOT} and the comparison operators {>, ≥, <, ≤, =, ≠}.

1 "Delay is preferable to error." - Thomas Jefferson (1743-1826).

Since this encoding follows the conventional GP paradigm [Koza 92], a conventional crossover operator can be used.

To give a simple example of the encoding of a TSD, consider the following TSD:

((A₁ > V₁) AND (A₂ = V₂)) OR (A₃ < A₄),

where A_i, i=1,...,4, denotes the i-th attribute and V_1 and V_2 denote values of the domain of the attributes A_1 and A_2 respectively. This TSD would be encoded as the query tree:

(OR (AND (> A_1 V_1) (A_2 = V_2)) (< A_3 A_4)).

4.1. Two-Class versus Multiple-Class Classification Problems

We now consider separately two types of classification problems, namely two-class problems and multiple-class ones. In our framework this distinction is important for two reasons. First, after running the GP, the result designation method is somewhat different for two-class and multiple-class problems. Second, the multiple-class problem suggests the use of some kind of niching method. These issues are discussed in the following.

In the case of two-class problems, which are more common than multiple-class ones [Weiss & Kulikowski 91], it is enough to designate the best individual as the result - i.e. the classification rule that will be returned to the user. The reason is simple. The selected individual predicts a given class, say class-1, to all tuples satisfying its TSD tree. Hence, all tuples that do not satisfy the individual's TSD tree are implicitly classified as belonging to the other class, say class-2 (called the default class). There is no need to discover an explicit rule for predicting the default class.

In the case of multiple-class problems, where the number of classes is greater than two, we now need to select NC - 1 (where NC is the number of classes) individuals as the result, where each individual predicts a different class for its corresponding TSD tree. (Note that this reduces to select 1 class in the case of the two-class problem.) The class for which no individual is selected becomes the default class.

Since we are interested in selecting several individuals as the result, it might be useful to use some niching method [Mahfoud 95], [Miller & Shaw 95] to incentivate the evolution of individuals with different predicted classes. Note that niching is not necessary in the two-class problem, where a single best individual is selected.

One of the difficulties in designing a niching method is that typically this kind of method is used in the context of multimodal function optimization, where we do not know a priori the number and location of peaks (niches) in the fitness landscape. However, in our data mining problem we know, a priori, the number of niches that should be formed, which is simply the number of classes. This allows us to use a better informed niching method, which takes this information into account. For instance, it is desirable to guarantee that, for all the classes, there will be at least one individual predicting that class. This can be done by enforcing this condition when creating the initial population and then by using a niching-oriented elitist reproduction scheme that passes to the next generation's population an unaltered copy of the best individual for each class. Regarding the set of individuals with the same predicted class as a niche, this scheme is equivalent to an elitist reproduction within each niche.

5. Genetic Programming for the Task of Generalized Rule Induction

In the generalized rule induction task, similarly to classification, the goal is to discover rules that predict the value of a goal attribute, given the value of other attributes. However, unlike classification, the goal attribute can be any attribute not occurring in the antecedent of the rule. Hence, a GP individual in our framework consists of two parts: (a) a query tree encoding the Tuple-Set Descriptor (TSD) of the query shown in Figure 2; and (b) a gene encoding the goal attribute. The function and terminal sets for the TSD tree are similar to the ones specified for the classification task. A slight difference is that there is no goal attribute specified in advance. Hence, the terminal set includes all attributes and their values, while in the classification task the terminal set included only the predicting attributes and their values.

The extra gene encoding the goal attribute (unnecessary in the case of the classification task) introduces an opportunity for a mutation operator. We propose a mutation operator that replaces the goal attribute allele with another attribute, taken from the set of attributes not occurring in the TSD tree. Let this set be denoted by PGA (standing for "possible goal attributes"). The new attribute can be chosen in at least three ways.

First, we can randomly select an attribute from PGA, with a uniform probability distribution. Formally, the probability of selecting a given attribute A_i, i=1,...,m, where m is the cardinality of PGA, is $Prob(A_i) = 1/m$. We call this random mutation. The main advantage of this form of mutation is that its computation is very cheap, taking O(1).

Second, we can select an attribute from PGA with a probability inversely proportional to the occurrence of that attribute in the goal-attribute gene of other individuals of the current population. Let NI denote the number of individuals in the current population; and let n_i, i=1,...,m, denote the number of individuals of the current population whose goal-attribute gene encodes the attribute A_i. Thus, the probability of selecting a given attribute A_i is given by the formulae $Prob(A_i) = (NI - n_i)/NI$. The motivation for this operator is to increase the diversity of the goal attribute gene in the population. We call this diversity-oriented mutation. The computation of this operator is also relatively cheap, taking O(NI), since all the $Prob(A_i)$, i=1,...,NI, can be computed by a single pass through the current population. Note that in real-world databases NI << N, where N is the number of tuples being mined, so that the cost to implement the diversity-oriented mutation operator tends to be much smaller than the cost to evaluate the fitness of individuals.

As a third form of mutation, we can select the "best" attribute in PGA, i.e. the attribute that maximizes the fitness of the individual. We call this fitness-oriented mutation. This operator can significantly improve the fitness of an individual, but it has one drawback. Its computation is very computationally expensive. For each attribute in PGA a database query must be executed, to evaluate the individual's fitness when that attribute replaces the previous goal-attribute allele. This takes O(q*m), where q is the number of tuples selected by a query and m is the cardinality of PGA (i.e. the number of possible goal attributes).

As a final remark, our GP framework for generalized rule induction presents a problem that does not occur in the case of classification. A GP individual may represent an invalid rule, if the goal attribute specified in the rule occurs in the TSD tree. Although it is trivial to avoid this when generating the initial population, this situation can occur as a result of crossover in the TSD tree. The traditional approach to cope with this problem is to ignore the crossover when this situation happens. As an alternative solution, we suggest to use our previously-proposed mutation operator (in any of its three suggested forms) as a repair operator, since it effectively replaces the current goal attribute allele with an attribute that does not occur in the TSD tree. This avoids the need for a special repair operator and contributes for the simplicity and uniformity of our framework.

6. Advantages of the Integration Between DM and Relational DBMS

From a DM viewpoint, the tight integration between the DM algorithm (the GP, in this paper) and a Relational Database Management System (RDBMS) achieved in our framework has several advantages - see [Freitas 97] for details.

(a) Improved scalability - Our framework capitalizes on database and data warehouse technology, allowing DM systems to scale up to very large databases.

(b) Data re-use and minimization of data redundancy - In our framework the data being mined is kept stored in the RDBMS during all the DM process. Thus, we get the benefits of data re-use and minimization of data redundancy, which are crucial in large-scale DM applications [Brown et al. 95].

(c) Improved security and data-privacy control - Our integrated framework allows the DM system to benefit from the usual security and concurrency control mechanisms offered by RDBMSs. This is important due to the fact that DM applications, by their very nature, are cause for concern about security and data privacy [O'Leare 95]. Moreover, note that in our framework the GP algorithm analyzes only statistics or aggregated data (the Count values of Figure 3) retrieved from the database server, rather than the original data. Hence, our approach has the benefit of anonymization, i.e. it avoids the risk of re-identifying individual cases [Klosgen 95].

(d) Automatic Exploitation of Data Parallelism on Parallel SQL Servers - Despite its limitations, SQL is a declarative-style RDBMS query language. Hence, the SQL database queries (requests for fitness evaluation) generated in our framework can be efficiently executed by having all processors of a Parallel SQL Server accessing the data in parallel [Hasan et al. 96]. This parallelization is *automatically* done by the Parallel SQL Server, i.e. it is entirely transparent for the GP. We stress that the vast majority of large data warehouses are implemented on Parallel SQL Servers [Hedberg 95], [IBC 95].

(e) Portability Across a Wide Range of (Parallel) RDBMS - Once the database queries produced by our framework are expressed in standard SQL, they can be executed in a wide range of commercially-available RDBMS. Furthermore, virtually all commercially-available Parallel SQL Servers offer an SQL interface [IBC 95]. This extends the benefit of portability to a wide range of parallel computer architectures, including both shared-memory and distributed-memory systems.

7. Discussion and Summary

We proposed a genetic programming (GP) framework for two major data mining tasks, namely classification and generalized rule induction. The framework specifies a GP phenotype and a GP genotype for these two tasks. The phenotype is based on relational algebraic operations, expressed by an SQL query. This leads to a tight integration between the data mining algorithm (the GP) and relational database systems.

Concerning data mining issues, this integration leads to minimization of data redundancy and improved scalability, data-privacy control and portability (see the previous Section). Furthermore, this integration can be used to significantly speed up the execution of the GP, since the evaluation of the fitness of an individual can be done by submitting SQL queries to a parallel SQL server. This is important, since the vast majority of current data warehouses are implemented on parallel SQL servers.

Turning to GP issues, our phenotype involves a form of lazy learning, where the actual generation of a full rule is delayed until more information is available to produce a better rule. Moreover, we discussed some GP-search issues separately for the tasks of classification and generalized rule induction.

In the case of the classification task we discussed the influence of the type of classification (two-class versus multiple-class problems) on the GP search. In particular, we discussed the need for a niching scheme in the case of multiple-class problems and pointed out an important difference between niching in multimodal function optimization and niching in multiple-class classification problems. As a simple example of how to exploit this difference to improve the GP, we suggested a niching-oriented elitist reproduction scheme.

In the case of the task of generalized rule induction, we proposed three forms of mutation to modify the allele of the goal attribute gene, namely random mutation, diversity-oriented mutation (aiming at increasing the diversity of the goal attribute gene in the population) and fitness-oriented mutation (aiming at maximizing the fitness of an individual). We also analyzed the computational complexity of each of these forms of mutation. Finally, we also suggested to use our mutation operators as repair operators, to transform invalid (lethal) offspring produced by crossover into valid offspring.

We believe that the proposed framework is a promising approach for highly autonomous data mining, particularly concerning the task of generalized rule induction. This task tends to have a huge search space associated with it, making it difficult to use conventional rule induction methods. However, there is a caveat, not only for GP but also for other data mining paradigms. The task of generalized rule induction usually requires some constraints to limit the search, otherwise many trivial relationships among attributes can be discovered. This issue is left for future work.

In addition, future work will involve the evaluation of algorithms developed under the proposed framework on real-world databases.

Acknowledgments

The author was financially supported by the Brazilian Government's CNPq, grant number 200384/93-7.

Bibliography

[Augier et al. 95] S. Augier, G. Venturini and Y. Kodratoff. Learning first order logic rules with a genetic algorithm. *Proc. 1st Int. Conf. Knowledge Discovery & Data Mining*, 21-26. AAAI Press, 1995.

[Brown et al. 95] M. Brown, I. Watson and N. Filer. Separating the cases from the data: towards more flexible case-based reasoning. *Proc. 1st Int. Conf. On Case-Based Reasoning (ICCBR-95). LNAI 1010*, 157-168. 1995.

[Fayyad et al. 96] U.M. Fayyad, G. Piatetsky-Shapiro and P. Smyth. From data mining to knowledge discovery: an overview. In: U.M. Fayyad, et al. (Eds.) *Advances in Knowledge Discovery and Data Mining*, 1-34. AAAI/MIT Press. 1996.

[Freitas 97] A.A. Freitas. Generic, set-oriented primitives to support data-parallel knowledge discovery in relational database systems. *Ph.D. Thesis.* University of Essex, UK. 1997.

[Freitas & Lavington 96] A.A. Freitas and S.H. Lavington. Using SQL primitives and parallel DB servers to speed up knowledge discovery in large relational databases. R. Trappl. (Ed.) *Cybernetics and Systems'96: Proc. 13th European Meeting on Cybernetics and Systems Research*, 955-960. Vienna, 1996.

[Friedman et al. 96] J.H. Friedman, R. Kohavi and Y. Yun. Lazy decision trees. *Proc. 1996 Nat. Conf. AAAI*, 1996.

[Han et al. 96] J. Han, Y. Fu, W. Wang, J. Chiang, W. Gong, K. Koperski, D. Li, Y. Lu, A. Rajan, N. Stefanovic, B. Xia and O.R. Zaiane. DBMiner: A system for mining knowledge in large relational databases. *Proc. 2nd Int. Conf. Knowledge Discovery & Data Mining*, 250-255. AAAI Press, 1996.

[Hasan et al. 96] W. Hasan, D. Florescu and P. Valduriez. Open issues in parallel query optimization. *SIGMOD Record* 25(3), Sep 1996.

[Hedberg 95] S.R. Hedberg. Parallelism speeds data mining. (Industrial Spotlight) *IEEE Parallel & Distributed Technology*, Winter 1995, 3-6.

[IBC 95] IBC Ltd. *Proc. Conf. Commercial Parallel Processing*. London, Nov./95. (IBC Technical Services Ltd., 57-61 Mortimer St., London, W1N 7TD.)

[Imielinski et al. 96] T. Imielinski, A. Virmani and A. Abdulghani. DataMine: application programming interface and query language for database mining. *Proc. 2nd Int. Conf. Knowledge Discovery & Data Mining*, 256-261. AAAI Press, 1996.

[Janikow 93] C.Z. Janikow. A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 13, 1993, 189-228.

[Klosgen 95] W. Klosgen. Anonymization techniques for knowledge discovery in databases. *Proc. 1st Int. Conf. Knowledge Discovery & Data Mining*, 186-191. AAAI Press, 1995.

[Knight & Sen 95] L. Knight and S. Sen. PLEASE: a prototype learning system using genetic algorithms. *Proc. 6th Int. Conf. Genetic Algorithms*, 429-435. 1995.

[Koza 92] J.R. Koza. *Genetic Programming: on the programming of computers by means of natural selection.* MIT, 1992.

[Mahfoud 95] S.W. Mahfoud. A comparison of parallel and sequential niching methods. *Proc. 6th Int. Conf. Genetic Algorithms*, 136-143. 1995.

[Michalski 83] R.W. Michalski. A theory and methodology of inductive learning. *Artif. Intellig.* 20, 1983, 111-161.

[Miller & Shaw 95] B.L. Miller and M.J. Shaw. Genetic algorithms with dynamic niche sharing for multimodal function optimization. *IlliGAL Report No. 95010*. 1995.

[Neri & Giordana 95] F. Neri and A. Giordana. A parallel genetic algorithm for concept learning. *Proc. 6th Int. Conf. Genetic Algorithms*, 436-443. 1995.

[O'Leary 95] D.E. O'Leary. Some privacy issues in knowledge discovery: the OECD Personal Privacy Guidelines. *IEEE Expert*, 10(2), Apr 1995, 48-52.

[Raymer et al. 96] M.L. Raymer, W.F. Punch, E.D. Goodman and L.A. Kuhn. Genetic programming for improved data mining -- application to the biochemistry of protein interactions. *Genetic Programming 1996: Proc. 1st Annual Conf.*, 375-380. July 1996.

[Ryu & Eick 96] T.-W. Ryu and C.F. Eick. Deriving queries from results using genetic programming. *Proc. 2nd Int. Conf. Knowledge Discovery & Data Mining*, 303-306. AAAI Press, 1996.

[Weiss & Kulikowski 91] S.M. Weiss and C.A. Kulikowski. *Computer Systems that Learn*. Morgan Kaufmann, 1991.

[Zytkow 93] J.M. Zytkow. Cognitive autonomy in machine discovery. *Machine Learning*, 12(1-3), Aug 1993, 7-16.