On the Suitability of Genetic-Based Algorithms for Data Mining*

Sunil Choenni

Nat. Aerospace Lab., NLR, P.O. Box 90502, 1006 BM Amsterdam, The Netherlands,
and
Univ. of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
email: choenni@nlr.nl

Abstract. Data mining has as goal to extract knowledge from large databases. A database may be considered as a search space consisting of an enormous number of elements, and a mining algorithm as a search strategy. In general, an exhaustive search of the space is infeasible. Therefore, efficient search strategies are of vital importance. Search strategies on genetic-based algorithms have been applied successfully in a wide range of applications. We focus on the suitability of genetic-based algorithms for data mining. We discuss the design and implementation of a genetic-based algorithm for data mining and illustrate its potentials.

1 Introduction

Research and development in data mining evolves in several directions, such as association rules, time series, and classification. The latter field has our attention. We have developed an algorithm to classify tuples in groups and to derive rules from these groups. In our view, a user formulates a mining question and the algorithm selects the group(s) that satisfy this question. For example, in an insurance environment, a question may be to identify persons with (more than average) chances of causing an accident. Then, the algorithm searches for the (group) profiles of these persons.

In general, the search spaces that should be inspected in order to find answers on mining questions are very large, making exhaustive search infeasible. So, heuristic search strategies are of vital importance to data mining. Genetic algorithms, which are heuristic search strategies, have been successfully used in a wide range of applications. A genetic algorithm is capable of exploring different parts of a search space [10].

In this paper, we discuss the applicability of a genetic-based algorithm to the search process in data mining. We show how a genetic algorithm can be suited for data mining problems. In our approach, a search space consists of expressions. An expression is a conjunction of predicates and each predicate is defined on a database attribute. Initially, a random number of expressions, called initial population, is selected. Then, the initial population is manipulated

* This research has been sponsored by the Dutch Ministry of Defense.

by applying a number of operations. The best individuals are selected to form the next generation and the manipulation process is repeated until no significant improvement of the population can be observed.

In general, data mining algorithms require a technique that partitions the domain values of an attribute in a limited set of ranges, simply because considering all possible ranges of domain values is infeasible. Suppose that we have an attribute *age* which has a domain between 18 to 65, and an expression of the form *age* **in** $[v_i, v_k]$, in which v_i and v_k are values from the domain of *age*, defining a range of values. The problem is how to choose the values for v_i and v_k. As illustrated in [11], this is in general an NP-complete problem. Our solution to this problem is based on a suitable choice of the mutation operator (see Section 3.3). Furthermore, we have chosen a representation for individuals that seamlessly fits in the field of databases. The same holds for the manipulation operators and the function to rank individuals (fitness function). The fitness function discussed in this paper is close to our intuition and gives rise to a speed up of the optimization process. Based on our approach, we have implemented a (prototype) tool for data mining, and have performed a preliminary evaluation. The results will be presented in this paper.

A genetic approach has been proposed in [2] to learn first order logic rules and in [7] a framework is proposed for data mining based on genetic programming. However, the authors neither come up with a implementation nor with experiments. The effort in [2] is focussed towards machine learning, and the important data mining issue of integration with databases is superficially discussed. The effort in [7] describes a framework for data mining based on genetic programming, and stresses on the integration of genetic programming and databases. However, an elaborated approach to implement and evaluate the framework is not presented. Other related research has been reported in [1, 8, 9]. While in [8, 9] variants of a hill climber are used to identify the group(s) of tuples satisfying a mining question, the approach in [1] is based on decision trees. However, the problem of partitioning attribute values has not been discussed in these efforts. We note that a genetic-based algorithm has, by nature, a better chance to escape from a local optimum than a hill climber.

The remainder of this paper is organized as follows. In Section 2, we outline some preliminaries and problem limitations. In Section 3, we identify the issues that play a role in genetic-based algorithms and adapt them in a data mining context. In Section 4, we point out a number of rules that may speed up the search process of a genetic-based algorithm. Section 5 is devoted to an overall algorithm for data mining. In Section 6, we discuss the implementation of the algorithm and some preliminary results. Finally, Section 7 contains conclusions and further work.

2 Preliminaries & problem limitations

In the following, a database consists of a universal relation [6]. The relation is defined over some independent single valued attributes, such as att_1,att_2,..., att_n, and is a subset of the Cartesian product dom(att_1) $\times$ dom(att_2) $\times ... \times$ dom(att_n), in which dom(att_j) is the set of values that can be assumed by attribute att_j. A tuple is an ordered list of attribute values to which a unique identifier (tid) is associated. So, we do not allow missing attribute values. Furthermore, we assume that the content of the database remains the same during the mining process.

An expression is used to derive a relation, and is defined as a conjunction of predicates over some attributes. The length of an expression is the number of attributes involved in the expression. An example of an expression of length 2 is (*age* **in** $[19, 24] \wedge$ *gender* **is** 'male'), representing the males who are older than 18 and younger than 25. An expression with length 1 is called an *elementary* expression. In this paper, we deal with search spaces that contain expressions.

3 Data Mining with Genetic algorithms

Initially, a genetic algorithm [10] randomly generates an initial population. Traditionally, individuals in the population are represented as bit strings. The quality of each individual, i.e., its fitness, is computed. On the basis of these qualities, a selection of individuals is made (an individual may be chosen more than once). Some of the selected individuals undergo a minor modification, called mutation. For some pairs of selected individuals a random point is selected, and the substrings behind this random point are exchanged; this process is called cross-over. The selected individuals, modified or not, form a new population and the same procedure is applied to this generation until some predefined criteria are met.

In the following, we discuss the issues that play a role in tailoring a genetic algorithm for data mining. Section 3.1 is devoted to the representation of individuals and Section 3.2 to the fitness function. Finally, in Section 3.3, we discuss the two operators to manipulate an individual.

3.1 Representation

An individual is regarded as an expression to which some restrictions are imposed with regard to the the notation of elementary expressions and the number of times that an attribute may be involved in the expression. The notation of an elementary expression depends on the domain type of the involved attribute. If there exists no ordering relationship between the attribute values of an attribute *att*, we represent an elementary expression as follows: *expression* := *att* **is** (v_1, v_2,...,v_n), in which $v_i \in$ dom(*att*), $1 \leq i \leq n$. In this way, we express that an attribute *att* assumes one of the values in the set $\{v_1, v_2,...,v_n\}$. If an ordering relationship exists between the domain values of an attribute, an elementary expression is denoted as *expression* := *att* **in** $[v_i, v_k]$, $i \leq k$, in which $[v_i, v_k]$ represents the values within the range of v_i and v_k. An attribute is allowed to participate at most once in an individual. This restriction is imposed to prevent the exploration of expressions to which no tuples satisfy. In the following, an expression to which no tuples qualify will be called an *empty* expression. Consider a database in which, among others, the age of persons is recorded. Then, the expression *age* **in** [19,34] $\wedge$ *age* **in** [39,44] represents the class of persons whose

p_1 = *gender* **is** ('male') $\wedge$ *age* **in** [19,34]
p_2 = *age* **in** [29,44] $\wedge$ *town* **is** ('Almere', 'Amsterdam', 'Weesp') $\wedge$ *category* **is** ('lease')
p_3 = *gender* **is** ('male') $\wedge$ *age* **in** [29,34] $\wedge$ *category* **is** ('lease')
p_4 = *gender* **is** ('female') $\wedge$ *age* **in** [29,40] $\wedge$ *category* **is** ('lease') $\wedge$ *price* **in** [50K, 100K]
p_5 = *gender* **is** ('male') $\wedge$ *price* **in** [20K,45K]

Fig. 1. Example of a population

age is between 19 and 34 as well as between 39 and 44. It should be clear that no persons will satisfy this expression, since *age* is a single-valued attribute.

In the following, a population is defined as a set of individuals. As a running example, we use a database that keeps record of cars and their owners. This artificial database consists of the following universal relation[2]: *Ex(gender, age, town, category, price, damage)*, in which the attributes *gender*, *age*, and *town* refer to the owner and the remainder of the attributes refer to the car. Attribute *category* records whether a car is leased or not, and *damage* records whether a car has been involved in an accident or not. An example of a population consisting of 5 individuals is given in Figure 1.

3.2 Fitness function

A central instrument in a genetic algorithm is the fitness function. Since a genetic algorithm is aimed to the optimization of such a function, this function is one of the keys to success. Consequently, a fitness function should represent all issues that play a role in the optimization of a specific problem. Before enumerating these issues in the context of data mining, we introduce the notion of cover.

Definition 1: Let D be a database and p an individual defined on D. Then, the number of tuples that satisfies the expression corresponding to p is called the cover of p, and is denoted as $||\sigma_p(D)||$. The set of tuples satisfying p is denoted as $\sigma_p(D)$.

Note that p can be regarded as a description of a class in D and $\sigma_p(D)$ summarizes the tuples satisfying p. Within a class we can define subclasses. In the following, we regard classification problem as follows: *Given a target class t, search interesting subclasses, i.e., individuals, within class t.* We note that the target class is the class of tuples in which interesting knowledge should be searched for. Suppose we want to expose the profiles of risky drivers, i.e., the class of persons with (more than average) chances of causing an accident, from the database *Ex(gender, age, town, category, price, damage)*. Then, these profiles should be searched for in a class that records the characteristics of drivers that caused accidents. Such a class may be described as *damage* = 'yes'.

We feel that the following issues play a role in classification problems.

- The cover of the target class. Since results from data mining are used for informed decision making, knowledge extracted from databases should be

[2] We note that a universal relation can be obtained by performing a number of joins between the relations involved in a database.

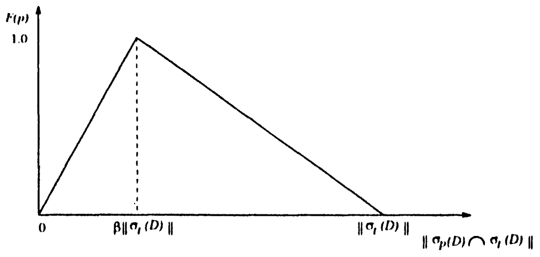

Fig. 2. Shape of the fitness function

supported by a significant part of the database. This increases the reliability of the results. So, a fitness function should take into account that small covers are undesired.

- The ratio of the cover of an individual p to the cover of the target class t, i.e., $\frac{||\sigma_p(D) \cap \sigma_t(D)||}{||\sigma_t(D)||}$. If the ratio is close to 0, this means that only a few tuples of the target class satisfy individual p. This is undesired for the same reason as a small cover for a target class. If the ratio is close to 1, almost all tuples of the target class satisfy p. This is also undesired because this will result in knowledge that is often known. A fitness function should take these properties into account.

Taking into account above-mentioned issues, we have defined the following fitness function:

$$F(p) = \begin{cases} \frac{||\sigma_p(D) \cap \sigma_t(D)||}{\beta ||\sigma_t(D)||} C(t) & \text{if } ||\sigma_p(D) \cap \sigma_t(D)|| \leq \beta ||\sigma_t(D)|| \\ \frac{||\sigma_p(D) \cap \sigma_t(D)|| - ||\sigma_t(D)||}{||\sigma_t(D))||(\beta - 1)} C(t) & \text{otherwise} \end{cases}$$

in which $0 < \beta \leq 1$, and

$$C(t) = \begin{cases} 0 \text{ if } \frac{||\sigma_t(D)||}{||\sigma(D)||} \leq \alpha \\ 1 \text{ otherwise} \end{cases}$$

and $0 \leq \alpha \leq 1$.

We note that the values for α and β should be defined by the user and will vary for different applications. The value of α defines the fraction of tuples that a target class should contain in order to be a candidate for further exploration. The value β defines the fraction of tuples that an individual should represent within a target class in order to obtain the maximal fitness. In Figure 2, the shape of the fitness function is presented.

The fitness grows linearly with the number of tuples satisfying the description of an individual p as well as satisfying a target class t, i.e., $||\sigma_p(D) \cap \sigma_t(D)||$, above a user-defined value α, and decreases linearly with $||\sigma_p(D) \cap \sigma_t(D)||$ after reaching the value $\beta||\sigma_t(D)||$.

It should be clear that our goal is to search for those individuals that approximate a fitness of $\beta||\sigma_t(D)||$. Consider the target class *damage* = *yes* that consists of 100.000 tuples. Assume that a profile is considered risky if about 30.000 out of 100.000 persons satisfy this profile. This means that $\beta \approx 0.3$. Assuming that 33.000 of the persons caused an accident are young males, the algorithm should find individuals like (*gender* **is** ('male') $\wedge$ *age* **in** [19,28]).

3.3 Manipulation operators

Mutation As stated in the introduction of this section, a mutation modifies an individual. In defining the mutation operator, we take into account the domain type of an attribute. If there exists no ordering relationship between the domain values, then we select randomly an attribute value and replace it by another value, which can be a NULL value as well, in an expression that contains this attribute. For example, a mutation on attribute town of individual p_2 (see Figure 1) may result into $p_2' =$ *age* **in** [29,44] $\wedge$ *town* **is** ('Almere', 'Den Haag', 'Weesp') $\wedge$ *category* **is** ('lease').

If there exists a relationship between the domain values of an attribute, the mutation operator acts as follows in the case that a single value is associated with this attribute in an expression, i.e., the expression looks as *att* **is** (v_c). Let $[v_b, v_e]$ be the domain of attribute *att*. In order to mutate v_c, we choose randomly a value $\delta_v \in [0, (v_e - v_b)\mu]$, in which $0 \leq \mu \leq 1$. The mutated value v_c' is defined as $v_c' = v_c + \delta_v$ or $v_c' = v_c - \delta_v$ as long as $v_c' \in [v_b, v_e]$. The parameter μ is used to control the maximal increase or decrease of an attribute value.

To handle overflow, i.e., if $v_c' \notin [v_b, v_e]$, we assume that the successor of v_e is v_b, and, consequently the predecessor of v_b is v_e. To compute a mutated value v_c' appropriately, we distinguish between whether v_c will be increased or decreased, which is randomly determined.

In the case that v_c is increased

$$v_c' = \begin{cases} v_c + \delta_v & \text{if } v_c + \delta_v \in [v_b, v_e] \\ v_b + \delta_v - (v_e - v_c) & \text{otherwise} \end{cases}$$

and in the case v_c is decreased

$$v_c' = \begin{cases} v_c - \delta_v & \text{if } v_c - \delta_v \in [v_b, v_e] \\ v_e - \delta_v + (v_c - v_b) & \text{otherwise} \end{cases}$$

Let us consider the situation in which more than one value is associated with an attribute *att* in an expression. If a list of non successive (enumerable) values is associated with *att*, we select one of the values and compute the new value according to one of the above-mentioned formulas. If a range of successive values, i.e., an interval, is associated with *att*, we select either the lower or upper bound value and mutate it. A potential disadvantage of this strategy for intervals is that an interval may be significantly enlarged, if the mutated value crosses a domain boundary. Suppose that the domain of *age* is [18,60], and we mutate the upper bound value of the expression *age* **in** $[55, 59]$, i.e., the value 59. Assume that the value 59 is increased by 6, then 59 is mutated in the value 23. The new expression becomes *age* **in** $[23, 55]$.

We note that the partitioning of attribute values, i.e., the selection of proper intervals in an expression, is simply adjusted by the mutation operator.

Cross-over The idea behind a crossover operation is as follows; it takes as input 2 expressions, selects a random point, and exchanges the subexpressions behind this point. In general, not all attributes will be involved in an expression. This may have some undesired effects for a cross-over. First, a cross-over may produce individuals in which an attribute is involved more than once. For example, a cross-over between the individuals p_1 = *gender* **is** ('male') $\wedge$ *age* **in** [19,34] and p_2 = *age* **in** [29,44] $\wedge$ *town* **is** ('Almere', 'Amsterdam', 'Weesp') $\wedge$ *category* **is** ('lease') after the first attribute results into the following individuals: p_1' = *gender* **is** ('male') $\wedge$ *town* **is** ('Almere', 'Amsterdam', 'Weesp') $\wedge$ *category* **is** ('lease') and p_2' = *age* **in** [19,34] $\wedge$ *age* **in** [29,44]. As we can see, the attribute *age* appears twice in p_2'.

Second, a cross-over may result in an offspring that is exactly the same as the parents with probability 1.0. For example, a cross-over between p_1 and p_2 that occurs on a point that is beyond the last elementary expression of p_2, i.e., *category* **is** ('lease'), will result into the equal new individuals.

To prevent the above-mentioned effects, we apply the following technique to perform cross-overs. Consider two individuals p_i and p_j that have been selected for crossover. Let A_i be the set of attributes that is not involved in p_i but is involved in p_j and A_j the set of attributes that is not involved in p_j but is involved in p_i. Then, for each attribute in A_i, we generate an empty elementary expression using this attribute and add it to p_i. The same procedure is applied to the attributes of A_j. This procedure has as effect that the lengths of p_i and p_j become equal. Finally, we regard an individual as a sequence of elementary expressions, and order these in p_i and p_j according to the rule that elementary expressions having the same attribute will appear at the same position in p_i and p_j. Then, the cross-over between p_i and p_j can be performed. We note that the cross-over point should be chosen between the first and final position of p_i or p_j. The following example illustrates this technique.

Example 1 Consider the individuals p_1 = *gender* **is** ('male') $\wedge$ *age* **in** [19,34] and p_2 = *age* **in** [29,44] $\wedge$ *town* **is** ('Almere', 'Amsterdam', 'Weesp') $\wedge$ *category* **is** ('lease') again. Then, $A_1 = \{$*town*, *category*$\}$ and $A_2 = \{$*gender*$\}$. So, we extend p_1 with the following expression *town* **is** ('') $\wedge$ *category* **is** ('') and p_2 is extended with *gender* **is** ('').

After ordering the elementary expressions p_1 and p_2 look as follows:
p_1 = *gender* **is** ('male') $\wedge$ *age* **in** [19,34] $\wedge$ *town* **is** ('') $\wedge$ *category* **is** ('')
p_2 = *gender* **is** ('') $\wedge$ *age* **in** [29,44] $\wedge$ *town* **is** ('Almere', 'Amsterdam', 'Weesp') $\wedge$ *category* **is** ('lease')
Now, a cross-over at position 2 results into
p_1' = *gender* **is** ('male') $\wedge$ *age* **in** [19,34] $\wedge$ *town* **is** ('Almere', 'Amsterdam', 'Weesp') $\wedge$ *category* **is** ('lease')
p_2' = *gender* **is** ('') $\wedge$ *age* **in** [29,44] $\wedge$ *town* **is** ('') $\wedge$ *category* **is** ('')
Note that p_2' is equal to *age* **in** [29,44]. □

In the next section, we introduce a number of rules that may prevent the exploration of unpromising individuals.

4 Optimization rules

In this section, we discuss two propositions that may be used to prevent the exploration of unprofitable individuals. These propositions are derived from the shape of the fitness function. The complexity of a genetic-based algorithm for data mining is determined by the evaluation of the fitness function [5], since this is computationally the most expensive operation. Before presenting these propositions, we introduce the notion of a similar of an individual.

Definition 2: Let length(p) be the number of elementary expressions involved in p. An individual p_{sim} is a *similar* of p if each elementary expression of p_{sim} is contained in p or p_{sim} contains each elementary expression of p and length(p_{sim}) $\neq$ length(p).

As stated in the foregoing, we search for individuals with high values for the fitness function F, see section 3.2. for F.

We note that the computation of $F(p)$ requires the number of tuples that satisfy individual p. So, these tuples should be searched for and retrieved from the database, which is a costly operation [6]. Although several techniques may be used to minimize the number of retrievals from a database, still large amounts of tuples have to be retrieved from the database in mining applications. The techniques to minimize retrievals are mainly based on storing frequently used tuples in an efficient way in main memory [4]. In this way, disk accesses are reduced.

In the following, two propositions will be presented that may be used to avoid the computation of fitness values of unprofitable individuals. These propositions decide if the fitness value of a similar of an individual p is worse than the fitness of p. If this is the case, this similar can be excluded from the search process.

Proposition 1: Let p_{sim} be a similar of p. If $||\sigma_p(D) \cap \sigma_t(D)|| \leq \beta||\sigma_t(D)||$ and length(p_{sim}) $>$ length(p) then $F(p_{sim}) \leq F(p)$.

Proof. From length(p_{sim}) $>$ length(p) follows that $\sigma_{p_{sim}}(D) \subseteq \sigma_p(D)$. As a consequence, $||\sigma_{p_{sim}}(D) \cap \sigma_t(D)|| \leq ||\sigma_p(D) \cap \sigma_t(D)||$. Since $||\sigma_p(D) \cap \sigma_t(D)|| \leq \beta||\sigma_t(D)||$, it follows $F(p_{sim}) \leq F(p)$. □

Proposition 2: Let p_{sim} be a similar of p. If $||\sigma_p(D) \cap \sigma_t(D)|| \geq \beta||\sigma_t(D)||$ and length(p_{sim}) $<$ length(p) then $F(p_{sim}) \leq F(p)$.

Proof. Similar to the proof of Proposition 1. □

Note that the propositions do not require additional retrievals from a database to decide if $F(p_{sim}) \leq F(p)$.

We discuss an alternative how these propositions at cross-over level may contribute in optimizing the search process. As stated in the foregoing, a cross-over is applied on a mating pair and results into two offsprings. Suppose that a mutation is performed after a cross-over, and the parent and the offspring with the highest fitness values are eligible to be mutated (see Section 5). Consider an offspring p_o resulted from a cross-over, and let p_o be a similar of p, one of its parents. If we can decide that $F(p_o) \leq F(p)$, then it is efficient to mutate p_o. The reason is that computation on an unmutated p_o will be a wasting of effort.

In the next section, we propose an overall algorithm, in which we apply the two propositions.

5 Algorithm

The previous section was devoted to the major issues that play a role in designing a genetic-based algorithm for data mining. In this section, we describe the overall algorithm. Before starting this description, we discuss a mechanism to select an individual for a next generation.

The mechanism to select individuals for a new generation is based on the technique of elitist recombination [12]. According to this technique, the individuals in a population are randomly shuffled. Then, the cross-over operation is applied on each mating pair, resulting into two offsprings. The parent and the offspring with the highest fitness value are selected for the next generation. In this way, there is a direct competition between the offsprings and their own parents. Note, the offspring provides the possibility to explore different parts of a search space.

The elitist recombination technique has been chosen for two reasons. First, there is no need to specify a particular cross-over probability, since each individual is involved in exactly one cross-over. Second, there is no need for intermediate populations in order to generate a new population as is the case in a traditional genetic algorithm. These properties simplify the implementation of a genetic algorithm. Let us outline the overall algorithm.

The algorithm starts with the initialization of a population consisting of an even number of individuals, called $P(t)$. The individuals in this population are shuffled. Then, the cross-over operation is applied on two successive individuals. After completion of a cross-over, the fitness values of the parents are compared[3]; the parent with the highest value is selected and it may be mutated with a probability c. This parent, p'_{sel}, is added to the next generation, and in case it is mutated its fitness value is computed. Then, for each offspring, p_o, we test if this offspring is a similar of p'_{sel} and if its fitness value is worse or equal than p'_{sel}. If this is true, p_o is an unpromising individual, and, therefore, we always mutate p_o. Otherwise, we mutate p_o with probability c. Note, to compare the fitness value between p'_{sel} and p_o, the propositions of the previous section are used. So, no additional fitness values are computed for this comparison. After possible mutation of the offsprings, their fitness values are computed, and the most fittest offspring is added to the new generation. This process is repeated for all individuals in a generation.

Once the new population has been built up, the total fitness of the existing as well as of the new population is computed, and compared. The algorithm

[3] These values are already computed and stored by the algorithm.

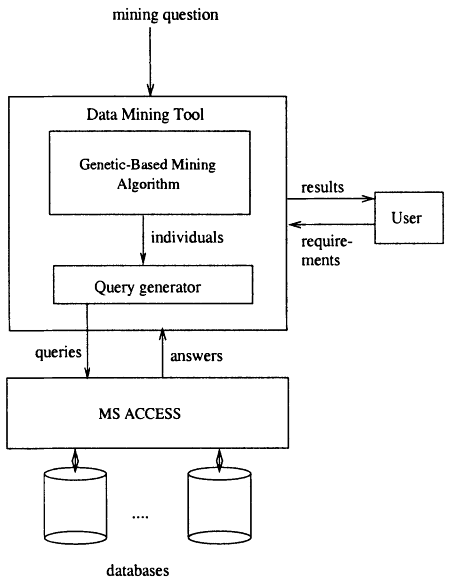

Fig. 3. Architecture of a data mining tool

terminates if the total fitness of the new population does not significantly improve compared with the total fitness of the existing population, i.e., that the improvement of the total fitness of the new population is less than a threshold value ϵ. For a detailed discussion with regard to the algorithm, we refer to [5].

6 Implementation and preliminary results

Based on the algorithm described in Section 5, we have built a re-targetable prototype of a data mining tool, which means that the tool can be coupled to different databases. The main goal of the current effort is to determine if the genetic-based algorithm is able to find hidden knowledge.

Let us continue with the description of the tool. The tool takes as input a mining question and produces a set of answers for the question. The prototype is running in a Microsoft Access 97 environment that uses the Microsoft Jet Engine[4]. The genetic-based algorithm is implemented in Visual Basic. We have chosen this environment for two reasons. First, this environment is available at our laboratory. Second, the database that we want to mine is a Microsoft database.

In Figure 3, the architecture of our tool is represented. Once the tool receives a mining question, it runs the genetic-based mining algorithm, which generates, among others, a population. The individuals of the population are passed to the Query Generator. For each individual a corresponding SQL query is generated.

[4] Within the scope of a feasibility study, a previous version of the algorithm was implemented in C and connected to the Monet Database Server [3].

These queries are passed to the MS ACCESS DBMS, which on its turn processes the queries and passes the results to the data mining tool. These results are used to compute the fitness of each individual. Upon request of the user individuals and their associated fitness values can be shown. The user has the possibility to modify the initial mining question or to specify additional requirements with regard to the question. We note that this is a very useful feature of a mining tool, since, in practice, a user starts a mining session with a rough idea of what information might be interesting, and during the mining session (with help of the results provided by the system) the user specifies more precisely what information should be searched for.

The generation of a query corresponding to an individual p is straightforward. Recall that an individual is a conjunction of predicates. So, this forms the major part of the WHERE clause of a query corresponding to p. Since we require the number of tuples satisfying p and a target class t, the query corresponding to p is: *select count(*) from* database *where* $p \wedge t$.

We have applied the tool on an artificial database, and are currently applying it on a real-life database. In the following, we describe the databases and the results obtained so far.

Artificial database This database consists of the relation *Ex(gender, age, town, category, price, damage)*. For this database 100.000 tuples have been generated, of which 50% have a value "yes" for *damage*, i.e., 50% of the tuples relate to an accident. Furthermore, the fact that young men in lease cars have more than average chances to cause an accident was hidden in the database. The goal of mining this database was to determine whether the tool is capable to find the hidden fact. Therefore, we have set the target class as *damage* = 'yes', and we searched for the profile of risky drivers. We note that the expression for the hidden profile is: *age* **in** [19,24] $\wedge$ *category* **is** ('lease') $\wedge$ *gender* **is** ('male').

We have mined the database with varying initial populations, consisting of 36 individuals. The following three classes of initial population were distinguished: (1) random: the populations contained a few individuals that could set the algorithm quickly on a promising route, (2) modified random: individuals that could apparently set the algorithm on a promising route were replaced by other (not promising) individuals, and (3) bad converged: the populations contained individuals with low fitness values.

We have observed that the algorithm usually finds near optimal solutions, i.e., profiles that look like the hidden one, in less than 1000 fitness evaluations. The differences between the hidden profile and profiles found by the algorithm (for different initial populations) were mainly caused by variations in the range of attribute *age*.

With regard to the settings of the parameters α and β, we note that appropriate values could be easily selected, since the content of the database is precisely known. □

Real-life database Currently, we are mining a real-life database, the so-called

FAA incident database, which is available at our aerospace laboratory. This database contains aircraft incident data that are recorded from 1978 to 1995. Incidents are potentially hazardous events that do not meet the aircraft damage or personal injury thresholds as defined by American National Transportation Safety Board (NTSB). For example, the database contains reports of collisions between aircraft and birds while on approach to or departure from an airport. The FAA database consists of more than 70 attributes and about 80.000 tuples.

The initial mining task on the FAA database was: search for the class of flights with (more than average) chances of causing an incident, i.e., profiles of risky flights. This search resulted in (valid) profiles but which could be easily declared. An example of such a profile is that aircraft with 1 or 2 engines are more often involved in incidents. The explanation for this profile is that these types of aircraft perform more flights.

During the mining process the mining question was refined in the following three more specific questions: (1) given the fact that an incident was due to operational defects not inflicted by the pilot, what is the profile of this type of incident?, (2) given the fact that an incident was due to mistakes of the pilot, what is the profile of this type of incident?, and (3) given the fact that an incident was due to improper maintenance, what is the profile of this type of incident?

We have proposed these questions to our tool with the following values for the parameters, $\alpha = 0$, $\beta = 0.25$, mutation probability (c) = 0.3, and $\epsilon = 0.1$. Furthermore, the population size was set on 50. On the first glance, the results of the tool appear to be promising. Safety experts at our laboratory are analysing the results. On the basis of their analysis, we will set up a plan to mine the database more systematically, and to study the impact of different parameter values on the results provided by the tool. The goal of the latter study is to formulate some guidelines for selecting parameter values for similar type of databases, such as an aircraft accident database. □

Although our evaluation is not completed yet and a significant amount of research has to be done, e.g., on performance issues, in order to build an adequate genetic-based data mining tool, the preliminary results are promising. A second observation is that the range interval of an attribute in an expression may significantly enlarged, if a mutation occurs on a domain boundary. An interval that consists of (almost) the whole the domain slows down the search process. In a next version of the tool, we will enhance the mutation operator. An alternative is to clip on boundary values in cases of overflow.

7 Conclusions & further research

In order to answer mining questions, very large search spaces should be inspected, making an exhaustive search infeasible. So, heuristic search strategies are of vital importance in searching such spaces. We have discussed a genetic-based algorithm that may be used for data mining. Contrary to the conventional bit string representation in genetic algorithms, we have chosen a representation that fits better in the field of databases. The fitness function discussed in this paper is close to our intuition and gives rise to an optimization of the search process.

A genetic-based algorithm for data mining has two major advantages. First, the problem of partitioning attribute values in proper ranges could be solved by choosing a suitable mutation operator. Second, a genetic-based algorithm is able to escape a local optimum and does not pose any restrictions on the structure of a search space.

By means of a (prototype) implementation and a preliminary evaluation, we have shown the potentials of a genetic-based data mining tool. Since the preliminary results of the tool appear to be promising, we are setting up a research plan to evaluate this tool thoroughly. The outcome of the evaluation will determine our future research activities in this field.

Acknowledgements The author is grateful to Wim Pelt from the Dutch Ministry of Defense, who made this research possible. Hein Veenhof and Egbert Boers from NLR are thanked for their valuable comments on earlier drafts of this paper. Finally, Leo de Penning and Martijn Suurd from the Univ. of Twente are thanked for their implementation efforts.

References

[1] Agrawal, R., Ghosh, S., Imielinski, T., Iyer, B., Swami, A., An Interval Classifier for Database Mining Applications, Proc. 18th Int. Conf. Very Large Data Base, pp. 560-573.

[2] Augier, S., Venturini, G., Kodratoff, Y., Learning First Order Logic Rules with a Genetic Algorithm, Proc. 1st Int. Conf. on Knowledge Discovery and Data Mining, pp. 21-26.

[3] Boncz, P., Wilschut, A., Kersten, M., Flattening an Object Algebra to Provide Performance, Proc. 14th Int. Conf. on Data Engineering, pp. 568-577.

[4] Choenni, R., Siebes, A., Query Optimization to Support Data Mining, Proc. DEXA '97 8th Int. Workshop on Database and Expert Systems Applications, pp. 658-663.

[5] Choenni, R., On the Suitability of Genetic-Based Algorithms for Data Mining, extended version, to appear as NLR technical publication.

[6] Elmasri, R., Navathe, S., Fundamentals of Database Systems, The Benjamin/Cummings Publishing Company, 1989.

[7] Freitas, A., A Genetic Programming Framework for two Data Mining Tasks: Classification and Generalized Rule Induction, Proc. Int. Conf on Genetic Programming 1997, pp. 96-101.

[8] Han, J., Cai, Y., Cerone, N., Knowledge Discovery in Databases: An Attribute-Oriented Approach, in Proc. 18th Int. Conf. Very Large Data Base, pp. 547-559.

[9] Holsheimer, M., Kersten, M.L., Architectural Support for Data Mining, Proc. AAAI-94 Workshop on Knowledge Discovery, pp. 217-228.

[10] Michalewicz, Z., Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, New York, USA.

[11] Srikant, R., Agrawal, R., Mining Quantitative Association Rules in Large Relational Tables, Proc. ACM SIGMOD'96 Int. Conf. Management of Data, pp. 1-12.

[12] Thierens, D., Goldberg, D., Elitist Recombination: an integrated selection recombination GA, 1st IEEE Conf. on Evolutionary Computing, pp. 508-512.