Nationaal Lucht- en Ruimtevaartlaboratorium
National Aerospace Laboratory NLR

NLR-TP-2000-198

Mining aircraft incident databases with a genetic-based algorithm

R. Choenni

.

Nationaal Lucht- en Ruimtevaartlaboratorium
National Aerospace Laboratory NLR

NLR-TP-2000-198

Mining aircraft incident databases with a genetic-based algorithm

R. Choenni

This investigation has been carried out under a contract awarded by the Royal Netherlands Navy (RNLN), contract number 726.98.0343.01. RNLN has granted NLR permission to publish this report.

This report is published in the Proceedings of the 3rd Federal Symposium on Data Mining, AFCEA Press, March 2000.

Division:	Information and Communication Technology
Issued:	August 2000
Classification of title:	unclassified

Abstract

The capabilities of conventional data analysis tools are limited for a thorough analysis of aircraft incident databases. This is due to the enormous number of attributes and the various types of attributes involved in those databases. Since data mining has as goal to extract knowledge from databases without putting any restriction on the type and amount of data, data mining techniques are interesting for the analysis of aircraft incident databases. We have mined two real-life aircraft incident databases with a prototype data mining tool that is developed at our laboratory. This tool is based on a genetic algorithm. In this paper, we report on the implementation of this tool and the mining results that we have obtained with regard to the aircraft incident databases.

Contents

(23 pages in total)

1 Introduction

Organisations have realised that their databases may contain knowledge that can improve the quality of decisions taken in the present or future. Conventional data analysis tools are inadequate to extract this knowledge, while manually extracting this knowledge is a time-consuming and tedious process at best. Therefore, there is a practical need to (partially) automate this process. Data mining contributes to the need by combining techniques from different fields, such as database technology, machine learning, statistics, and artificial intelligence. Research and development in data mining evolves in several directions, such as association rules, time series, and classification. The direction of association rules is focussing on the development of algorithms to find frequently occurring patterns in databases [AgIS93, AgSr94]. In time series databases, one tries to find all common patterns embedded in a database of sequences of events [AgSr95]. Classification of tuples based on common characteristics and the search for rules in a class is another direction of interest [AgGI92, HaCC92, HoKe94]. In this paper, we address data mining problems as mentioned in the last direction.

Many data mining problems can be formulated as search problems. A database is regarded as a set of tuples, and each (projected) subset of tuples is considered as an element in the search space. The problem is to select interesting subsets without inspecting the whole search space. For example, in a car insurance environment the identification of profiles of risky drivers, i.e., drivers with (more than average) chances of causing an accident, can be modelled as a search problem. Consider an artificial relation *Driver(gender, age, town, category, price, damage)*, in which the attributes *gender, age, town* refer to the driver, while the remaining 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. The challenge is to select a conjunction of predicates that represents the group of risky drivers. Assume that young males in leased cars form this group. Then, an expression like: *gender* = “male” $\wedge$ *age* $\in$ [18, 24] $\wedge$ *category* = “leased” should be searched for, which actually represents a projected subset of tuples in the database.

In general, the search spaces that should be inspected in order to answer mining questions are very large, making an exhaustive search infeasible. Therefore, heuristic search strategies are of vital importance to data mining. In the literature a wide variety of heuristic search strategies have been reported to walk efficiently through a search space, e.g., hill climbing, simulated annealing, genetic algorithms, etc. The success of an algorithm is often dependent on the structure of the search space. For example, a hill climber will generally perform better on a search that consists of a few optima, while a genetic algorithm will perform better if the search space consists of many optima. The reason is that a hill climber terminates if it reaches an

optimum, while a genetic algorithm does not. In a search space that contains many optima, it will generally be worthwhile to continue searching the space after having found the first optimum. Unfortunately, the search spaces that stem from data mining problems neither has a specific structure nor are the structures known in advance. On the basis of evidence, one should choose for a search strategy. Therefore, a data mining tool should be equipped with several search strategies.

It has been recognised by several researchers that genetic algorithms might be suitable for a wide range of data mining tasks [AuVK95, Frei97]. Genetic algorithms have been successfully applied in several areas, such as air traffic management [KeAK96], query optimization [Grae93], data warehouse design [ZaYa99], etc. A genetic algorithm is capable of exploring different parts of a search space.

This paper is devoted to the implementation of a prototype genetic-based data mining tool, called SHARVIND, and the application of SHARVIND on two real-life databases, containing aircraft incident data. SHARVIND is running in a Microsoft ACCESS environment that uses the Microsoft Jet Engine. The tool takes as input a mining question, which actually selects the part of the database where interesting knowledge should be searched. Furthermore, a number of parameter values are required as input. These values are necessary to run the genetic-based algorithm, which is the core of SHARVIND. We note that it is possible to specify requirements: as use attribute att_j and/or do not use att_k. The output of the tool is a set of expressions. In order to be selected as output, an expression should meet the following two criteria. First, the expression should be a conjunction of predicates, in which an attribute (involved in the database) appears at most once. We note that an expression actually represents a projected subset of tuples in the database. Second, the number of tuples that is represented by an expression should be in an interval, which is defined by a domain expert.

As noted before, we have mined two real-life databases. Both databases contain aircraft incident data, one is set up by the Federal Aviation Administration (FAA) in the USA, referred as FAA database, and the other is set up by the Joint Research Centre (JRC) in Italy, referred as ECCAIRS database. We note that the FAA database is obtained from the Internet and is mined at our laboratory, while the JRC database is mined on location. In the FAA database aircraft incident are recorded from 1978 to 1995, while ECCAIRS is recently in production. Currently, ECCAIRS consists of data that is converted from the Scandinavian accident and incident reporting system.

We have mined both databases by posing the mining question “What are the profiles of risky flights?” We have posed to the FAA databases some additional mining questions (concerning safety aspects), such as “Given the fact an incident was due to operational defects not inflicted by the pilot, what is the profile of this type of incident”, etc. We have presented the mining results to safety experts at our laboratory and our overall conclusion was that the answers to the mining questions were correct and promising. The mining results helped safety experts to gain insight in the databases and hopefully also knowledge in future.

The remainder of this paper is organised as follows. In Section 2, we view data mining tasks as searching problems, in which the search space consists of an enormous number of expressions. Then, in Section 3, we tailor a genetic algorithm to walk efficiently through the search space. In Section 4, we discuss a prototype data mining tool, called SHARVIND, which is based on the previously described genetic algorithm. Section 5 is devoted to the mining of two aircraft incident databases with SHARVIND and the mining results that we have obtained. Finally, Section 6 concludes the paper.

2 Data mining and searching

The large amount of data stored in databases may serve two purposes. First, it may help in understanding a phenomenon, and second it may help to predict the outcome of similar phenomena. In our view, data mining is a powerful tool that contributes to the realization of these purposes. Data mining has as goal to extract potentially useful information from large databases by using a wide variety of methods and techniques that are able to explore large data sets efficiently.

In the following, a database consists of a universal relation [Ullm89]. 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 $\mathbf{dom}(att_1) \times \mathbf{dom}(att_2) \times ... \times \mathbf{dom}(att_n)$, in which $\mathbf{dom}(att_j)$ 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 is attached. An expression is defined as a conjunction of predicates, and is used to select a set of tuples from the database, which in turn represents a class in the database. Consider the earlier introduced relation *Driver(gender, age, town, category, price, damage*) and a snapshot of this relation as depicted in Table 1. For example, the expression *gender* = “male” selects the set of tuples corresponding to the males in the database, i.e., the tuples 1, 3 and 4 in Table 1, the expression $age \in [18, 24] \wedge category$ = “leased” selects the set of tuples corresponding to persons between eighteen and twenty four years old driving a leased car, i.e., the tuples 1 and 3, etc. A main advantage of introducing the notion of expression is that we do

not have to enumerate explicitly all tuples of a class, and therefore an expression can be regarded as a summary/description of a class. So, a database contains an enormous number of classes, and each class can be represented as an expression or a disjunction of expressions.

Table 1 Snapshot of the database Driver

Tuple Identifier	gender	age	Town	category	price	damage
1	male	20	Almere	leased	70K	yes
2	female	35	Amsterdam	not leased	80K	yes
3	male	24	Amsterdam	leased	75K	yes
4	male	28	Almere	not leased	40K	yes
5	female	28	The Hague	leased	50K	no
…	…	…	…	…	…	…

Note that the following disjunction (*gender* = "male" $\wedge$ *age* = 20 $\wedge$ *town* = "Almere" $\wedge$ *category* = "leased"$\wedge$ *price* = 70K $\wedge$ *damage* = "yes") $\vee$ (*gender* = "female" $\wedge$ *age* = 35 $\wedge$ *town* = "Amsterdam" $\wedge$ *category* = "not leased"$\wedge$ *price* = 80K $\wedge$ *damage* = "yes") is a straightforward way to select the first and the second tuple from Table 1.

Data mining may now be regarded as the search for useful, previously unknown expressions from a space of expressions without inspecting the whole space. The search space is formed by all possible expressions with regard to a database. Note that the number of expressions grows exponentially with the number of attributes and the number of tuples in the database.

Although the usefulness of an expression generally depends on the application, we know beforehand that the following three categories of expressions will not be interesting.

1. Expressions that select zero tuples. An example of such an expression is *age* = 20 $\wedge$ *age* = 36. Since *age* is a single valued attribute, no tuples will qualify for this expression.

2. Expressions that select a set of arbitrary tuples that do not have any common characteristics. Such a set can easily be obtained by a disjunction of an arbitrary number of expressions.

3. Expressions that consist of a single predicate. Knowledge with regard to these expressions is stored in the data dictionary, which is easily accessible for database administrators.

Therefore, we discard these categories from the search space by imposing the following restrictions to expressions.

1. An attribute appears at most once in an expression.
2. Disjunctions of expressions are not allowed in the search space.
3. An expression should contain at least one conjunction.

So, the elements of the search space are expressions that satisfy above-mentioned restrictions. The challenge is to come up with search strategies that are able to find the interesting elements by inspecting a relatively small part of the search space. In the next section, we discuss a genetic-based search strategy.

3 Genetic-based mining algorithm

Genetic algorithms are heuristic search strategies inspired by natural genetics and evolutionary principles [Holl75, Mich95]. They have been proven to be promising in a wide variety of applications. In some cases it is shown that genetic algorithms converge to an optimum, while in other cases experience show that they converge. However it is still an open question why convergence occurs.

A genetic algorithm 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 (individuals 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. So, a genetic algorithm is characterised by the representation, the fitness function, and the manipulation operators. In the following, we briefly discuss these characterisations in the context of data mining. For a more detailed discussion, we refer to [Choe98].

3.1 Representation

A population is defined as a set of individuals. We represent an individual as an expression to which a few restrictions are imposed with regard to the notation. The notation of an elementary expression, i.e., the expression consists of exactly one predicate, 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.

We note that an individual represents a class or group in the database. In the following, the terms individual and class are used interchangeably.

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 contain all issues that play a role in the optimization of a specific problem. We note that three issues have been implicitly incorporated in the notion of expressions. First, we have demanded that classes are not empty. Second, we have demanded that the tuples within a class meet some common properties (see Section 2). Third, we have demanded that a class should be described by at least one conjunction.

Before introducing two other issues that play a role in searching interesting classes, we introduce the notion of cover and target class. The number of tuples that corresponds to an individual is called the cover of the individual. A target class is the set of tuples within interesting expressions should be search for. This may be the whole database or a part of the database. Suppose that we want to expose the profiles of risky drivers from the database *Driver(gender, age, town, category, price, damage*), i.e., the class of persons with (more than average) chances of causing an accident. 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 should be modelled explicitly in a fitness function.

- The cover of the target class. Since results from data mining are used for informed decision making, knowledge extracted from databases should be 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. If this 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 this ratio is close to 1, almost all tuples of the target class satisfy p. This is also undesirable because this will result in knowledge that is often known. A fitness function should take these properties into account.

We have modelled these issues in a fitness function. The fitness grows linearly with the number of tuples satisfying the description of an individual and the description of a target class above a threshold value α, and decreases linearly with the number of tuples satisfying the description of an individual and the description of a target class after reaching a user defined value β. For a detailed description of the function, we refer to [Choe98].

Our goal is to reward those individuals that approximate a fitness of β. Consider the target class *damage* = "yes" that consists of 100.000 tuples. Assume that according to the domain experts a profile is considered risky if about 30.000 persons satisfy this profile. This means that $\beta \approx 30.000$. Assuming that 33.000 of the persons that caused an accident are young males, the algorithm should find individuals like (*gender* **is** (`male') $\wedge$ *age* **in** [19,28]).

3.3 Manipulation operators

Genetic algorithms provide two operators to manipulate an individual, mutation and cross-over. A mutation takes care of a minor modification of an individual, and is therefore, responsible for locally inspecting a search space. The effect of a cross-over may be that a completely different part of the search space is inspected. In this section, we briefly discuss the meaning of mutation and cross-over operators for data mining.

Mutation 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 we 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 an individual p = *age* **in** [29,44] $\wedge$ *town* **is** ("Almere", "The Hague", "Rotterdam") $\wedge$ *category* **is** ("leased") may result into p' = *age* **in** [29,44] $\wedge$ *town* **is** ("Amsterdam", "The Hague", "Rotterdam") $\wedge$ *category* **is** ("leased").

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 parameter μ is used to control the maximal increase or decrease of an attribute value. The mutated value v_c' is defined as $v_c' = v_c + \delta_v$ or $v_c' = v_c - \delta_v$ as long as $v_c' \in [v_b,v_e]$. 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 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.

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. As noted before, partitioning of attribute values is in general a tough problem [SrAg].

Cross-over The idea behind a crossover operator is as follows; it takes as input two expressions, selects a random point, and exchanges the subexpressions behind this point. To illustrate this idea, we consider a relation $R(att_1, att_2, ..., att_n)$ and two expressions, e_i and e_j, in which all attributes are involved. Let e_i be defined as $(e_i^1 \wedge e_i^2 \wedge e_i^3 \cdots e_i^{k-1} \wedge e_i^k \wedge e_i^{k+1} \cdots e_i^n)$, in which e_i^k represents an elementary expression in which attribute att_i is involved. And, let e_j be defined as $(e_j^1 \wedge e_j^2 \wedge e_j^3 \cdots e_j^{k-1} \wedge e_j^k \wedge e_j^{k+1} \cdots e_j^n)$. Then, a cross-over between e_i and e_j at point k results into the following two expressions: $(e_i^1 \wedge e_i^2 \wedge e_i^3 \cdots e_i^{k-1} \wedge e_i^k \wedge e_j^{k+1} \cdots e_j^n)$ and $(e_j^1 \wedge e_j^2 \wedge e_j^3 \cdots e_j^{k-1} \wedge e_j^k \wedge e_i^{k+1} \cdots e_i^n)$.

In general, not all attributes will be involved in an expression, which may cause undesired effects after a cross-over. Therefore, an expression is completed with the missing attributes before it undergoes a cross-over [Choe98].

4 SHARVIND: A prototype data mining tool

In this section, we discuss the architecture and the implementation of a prototype data mining tool based on a genetic algorithm as described in the previous section. We start with the description of SHARVIND. Then, we discuss some implementation issues.

4.1 Architecture

Currently, we may distinguish three modules in SHARVIND, a user interface, a genetic-based mining algorithm, and a query generator. These modules and the architecture of SHARVIND are depicted in Figure 1.

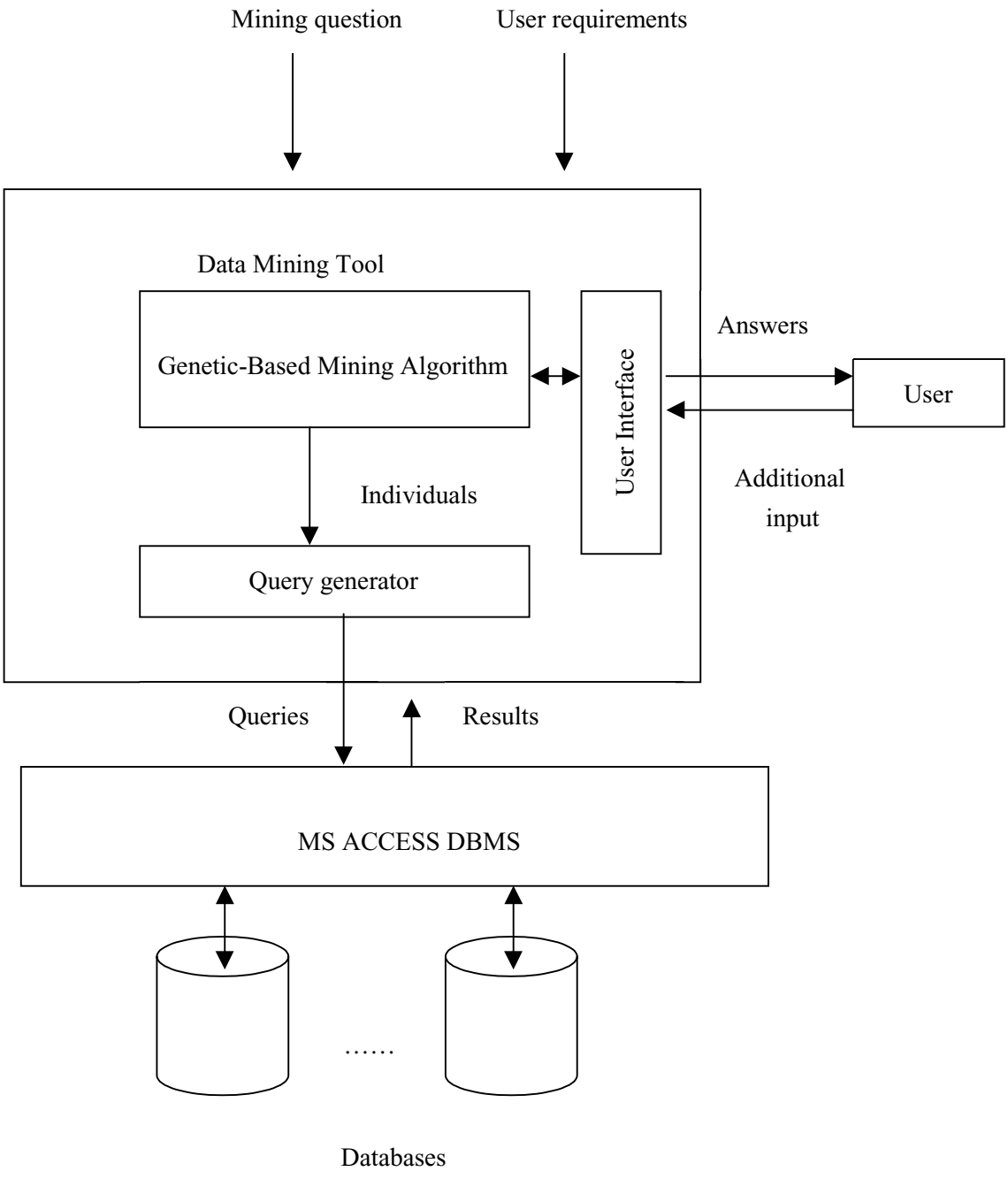

Figure 1: Architecture of SHARVIND

In Figure 1, the input consists of a mining question and additional requirements that may be posed by the user. For example, a user may demand that an attribute in a database should not be involved in the mining process for reasons of privacy (e.g., income of pilots). The user is allowed to request intermediate mining results, and if desired, the user is able to modify the input. We note that this is a useful feature of a mining tool, since, in practice, a user starts a

mining session with a rough idea of the information that might be interesting, and during the mining session the user more precisely specifies, based on, amongst others, the mining results obtained so far, which information should be searched for.

Once SHARVIND receives the input, it invokes the genetic-based mining algorithm. This algorithm starts with the initialization of a population consisting of an even number of individuals. The individuals in this population are shuffled. Then, the cross-over operation is applied on two successive individuals. We note that the cross-over operator is applied exactly once for an individual. After completion of a cross-over, the fitness values of the parents are compared; the parent with the highest value is selected and it may be mutated with a probability *c*. This parent, is added to the next generation, and in the case it is mutated its fitness value is computed. After possible mutation of the offsprings, their fitness values are computed and compared. The fittest offspring is added to the new generation as well. 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 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.

In order to compute the fitness value of an individual, the number of tuples that satisfies the description of the individual is required. Therefore, the database should be interrogated. An individual should be translated into queries that is understood by the underlying database management system (dbms), in our case the MS ACCESS dbms. This is the task of the Query Generator.

Since in our case an individual is an expression, the translation of an individual in an equivalent set of SQL queries is straightforward. Note that the individual already forms the WHERE clause of a SQL query. Therefore, the Query generator is relative simple. Suppose that we require the number of tuples in the database *Driver* that satisfies the description of an individual *p*: (*gender* **is** “male” $\wedge$ *age* $\in$ [18, 24]). Then the corresponding SQL query is:
SELECT COUNT (*) FROM *Driver* WHERE (*gender* = “male”) AND (*age* BETWEEN 18 AND 24).

4.2 Implementation

Currently, SHARVIND is running in a Microsoft Access 97 environment that uses the Microsoft Jet Engine. The genetic-based algorithm is implemented in Visual Basic and consists of approximately 2000 lines of code. We have chosen this environment for three reasons. First, this environment is available at our laboratory. Second, the database that we want to mine is a Microsoft database. Third, this environment is suitable for rapid application.

As discussed in the previous section, major components in developing a genetic-based algorithms are the representation of individuals, manipulation operators, fitness function, and the translation of an individual into a query that is understood by the MS ACCESS database management system (dbms).

Individuals, which are regarded as a conjunction of predicates over attributes, are implemented as binary tables. An ordered attribute, att is implemented as two tuples: < att, lower bound value > and < att, upper bound value >. An unordered attribute having n values is implemented as n tuples, having the form < att, value >, in the table. For each individual, a binary table is built up in this way. So, the individual (gender is "male" $\wedge$ age in [18, 24] $\wedge$ category is "leased") is implemented as follows:

Attribute name	**Attribute value**
gender	male
age	18
age	24
category	leased

Once the data structure of an individual was defined, the implementation of the manipulation operators was straightforward. A cross-over is obtained by selecting two tables, splitting each table into two subtables, let's say a head and a tail table. Then, the tail tables are exchanged. A mutation is obtained by deleting and/or inserting one or more tuples. Suppose that in the above-mentioned individual *gender* **is** "male" should be mutated in *gender* **is** "female". This obtained by removing the tuple < *gender*, male > from the table and inserting the new tuple < *gender*, female >.

Due to mutations it may occur that the range interval of an attribute grows to the domain of that attribute. In such a case, the whole database will be covered by the attribute, which is undesired for the search process. To prevent this situation, we have decided that an interval corresponding

to an attribute may not grow harder than a user defined threshold value, e.g., (upper bound domain value - lower bound domain value)* x, in which $x \in (0, 1]$.
The implementations of the fitness function as well as the query generator were straightforward. For each generation, we keep track of the average fitness, and the individuals with the highest and lowest fitness. These values are stored in tables as well. From these tables, the progress of the search is plotted, i.e., the average fitness, best and worst fitness, with primitives of MS Graph.

5 Mining two databases

In this section, we give an overview of two databases that we have mined with SHARVIND and the results that we have obtained. The first database, called ECCAIRS is located at the Joint Research Centre (JRC) in Italy. Currently, this database contains serious incident and accident data that is converted from the Scandinavian accident- and incident reporting system. This database grows with approximately 4% per year. Detail information about ECCAIRS can be found on the web site of JRC, which http://eccairs-www.jrc.it, and in [Groe00].

The second database, called the FAA database, contains aircraft incident data that have been 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. This database can be obtained from the Internet, http://www.asy.faa.gov/asp/asy_fids.asp.

Both databases are implemented in MS ACCESS and contain NULL values and redundant data. Furthermore, integrity rules (e.g., for domain values) are hardly implemented.

In the following we discuss these databases in more detail and the mining results that we have obtained.

5.1 ECCAIRS

The ECCAIRS database consists of 36 relations and about 300 attributes. Two major relations of the database are the ACS and the OCCS relations. The ACS relation contains information with regard to aircraft, such as manufacturer, motor, speed of the aircraft, etc., and information about the environment in which the aircraft is involved, such as weather conditions. The OCCS relation describes in general terms an occurrence (incident or accident) and contains general information with regard to an occurrence, for example, time and location of an occurrence, etc.

The relation ACCS contains 5202 tuples and 186 attributes and *OCCS* contains 5202 tuples as well and 27 attributes. Although the number of tuples is not large, mining might be interesting due to the large number of attributes.

However, currently 17 relations do not contain any tuples, while other relations consist of tuples having many NULL values. To gain insight in the number of NULL values in each relation, we define a filling factor as follows:

$$filling(R) = \frac{\#\, values(R)}{\max(values(R))} \times 100\%$$, in which **max**(*values(R)*)= # *tuples(R)* × # *attributes(R)*.

We note that # *values(R)* is the number of values in relation R, # *tuples(R)* is the number of tuples in *R,* etc.

It appears that *filling(ACCS)* = 19% which is a bad score especially for mining and *filling(OCCS)* = 72%. In order to make the database suitable for mining we have removed

- All attributes that have less than 2000 entries filled in.

- All attributes whose values consist of natural language. Although these attributes may expose interesting knowledge, they are removed, since our algorithm is not yet able to handle them adequately. A major revision of the algorithm is required if we allow such attributes in the mining process.

- Attributes that are fully functional dependent on another attribute. For example, from the latitude/longitude the city name can be derived. So, city name is a redundant attribute

- Attributes with a high selectivity factor, i.e. equal to one. The selectivity factor of an attribute *att* is defined as $\frac{1}{\text{card}(att)}$, in which card(*att*) is the number of distinct values that *att* assumes.

- Attributes with a very low selectivity factor, i.e., close to $1/5202$. Note that attributes that serve as identifiers typically have selectivity factors close to 1/5202.

After performing the removals 64 attributes were left and the filling factors for *ACCS* and *OCCS* become 81% and 84% respectively. Then, we join these relations into a single relation.

Since the amount of tuples is relatively small, we have mined the whole joined relation. So, we have not defined any specific target classes.

5.2 FAA

At our laboratory, this database is implemented as a single table that is sorted on an attribute, called report number, which served as primary key. We note that the data in this database has been copied from the database that is accessible from Internet and is made available by FAA. In the following, we mean by the FAA database, the database as it is implemented and filled at our laboratory.

The FAA database consists of more than 70 attributes and about 60.000 tuples. As in the case of ECCAIRS, this database contains also NULL values, redundant data, and attributes with very high and low selectivity factors. Therefore, we have cleaned this database in the same way as ECCAIRS, in order to make it suitable for mining. After cleaning 30 attributes were left and 60.000 attributes.

5.3 Mining results

We have mined above-mentioned database with different values for a number of parameters such as number of individuals of a population, average length of an individual, mutation probability, and the maximal interval to which an attribute is allowed to grow. For the influence of different parameter settings, we refer to [Groe00] and [PeSu98]. It appears to be reasonable to set the number of individuals around 50, the average length between 2 and 5 attributes, mutation probability between 0.1 and 0.4, and the maximum interval to which an attribute may grow around 10%.

Since the amount of tuples in the ECCAIRS database is relatively small, we have mined the joined relations. So, we have not defined any specific target classes. So, the mining question was: “What are profiles of risky flights?” We have proposed this question to SHARVIND with different β values. Recall that β defines the fraction of tuples that an individual should represent within a target class (in this case the whole database) in order to obtain the maximal fitness. The results that we obtain were correct but most of them were trivial. For example, when we set β to 0.2, it appears that male pilots and scattered (1/8 to 4/8) sky conditions are involved in 19% of the incidents. Although the most of the results were trivial, it helped to gain insight in the database. For example, we have observed that about 50% of the database consist of incidents where no serious injury occurs and the pilot satisfies the required license.

Setting β to 0.3 delivers the following (more complex) association:

aircraft_type **is** "fixed wing" **AND** *power-type* **is** *"*reciprocating*"* **AND** *license_class* **is** "required rating" **AND** *highest_degree_of _injury* **is** "none" → accident.

This rule says that pilots who hold the required licenses and are flying with fixed wing aircraft and a reciprocating power-type cause about 30% of the accidents. However, these incidents do not cause any injury.

We have mined the FAA database by posing several mining questions to SHARVIND. Our initial mining question was: search for the class of flights with (more than average) chances of causing an incident, i.e., profiles of risky flights. We have searched for this class with different input values. These searches resulted in (valid) profiles that could easily be explained by our safety experts. 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.

Therefore, we have refined our mining question into a number of questions, such as

- 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?

- Given the fact that an incident was due to mistakes of the pilot, what is the profile of this type of incident?

- Given the fact that an incident was due to improper maintenance, what is the profile of this type of incident?

The cardinality of the target classes associated with the questions posed to SHARVIND varied from 2500 to 30000 tuples. The value of β varied from 0.10 to 0.20.

The answers to the proposed questions were correct and most of them contained no surprise to our domain expert. In one case the tool came up with an interesting and unexpected result. In this case, we had chosen as target class = "pilot induced". We had chosen to mine this target class in order to find out what type of pilots was causing incidents. The target class contains 26000 tuples. Mining the target class with β = 0.15 resulted in the following association:

aircraft_damage **is** "minor" **AND** *primary_ flight_type* **is** *"personal"* **AND** *type_of_operation* **is** "general operating rules" **AND** *flight_plan* **is** "none" **AND** *pilot_rating* **is** "no rating" → pilot_induced.

This association means that pilots without flight certificates and flight plans who are flying in private aircraft are causing more incidents than other groups. This result was on the first glance a bit strange for our safety expert, since pilots without flight certificates are not allowed to fly. After a while it appears that the association was correct and our safety expert was able to explain the association. The pilots without certificates appeared to be students whose incidents were recorded in the FAA database as well.

6 Conclusions and further research

It has been recognised by several researchers that genetic algorithms might be suitable for data mining tasks. In the literature, a number of efforts can be found that describe frameworks to tailor genetic algorithms for data mining. However, many of these frameworks are neither implemented nor evaluated. We have implemented vital parts of a prototype data mining tool that is based on a genetic algorithm. We have coupled the tool to an MS ACCESS environment in order to mine two aircraft incident databases. Given the large number and various types of attributes involved in these databases, conventional data analysis tools are inadequate for a thorough analysis of these databases. Our overall conclusions are:

- Both databases could be significantly reduced, especially in the number of attributes, due to NULL values, text attributes, etc.

- Although both databases are relatively small after cleaning, we feel that data mining is a promising direction for analysing aircraft incident databases. The mining results helped safety experts to gain insight in these databases. We expect that data mining may expose knowledge from these databases in future, if more and better data will be loaded. We note that data is recently started to be loaded in the ECCAIRS database.

- With regard to the genetic algorithm, our conclusion is that a genetic algorithm may rapidly be implemented for data mining, yielding reasonable results. However, to build an operational tool, there is still a significant effort required.

Since many real-life databases, including aircraft incident databases, contain many unstructured data, i.e., natural language, a topic for further research is to extend our tool with this type of data. Furthermore, to get insight in the results and performance provided by a genetic-based data mining tool, a large scale evaluation is required, which is a topic for further research as well.

Acknowledgements. The author is grateful to the Royal Netherlands Navy which sponsored this research. Furthermore, the author is grateful to Drs. W. Pelt, Ir. B. Bruggeman and Dr. Hein Veenhof for his valuable comments on earlier drafts of this paper and to his students Erik Groenheiden, Leo de Penning, and Martijn Suurd for analysing the databases and their implementation efforts.

Biographical sketch. Sunil Choenni is a principal scientist in the Data and Knowledge Systems group at the National Aerospace Laboratory (NLR) and an assistant professor in the computer science department at the University of Twente, both in the Netherlands. He holds a M.Sc (1990) degree in theoretical computer science from Delft University of Technology and a Ph.D (1995) in databases from the University of Twente. Before joining NLR and the University of Twente, he held research positions at the Joint Administration Office (GAK) and the National Centre of Mathematics and Computer Science (CWI), both in the Netherlands. Furthermore, he held visiting research positions at the University of Genoa and the State University of Milan, both in Italy. His current research interests are knowledge discovery in databases, multi-media databases, military information systems, and optimisation aspects of databases.

7 References

[AgGI92] Agrawal, R., Ghosh, S., Imielinski, T., Iyer, B., Swami, A., An Interval Classifier for Database Mining Applications, in Proc. of the 18th Very Large Data Base, pp. 560-573.
[AgIS93] Agrawal, R., Imielinski, T., Swami, A., Mining Association Rules between Sets of Items in Large Databases, in Proc. ACM SIGMOD '93 Int. Conf. on Management of Data, pp. 207-216.
[AgSr94] Agrawal, R., Srikant, R., Fast Algorithms for Mining Association Rules, in Proc. Int. Conf. on Very Large Databases, pp. 487-499.
[AgSr95] Agrawal, R., Srikant, R., Mining Sequential Patterns, in Proc. 11th Int. Conf. on Data Engineering, pp. 3-14.
[AuVK95] Augier, S., Venturini, G., Kodratoff, Y., Learning First Order Logic Rules with a Genetic Algorithm, in Proc.
1st Int. Conf. on Knowledge Discovery and Data Mining, pp. 21-26.
[Choe98] Choenni, R., On the Suitability of Genetic-Based Algorithms for Data Mining, in Proc. ER98 Workshops Advances in Database Technologies, pp.55-67.
[Frei97] Freitas, A., A Genetic Programming Framework for two Data Mining Tasks: Classification and Generalized Rule Induction, in Proc. Int. Conf. on Genetic Programming, pp. 96-101.
[Grae93] Graefe, G., Query Evaluation Techniques for Large Databases, in ACM Computing Surveys 25(2), pp73-170.
[Groe00] Groenheiden, E., A Genetic Data Mining Feasibility Study, To appear as internal report, University of Twente, The Netherlands.
[HaCC92] Han, J., Cai, Y., Cerone, N., Knowledge Discovery in Databases: An Attribute-Oriented Approach, in Proc. of the 18th Very Large Data Base, pp. 547-559.
[Holl75] Holland, J.H., Adaption in Natural and Artificial Systems, Ann Arbor: University of Michigan Press, USA, 1975.
[HoKe94] Holsheimer, M., Kersten, M., Architectural Support for Data Mining, in Proc. AAAI-94 Workshop on Knowledge Discovery, pp. 217-228.
[KeAK96] Kemenade van, C., Akker van den, M., Kok, J., Evolutionary Air Traffic Flow Management for large 3D-problems, in Proc. Int. Conf. Parallel Problem Solving from Nature, pp. 910-919.
[Mich95] Michalewicz, Z., Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, New York, USA.
[PeSu98] Penning, H. de, Suurd, P., NLR Genetic Search Base, internal report, University of Twente, The Netherlands.
[Ullm89] Ullman, J.D., Principles of Database and Knowledge-Base Systems, Vol.1, Computer Science Press, New York, USA.
[ZaYa99] Zhang, C., Yang, J., Genetic Algorithm for Materialized View Selection in Data Warehouse Environments, in Proc. DaWaK99, 1st Int. Conf. on Data Warehousing and Knowledge discovery, pp. 116-125.