

Web Intelligence: Web-Based BISC Decision Support System (WBISC-DSS)

Gamil Serag-Eldin, Souad Souafi-Bensafi, Jonathan K. Lee, Wai-Kit Chan,
Masoud Nikraves

BISC Program, Computer Sciences Division, EECS Department
University of California, Berkeley, CA 94720, USA

Email: nikraves@cs.berkeley.edu

Tel: (510) 643-4522

Fax: (510) 642-5775

Abstract: Most of the existing search systems (software) are modeled using crisp logic and queries. In this chapter, we introduce fuzzy querying and ranking as a flexible tool allowing approximation where the selected objects do not need to exactly match the decision criteria resembling natural human behavior. The model consists of five major modules: the Fuzzy Search Engine, Application Templates, the User Interface, the Database, and Evolutionary Computing. The system is designed in a generic form to accommodate more diverse applications and to be delivered as stand-alone software to academia and businesses.

1 Introduction

Searching database records and ranking the results based on multi-criteria queries is central for many database applications used within organizations in finance, business, industry and other fields. Most of the available systems (software) are modeled using crisp logic and queries, which results in rigid systems with imprecise and subjective processes and results. In this chapter we introduce fuzzy querying and ranking as a flexible tool allowing approximation where the selected objects do not need to exactly match the decision criteria resembling natural human behavior (Nikraves 2001b; Nikraves and Azvine 2002; Nikraves 2003a).

The model consists of five major modules: the Fuzzy Search Engine (FSE), Application Templates (AT), the User Interface (UI), the Database (DB) and Evo-

lutionary Computing (EC). We developed the software with many essential features. It is built as a web-based software system that users can access and use over the Internet. The system is designed to be generic so that it can run different application domains. To this end, the Application Template module provides information of a specific application as attributes and properties, and serves as a guideline structure for building a new application.

The Fuzzy Search Engine (FSE) is the core module of the system. It has been developed to be generic so that it would fit any application. The main FSE component is the query structure, which utilizes membership functions, similarity functions and aggregators.

Through the user interface, a user can enter and save his profile, input criteria for a new query, run different queries and display results. The user can manually eliminate the results he disapproves of or change the ranking according to his preferences.

The Evolutionary Computing (EC) module monitors ranking preferences of the user's queries. It learns to adjust to the intended meaning of the user's preferences.

We present our approach with three important applications: ranking (scoring), which has been used to make financing decisions concerning credit cards, car and mortgage loans; college admissions where hundreds of thousands of applications are processed yearly by U.S. universities; and date matching as one of the most popular internet programs. Even though we implemented three applications, the system is designed in a generic form to accommodate more diverse applications and to be delivered as stand-alone software to academia and businesses.

2 Model framework

The DSS system starts by loading the application template, which consists of various configuration files for a specific application (see section 4) and initializing the database for the application (see section 6), before handling a user's requests, (see figure 1).

Once the DSS system is initialized, users can enter their own profiles in the user interface or make a search with their preferences. The control unit of the system handles these requests. The control unit converts user input into data objects that are recognized by the DSS system. Based on the request types, it forwards them to the appropriate modules.

If the user wants to create a profile, the control unit will send the profile data directly to the database module, which stores the data in the database for the application. If the user wants to query the system, the control unit will direct the user's preferences to the Fuzzy Search Engine, which queries the database (see section 3). The query results will be sent back to the control unit and displayed to the users.

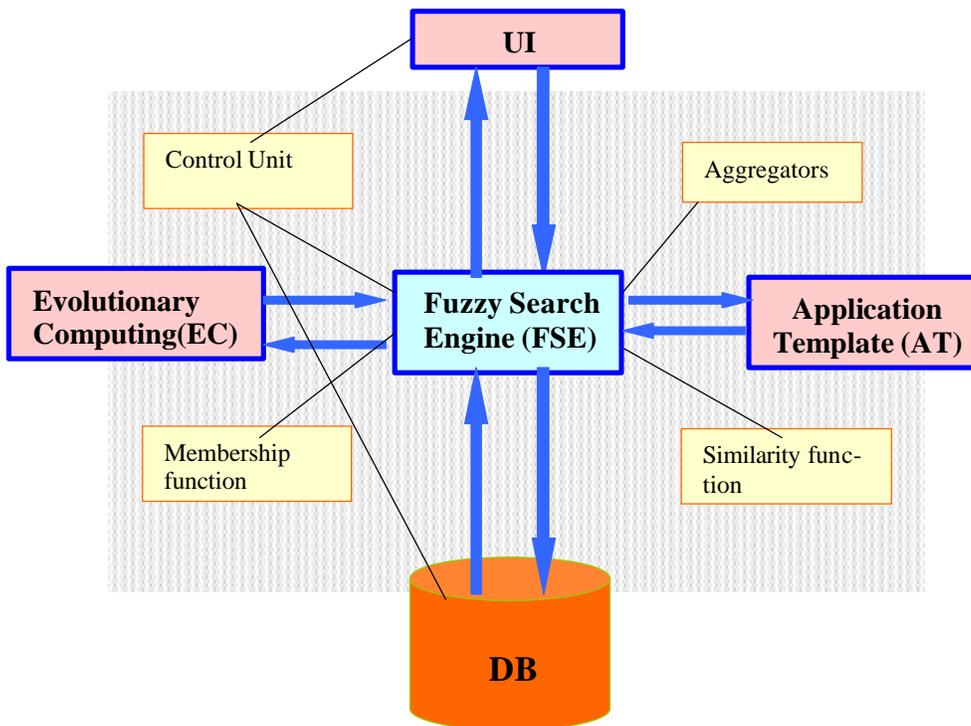


Figure 1 The BISC-DSS general framework

3 Fuzzy Engine

During the recent years, applications of fuzzy logic and the internet from web data mining to intelligent search engine and agents for internet applications have greatly increased (Nikraves 2002; Nikraves et al. 2002, 2003a, 2003b, 2003c;

Nikraves and Choi 2003; Ioi et al. 2002, 2003; Nikraves and Azvine 2001, 2002; Takagi et al. 2002a, 2002b).

3.1 Fuzzy Query, Search and Ranking

To support generic queries, the fuzzy engine has been designed to have a tree structure. There are two types of nodes in the tree, category nodes and attribute nodes, as depicted in figure 2. While multiple category levels are not necessary, they are designed to allow various refinements of the query through the use of the type of aggregation of the children. Categories act only to aggregate the lower levels. The attribute nodes contain all the important information about a query. They contain the membership functions for the fuzzy comparison as well as the use of the various aggregation methods to compare two values.

The flow of control in the program when a query is executed is as follows. The root node receives a query formatted as a fuzzy data object and is asked to compare the query fuzzy data to a record from the database also formatted as a fuzzy data object. At each category node, the compare method is called for each child and then aggregated using an aggregator object.

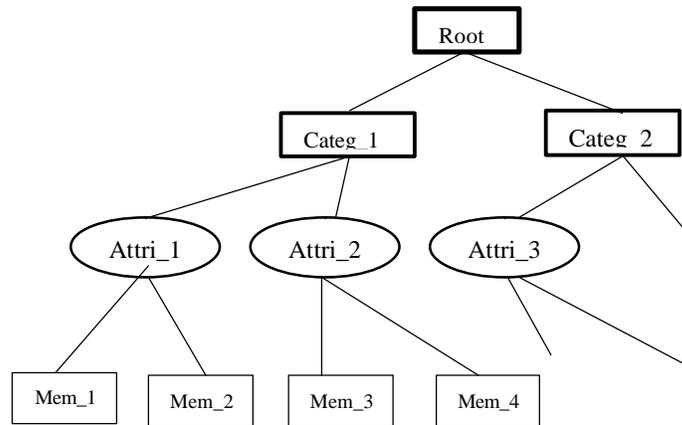


Figure 2 The Fuzzy search engine tree structure.

The attribute nodes handle the compare method slightly different than the category nodes. There are two different ways attributes may be compared. The attribute nodes contain a list of membership functions comprising the fuzzy set. The degrees of membership for this set are passed to the similarity comparator object, which currently has a variety of different methods to calculate the similarity be-

tween the two membership vectors. In the other method, the membership vector is created by having full membership to a single membership function specified in the fuzzy data object, but no membership value for the other functions (Sugeno 1974).

The resulting comparison value returned from the root node is assigned to the record. The search request is then added to a sorted list ordered by this ranking in descending value. Each of the records from the database is compared to the query and the results are returned. For certain search criteria, it may be desirable to have exact values in the query. For such criteria, the database is used to filter the records for comparison.

3.2 Membership function

Currently there are three membership functions implemented for the Fuzzy Engine. A generic interface has been created to allow several different types of membership functions to be added to the system (Grabisch et al 2000). The three types of membership functions in the system are: Gaussian, Triangular and Trapezoidal. These functions have three main points, for the lower bound, upper bound and the point of maximum membership. For other functions, optional extra points may be used to define the shape (an extra point is required for the trapezoidal form).

4 Application template

The DSS system is designed to work with different application domains. The application template is a format for any new application we build; it contains data of different categories, attributes and membership functions of that application. The application template module consists of two parts the application template data file, and the application template logic.

The application template data file specifies all the membership functions, attributes and categories of an application. We can consider it as a configuration data file for an application. It contains the definition of membership functions, attributes and the relationship between them.

The application template logic parses and caches data from the data file so that other modules in the system can have faster access to definitions of membership functions, attributes and categories. It also creates a tree data structure for the fuzzy search engine to transverse. Figure 3 shows part of the sample configuration file from the Date Matching application.

```
#####
#This is a properties file for membership definition. We should specify
#the following properties for an attribute:
# - A unique identifier for each defined membership function.
# - A type from the following: {Gaussian, Triangle, Trapezoid}
# - Three points: Lowerbound, Upperbound, Maximum
# - Optional point: Auxillary Maximum
# Format:
# <MF_Name>.membershipFunctionName = <MF_Name>
# <MF_Name>.membershipFunctionType = {Gaussian/Triangle/Trapezoid}
# <MF_Name>.lowerBound = lowerBoundValue
# <MF_Name>.upperBound = upperBoundValue
# <MF_Name>.maxValue = max Value
# <MF_Name>.optionPoint = pt1, pt2, pt3 ...
#
#####
```

```
#####
#
# Gender Membership Functions
#
male.membershipFunctionName = male
male.membershipFunctionType = Triangle
male.lowerbound = 1
male.upperbound = 1
male.maxValue = 1

female.membershipFunctionName = female
female.membershipFunctionType = Triangle
female.lowerbound = 0
female.upperbound = 0
female.maxValue = 0
#
# Age Membership Functions
#
young.membershipFunctionName = young
young.membershipFunctionType = Triangle
young.lowerbound = 0
young.upperbound = 35
young.maxValue = 20

middle.membershipFunctionName = middle
middle.membershipFunctionType = Triangle
middle.lowerbound = 20
middle.upperbound = 50
middle.maxValue = 35

old.membershipFunctionName = old
old.membershipFunctionType = Triangle
old.lowerbound = 35
old.upperbound = 100
old.maxValue = 50
```

Figure 3 Template of the date matching application

5 User interface

It is difficult to design a generic user interface that suits different kind of applications for all the fields. For example, we may want to have different layouts for user interfaces for different applications. To make the DSS system generic while preserving the user friendliness of the interfaces for different applications, we developed the user interfaces into two parts.

First, we designed a specific HTML interface for each application we developed. Users can input their own profiles, make queries by specifying preferences for different attributes. Details for the DSS system are encapsulated from the HTML interface so that the HTML interface design would not be constrained by the DSS system.

The second part of our user interface module is a mapping between the parameters in the HTML files and the attributes in the application template module for the application. The input mapping specifies the attribute names to which each parameter in the HTML interface corresponds. With this input mapping, a user interface designer can use input methods and parameter names freely.

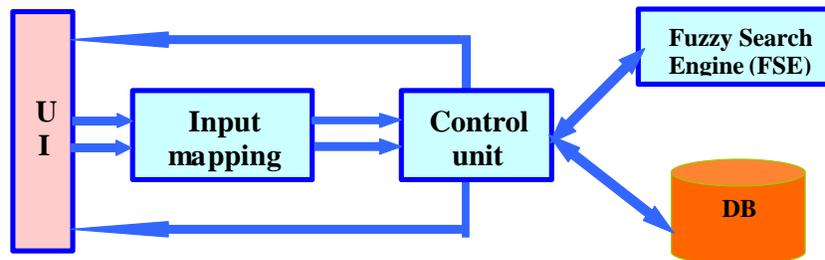


Figure 4 User interface data flow

6 Database (DB)

The database module is responsible for all the transactions between the DSS system and the database. This module handles all queries or user profile creations from the Fuzzy Engine and the Control Unit respectively. For queries from the Fuzzy Search Engine, it retrieves data from the database and returns it in a data object form. Usually queries are sets of attribute values and their associated weights. The database module returns the matching records in a format that can be manipulated by the user such as eliminating one or more record or changing their

order. To create a user profile, it takes data objects from the Control Unit and stores it in the database. There are three components in the DB module: the DB Manager (DBMgr), the DB Accessor (DBA) and DB Accessor Factory (DBA Factory).

6.1 DB Manager

The DB Manager is accountable for two things: setting up database connections and allocating database connections to DB Accessor objects when needed. During the initialization of the DSS system, DB Manager loads the right driver, which is used for the communications between the database and the system. It also supplies information to the database for authentication purposes (e.g. username, password, path to the database etc).

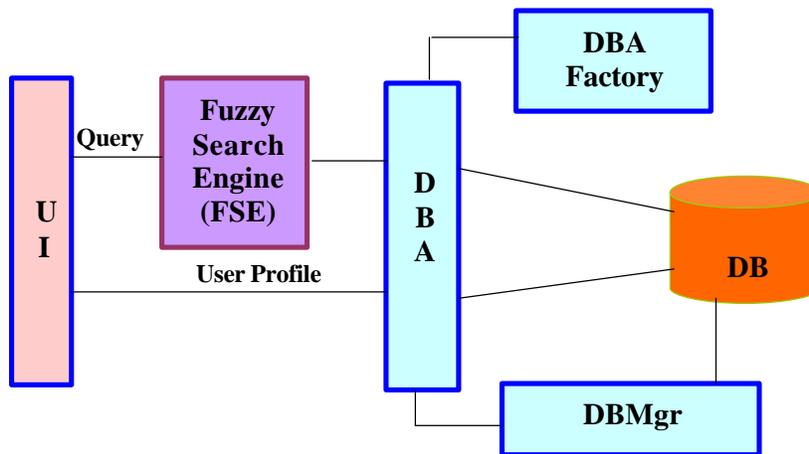


Figure 5 Database module components

6.2 DB Accessor Factory

The DB Accessor Factory creates DB Accessor objects for a specific application. For example, if the system is running the date matching application, DB Accessor Factory will create DB Accessor objects for the date matching application. The existence of this class serves the purpose of using a generic Fuzzy Search Engine.

6.3 DB Accessor

DB Accessor is responsible for storing and getting user profiles to and from the database. It also saves queries from users to the database so that other modules in the system can analyze user's preferences. It is the component that queries the da-

tabase and wrap result from the database into data objects that are recognized by our application framework.

7 Measure of association and fuzzy similarity

As in crisp query and ranking, an important concept in fuzzy query and ranking applications is the measure of association or similarity between two objects in consideration (Murofushi and Sugeno 1989). For example, in a fuzzy query application, a measure of similarity between a query and a document, or between two documents, provides a basis for determining the optimal response from the system (Fagin 1998). In fuzzy ranking applications, a measure of similarity between a new object and a known preferred (or non-preferred) object can be used to define the relative goodness of the new object. Most of the measures of fuzzy association and similarity are simply extensions from their crisp counterparts. However, because of the use of perception-based and fuzzy information, the computation in the fuzzy domain can be more powerful and more complex. This section gives a brief overview of various measures of fuzzy association and similarity and various types of aggregation operators involved, along with the description of a simple procedure of utilizing these tools in real applications (Detyniecki 2000).

Various definitions of similarity exist in the classical, crisp domain, and many of them can be easily extended to the fuzzy domain. However, unlike in the crisp case, in the fuzzy case the similarity is defined on two fuzzy sets. Suppose we have two fuzzy sets A and B with membership functions $\mu_A(x)$ and $\mu_B(x)$, respectively. The arithmetic operators involved in the fuzzy similarity measures can be treated using their usual definitions while the union and the intersection operators need to be treated specially. It is important for these operator pairs to have the following properties: (1) conservation, (2) monotonicity, (3) commutativity, and (4) associativity. It can be verified that the triangular norm (T-norm) and triangular co-norm (T-conorm) (Detyniecki 20001; Nikravesh 2001b; Mizumoto 1989; Fagin 1998; Grabisch 1996) conform to these properties and can be applied here. A detailed survey of some commonly used T-norm and T-conorm pairs along with other aggregation operators can be find at (Nikravesh et al 2003c).

Having introduced a variety of tools that are required to evaluate fuzzy association/similarity between two objects, a simple algorithm in pseudo code is provided below to illustrate how these machineries can be used in a practical implementation.

Input: two objects A and B

A: N discrete attributes

For the i^{th} attribute, A^i is an array of length M^i , where M^i is the number of possible linguistic values of the i^{th} attribute.

i.e. each A_j^i , i in $1, \dots, N$ and j in $1, \dots, M^i$, gives the degree of A 's i^{th} attribute having j^{th} linguistic value.

B: similar to A with the same dimensions.

Other parameters:

AggregatorType

SimilarityType

TNormType

OptionalWeights

Output: An aggregated similarity score between A and B

Algorithm:

For each $i=1$ to N

$SAB^i = \text{ComputeSimilarity}(A^i, B^i, \text{SimilarityType}, \text{TNormType})$

End

Return $\text{Aggregate}(SAB, \text{AggregatorType}, \text{OptionalWeights})$

Sub $\text{ComputeSimilarity}(X, Y, \text{SimilarityType}, \text{TNormType})$

Switch SimilarityType:

Case SimpleMatchingCoefficient:

Return $|X \cap Y|$

Case CosineCoefficient:

Return $|X \cap Y| / (|X|^{1/2} |Y|^{1/2})$

Case OverlapCoefficient:

Return $|X \cap Y| / \min(|X|, |Y|)$

Case Jaccard's Coefficient:

Return $|X \cap Y| / (|X \cup Y|)$

Case Dice's Coefficient:

Return $2|X \cap Y| / (|X| + |Y|)$

...

End

Sub $\text{Aggregate}(S, \text{AggregatorType}, \text{OptionalWeights})$

Switch AggregatorType:

Case Min:

Return $\min(S)$

Case Max:

Return $\max(S)$

Case Mean:

Return $\text{mean}(S)$

Case Median:

Return $\text{median}(S)$

Case WeightedAverage:

Return $\text{WeightedAverage}(S, \text{OptionalWeights})$

Case OrderedWeightedAverage:

Return $\text{OrderedWeightedAverage}(S, \text{OptionalWeights})$

Case ChoquetIntegral:

```

    Return ChoquetIntegral(S, OptionalWeights)
Case SugenoIntegral:
    Return SugenoIntegral(S, OptionalWeights)
...
End

```

This algorithm takes as input two objects, each with N discrete attributes. Similarity scores between the two objects are first computed with respect to each attribute separately, using a specified similarity metric and T-norm/conorm pair. As described previously, the computation of a similarity score with respect to an attribute involves a pair wise application of the T-norm or T-conorm operators on the possible values of the attribute, followed by other usual arithmetic operation specified in the similarity metric (Yager 1988). Finally, an aggregation operator with appropriate weights is used to combine the similarity measures obtained with respect to different attributes.

In many situations, the controlling parameters, including the similarity metric, the type of T-norm/conorm, the type of aggregation operator and associated weights, can all be specified based on the domain knowledge of a particular application. However, in some other cases, it may be difficult to specify a priori an optimal set of parameters. In those cases, various machine learning methods can be employed to automatically “discover” a suitable set of parameters using a supervised or unsupervised approach. For example, the Genetic Algorithm (GA) and DNA-based computing, as described in later sections, can be quite effective.

8 Implementation - Fuzzy Query and Ranking

In this section, we introduce fuzzy query and fuzzy aggregation for credit scoring, university admissions and date matching.

8.1 Credit Scoring

Credit scoring was first developed in the 1950's and has been used extensively in the last two decades. In the early 1980's, the three major credit bureaus, Equifax, Experian, and TransUnion worked with the Fair Isaac Company to develop generic scoring models that allow each bureau to offer an individual score based on the contents of the credit bureau's data. FICO is used to make billions of financing decisions each year serving a 100 billion dollar industry. Credit scoring is a statistical method to assess an individual's credit worthiness and the likelihood that the individual will repay his/her loans based on their credit history and current credit accounts. The credit report is a snapshot of the credit history and the credit score is a snapshot of the risk at a particular point in time. Since 1995, this scoring system has made its biggest contribution in the world of mortgage lending. Mortgage investors such as Freddie Mac and Fannie Mae, the two main government-chartered companies that purchase billion of dollars of newly originated home loans annu-

ally, endorsed the Fair Isaac credit bureau risk, ignored subjective considerations, but agreed that lenders should also focus on other outside factors when making a decision.

When you apply for financing, whether it's a new credit card, car or student loan, or a mortgage, about 40 pieces of information from your credit card report are fed into a model (Nikraves et al 2003c). This information is categorized into the following five categories with different level of importance (% of the score):

- Past payment history (35%)
- Amount of credit owed (30%)
- Length of time credit established (15%)
- Search for and acquisition of new credit (10%)
- Types of credit established (10%)

The screenshot displays the 'BISC Credit Rating System' web application. The browser address bar shows 'http://localhost:20144/credit/bisc/register.jsp'. The page features a navigation menu on the left with categories: Account Balances, Account Information, Revolving Accounts, Loan Information, Finance Accounts, and Delinquencies. The main content area is titled 'ONLINE FINANCIAL TOOLS' and contains several sections for data entry:

- Account Balances:** Information about outstanding account balances. Fields include: Amount used on accounts (4500), Amount used on revolving accounts (High), Amount paid due on accounts (1000), and Accounts currently paid as agreed (Many).
- Account Information:** The number of accounts. Fields include: Number of established accounts (2), Accounts with recent payment information (Many), Accounts with balances (Some), and Accounts opened in the last 12 months (Many).
- Revolving Accounts:** Information about bank and National Revolving Accounts. Fields include: Number of revolving accounts (1), Number of bank revolving or other revolving accounts (1), No recent revolving balances (radio buttons for True/False, with True selected), Number of bank or national revolving accounts with balances (2), Recent bank revolving information (Some Information), Recent revolving account information (Little Information), and Proportion of balances to credit limits on revolving accounts (Very Low).
- Loan Information:** Loan and Loan Balance information. Fields include: Years of installment loan history (0) and Recent installment loan information (Plenty of Information).

Figure 7. A snapshot of the variable input for credit scoring software.

Given the factors presented earlier, a simulated model has been developed. A series of excellent, very good, good, not good, not bad, bad, and very bad credit scores have been recognized (without including history). Then, fuzzy similarity and ranking have been used to rank the new user and define his/her credit score. In the inference engine, the rules based on factual knowledge (data) and knowledge drawn from human experts (inference) are combined, ranked, and clustered based on the confidence level of human and factual support. This information is then used to build the fuzzy query model with associated weights. In the query level, an intelligent knowledge-based search engine provides a means for specific queries. Initially we blend traditional computation with fuzzy reasoning. This effectively provides validation of an interpretation, model, hypothesis, or alternatively, indicates the need to reject or reevaluate. Information must be clustered, ranked, and translated to a format amenable to user interpretation.

Figures 7-8 show a snapshot of the software developed for credit scoring. To test the performance of the model, a demo version of the software is available at: <http://zadeh.cs.berkeley.edu/> (Nikravesh 2001a). Using this model, it is possible to have dynamic interaction between model and user. This provides the ability to answer "What if?" questions in order to decrease uncertainty, to reduce risk, and to increase the chance to increase a score.

8.2 University Admissions

Hundreds of millions of applications were processed by U.S. universities resulting in more than 15 million enrollments in the year 2000 for a total revenue of over \$250 billion. College admissions are expected to reach over 17 million by the year 2010, for total revenue of over \$280 billion. In Fall 2000, UC Berkeley was able to admit about 26% of the 33,244 applicants for freshman admission (University of California-Berkeley). In Fall 2000, Stanford University was only able to offer admission to 1168 men from 9571 applications (768 admitted) and 1257 women from 8792 applications (830 admitted), a general admit rate of 13% (Stanford University Admission).

The UC Berkeley campus admits its freshman class on the basis of an assessment of the applicants' high school academic performance (approximately 50%) and through a comprehensive review of the application including personal achievements of the applicant (approximately 50%) (University of California-Berkeley). For Fall 1999, the average weighted GPA of an admitted freshman was 4.16, with a SAT I verbal score range of 580-710 and a SAT I math score range of 620-730 for the middle 50% of admitted students (University of California-Berkeley). While there is no specific GPA for UC Berkeley applicants that will guarantee admission, a GPA of 2.8 or above is required for California residents and a test score total indicated in the University's Freshman Eligibility Index must be achieved. A minimum 3.4 GPA in A-F courses is required for non-residents. At Stanford University, most of the candidates have an un-weighted GPA between 3.6 and 4.0 and verbal SAT I and math SAT I scores of at least 650 (Stanford

University Admission) At UC Berkeley, the academic assessment includes student's academic performance and several measured factors such as:

- College preparatory courses
- Advanced Placement (AP)
- International Baccalaureate Higher Level (IBHL)
- Honors and college courses beyond the UC minimum and degree of achievement in those courses
- Uncapped UC GPA
- Pattern of grades over time
- Scores on the three required SAT II tests and the SAT I (or ACT)
- Scores on AP or IBHL exams
- Honors and awards which reflect extraordinary, sustained intellectual or creative achievement
- Participation in rigorous academic enrichment
- Outreach programs
- Planned twelfth grade courses
- Qualification for UC Eligibility in the Local Context

All freshman applicants must complete courses in the University of California's A-F subject pattern and present scores from SAT I (or ACT) and SAT II tests with the following required subjects:

- a. History/Social Science - 2 years required
- b. English - 4 years required
- c. Mathematics - 3 years required, 4 recommended
- d. Laboratory Science - 2 years required, 3 recommended
- e. Language Other than English - 2 years required, 3 recommended
- f. College Preparatory Electives - 2 years required

At Stanford University, in addition to the academic transcript, close attention is paid to other factors such as student's written application, teacher references, the short responses and one-page essay (carefully read for quality, content, and creativity), and personal qualities.

The information provided in this study is a hypothetical situation and does not reflect the current UC system or Stanford University admissions criteria. However, we use this information to build a model to represent a real admissions problem. For more detailed information regarding University admissions, please refer to the University of California-Berkeley and Stanford University, Office of Undergraduate Admission (University of California-Berkeley; Stanford University Admission).

Given the factors and general admission *criteria*, a simulated-hypothetical model (a Virtual Model) was developed. A series of excellent, very good, good, not good, not bad, bad, and very bad student given the criteria for admission has

been recognized. These criteria over time can be modified based on the success rate of students admitted to the university and their performances during the first, second, third and fourth years of their education with different weights and degrees of importance given for each year. Then, fuzzy similarity and ranking can evaluate a new student rating and find it's similarity to a given set of criteria.

Figure 9 shows a snapshot of the software developed for university admissions and the evaluation of student applications. Table 7 shows the granulation of the variables that was used in the model. To test the performance of the model, a demo version of the software is available at: <http://zadeh.cs.berkeley.edu/> (Nikravesh 2001a). Incorporating an electronic intelligent knowledge-based search engine, the results will eventually be in a format to permit a user to interact dynamically with the contained database and to customize and add information to the database. For instance, it will be possible to test an intuitive concept by dynamic interaction between software and the human mind.

Figure 9. A snapshot of the software for University Admission Decision Making.

This will provide the ability to answer "What if?" questions in order to decrease uncertainty and provide a better risk analysis to improve the chance for "increased success" on student selection or it can be used to select students on the basis of "diversity" criteria. The model can be used as for decision support and for a more uniform, consistent and less subjective and biased way. Finally, the model could

learn and provide the mean to include the feedback into the system through time and will be adapted to the new situation for defining better criteria for student selection.

In this study, it has been found that ranking and scoring is a very subjective problem and depends on user perception and preferences in addition to the techniques used for the aggregation process which will effect the process of the data mining in reduced domain. Therefore, user feedback and an interactive model are recommended tools to fine-tune the preferences based on user constraints. This will allow the representation of a multi-objective optimization with a large number of constraints for complex problems such as credit scoring or admissions. To solve such subjective and multi-criteria optimization problems, GA-fuzzy logic and DNA-fuzzy logic models are good candidates. In the case of the GA-Fuzzy logic model, the fitness function will be defined based on user constraints. For example, in the admissions problem, assume that we would like to select students not only on the basis of their achievements and criteria, but also on the basis of diversity which includes gender distribution, ethnic background distribution, geophysical location distribution, etc. The question will be "what are the values for the preferences and which criteria should be used to achieve such a goal?" In this case, we will define the genes as the values for the preferences and the fitness function will be defined as the degree by which the distribution of each candidate in each generation match the desired distribution. Fuzzy similarity can be used to define the degree of match, which can be used for better decision analysis.

Now, the question will be "what are the values for the preferences and which criteria should be used to achieve such a goal?"

- Given a set of successful students, we would like to adjust the preferences such that the model could reflect this set of students.
- Diversity, which includes gender distribution, ethnic background distribution, geophysical location distribution, etc.

To solve such subjective and multi-criteria optimization problems with a large number of constraints for complex problems such as University Admissions, the BISC Decision Support System is an excellent candidate.

8.3 Date Matching

The main objective is to find the best possible match in the huge space of possible outputs in the databases using the imprecise matching such as fuzzy logic concept, by storing the query attributes and continuously refining the query to update the

user's preferences. We have also built a Fuzzy Query system, which is a Java application that sits on top of a database.

With traditional SQL queries (relational DBMS), one can select records that match the selection criteria from a database. However, a record will not be selected if any one of the conditions fails. This makes searching for a range of potential candidates difficult. For example, if a company wants to find an employee who is proficient in skill A, B, C and D, they may not get any matching records, only because some candidates are proficient in 3 out of 4 skills and only semi-proficient in the other one. Since traditional SQL queries only perform Boolean matching, some qualities of real life, like "far" or "expensive" or "proficient", which involve matters of degree, are difficult to search for in relational databases. Unlike Boolean logic, fuzzy logic allows the degree of membership for each element to range over an interval. So in a fuzzy query, we can compute how similar a record in the database is to the desired record. This degree of similarity can be used as a ranking for each record in the database. Thus, the aim of the fuzzy query project for date matching is to add the capability of imprecise querying (retrieving similar records) to traditional DBMS. This makes some complex SQL statements unnecessary and also eliminates some repetitious SQL queries (due to empty-matching result sets).

In this program, one can basically retrieve all the records from the database, compare them with the desired record, aggregate the data, compute the ranking, and then output the records in the order of their rankings. Retrieving all the records from the database is a naïve approach because with some preprocessing, some very different records are not needed from the database. However, the main task is to compute the fuzzy rankings of the records so efficiency is not the main concern here.

The major difference between this application and other date matching system is that a user can input his hobbies in a fuzzy sense using a slider instead of choosing crisp terms like "Kind of" or "Love it". These values are stored in the database according to the slider value, **Figures 10, 11**.

The screenshot shows the 'Date Matching' input form. The form is titled 'Date Matching' and features a pink background with a heart shape. It includes various input fields for personal and professional information, as well as sliders for 'Degree of Importance' for each field.

Input fields include:

- Gender: Male (dropdown)
- Age: Young (dropdown)
- Height: 188 cm (text input)
- Weight: 20 kg (text input)
- Body: Normal (dropdown)
- Education: College Grad (dropdown)
- Industry: Food (dropdown)
- Income: 60000 USD (text input)

Sliders for 'Degree of Importance' are shown for each field, with numerical values on the right:

- Age: 20
- Height: 43
- Weight: 71
- Body: 27
- Education: 51
- Industry: 52
- Income: 80

Additional sliders are shown for 'HMM, Inc' and 'Degree of Importance' for 'Smoking', 'Alcohol', 'Music', and 'Movies'.

Figure 10. Date matching input form

The screenshot shows the 'Date Matching Search Results' page. The page is titled 'Date Matching Search Results' and features a pink background with a heart shape. It includes a table of search results.

Table with 15 columns: Username, Name, Email, Gender, Age, Body, Height, Weight, Education, Industry, Income, Ranking, Alcohol, Music, Movies, and a final unlabeled column. The table contains three rows of results, with the first row having the highest ranking.

Username	Name	Email	Gender	Age	Body	Height	Weight	Education	Industry	Income	Ranking	Alcohol	Music	Movies	Smoking	Spans	Phenograph
<input type="checkbox"/>	vikhan	vikhan	Male	20.0	Normal	188.0	20.0	College Grad	Food	60000.0	50.0	50.0	50.0	50.0	50.0	50.0	50.0
<input type="checkbox"/>	vikhan	vikhan	Male	20.0	Normal	170.0	85.0	College Grad	Food	50000.0	10.0	50.0	40.0	80.0	80.0	80.0	80.0
<input type="checkbox"/>	vikhan	vikhan	Male	40.0	Normal	188.0	60.0	College Grad	Food	50000.0	5.0	10.0	17.0	55.0	20.0	40.0	20.0

Search again

Figure 11 shows the results are obtained from fuzzy query using the search criteria in the previous page. The first record is the one with the highest ranking.

The current date matching software can be modified or expanded in several ways:

1. One can build a server/client version of date-matching engine so that we can use a centralized database and all users around the world can do the matching through the web. The ranking part (computation) can still be done on local machine since every search is different. This can also help reduce the server load.
2. The attributes, granulation models and the “meaning” of the data can be tunable so that the system is more configurable and adaptive to changes.
3. User preference capability can be added to the system. (The notion of “overweight” and “tall” can be different to different people.)
4. The GUI needs to be changed to meet real user needs.
5. One can build a library of fuzzy operators and aggregation functions such that one can choose the operator and function that matches the application.
6. One can instead build a generic fuzzy engine framework, which is tunable in every way to match clients’ needs.
7. The attributes used in the system are not very complete compared to other data matching systems online. However, the attributes can be added or modified with some modification to the program without too much trouble.

We have added a web interface to the existing software and built the database framework for further analysis in user profiling so that users could find the best match in the huge space of possible outputs. We saved user profiles and used them as basic queries for that particular user. Then, we stored the queries of each user in order to “learn” about this user’s preference. In addition, we rewrote the fuzzy search engine to be more generic so that it would fit any system with minimal changes. Administrator can also change the membership function to be used to do searches. Currently, we are working on a new generic software to be developed for a much more diverse applications and to be delivered as stand alone software to both academia and businesses.

9 Evolutionary Computing

In the Evolutionary Computing (EC) module of the BISC Decision Support System, our purpose is to use an evolutionary-based method to allow automatic adjusting of the user’s preferences. These preferences can be seen as parameters of the fuzzy logic model in form of degrees of importance of the used variables. Also, they can be extended to a representation of the way the variables have to be combined. In the fuzzy logic model, the variables are combined using aggregation

operators with eventually associated weights, which correspond to their degrees of importance. These operators and weights can be fixed based on the application expert knowledge. However, the application expert might need help to make decision regarding the choice of the aggregators and the variables' weighting which constitute the model's parameters. In this case, we are faced with an optimization problem and our EC module whose role consists in learning these parameters process, has to answer the following question: how to aggregate the variables and with which degrees of importance?

In a first stage, we propose to limit user's preferences to the variables weighting and to use genetic algorithms as learning technique. The corresponding model will be a weighted aggregator for which weights have to be determined by the GA. However, the fuzzy logic model could need a more complex combination of variables using weighted multi-aggregation operators. In this case, the learning process has to select automatically the appropriate aggregators for a given application according to some corresponding training data and to define the way they have to be combined. For this purpose, we propose to use a multi-aggregation model combining weighted aggregators in form of decision tree. In the Evolutionary computing approach, genetic programming, which is an extension of genetic algorithms, is the closest technique to our purpose. It allows us to learn a tree structure that represents the combination of aggregators. Selection of these aggregators is included in the genetic programming based learning process.

Genetic algorithms and genetic programming will be first introduced in the next section. Then, their adaptation to our decision system will be described.

9.1 Genetic algorithms and genetic programming

Introduced by John Holland (Holland 1992), Genetic Algorithms (GAs) constitute a class of stochastic searching methods based on the mechanism of natural selection and genetics. They have recently received much attention in a number of practical problems notably in optimization problems as machine learning processes (Banzhaf 1998).

Basic description

To solve an optimization problem, usually we need to define the search method looking for the best solution and to specify a measure of quality that allows to compare possible solutions and to find the best one. In GAs, the search space corresponds to a set of individuals represented by their DNA. These individuals are evaluated by a measure of their quality called fitness function which has to be defined according to the problem itself. The search method consists in an evolutionary process inspired by the Darwinian principle of reproduction and survival of the fittest individual.

This evolutionary process begins with a set of individuals called population. Individuals from one population are selected according to their fitness and used to form a new population with the hope to produce better individuals (offspring). The population is evolved through successive generations using genetic operations until some criterion is satisfied.

The evolution algorithm is resumed in **Figure 12**. It starts by creating randomly a population of individuals, which constitute an initial generation. Each individual is evaluated by calculating its fitness. Then, a selection process is performed based on their fitness in order to choose individuals that participate to the evolution. Genetic operators are applied on these individuals to produce new ones. A new generation is then created by replacing existing individuals in the previous generation by the new ones. The population is evolved by repeating individuals' selection and new generations creation until the end criterion is reached in which case the evolution is stopped.

The construction of a GA for any problem can be separated into five tasks:

- Choice of the representation of the individuals,
- Design of the genetic operators,
- Determination of the fitness function and the selection process,
- Determination of parameters and variables for controlling the evolution algorithm,
- Definition of the termination criterion.

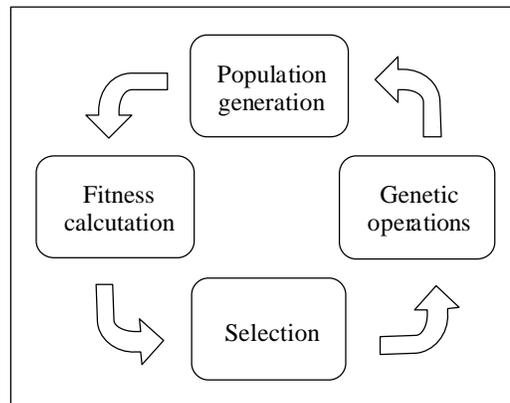


Figure 12 Genetic Algorithm Cycle

In the conventional GAs, individuals' DNA is usually represented by fixed-length character strings. Thus in this case, the DNA encoding requires a selection of the string length and the alphabet size. Binary strings are the most common encoding because its relative simplicity. However, this encoding might be not natu-

ral for many problems and sometimes corrections must be made on the strings provided by genetic operations. Direct value encoding can be used in problems where use of binary encoding would be difficult. In the value encoding, an individual's DNA is represented by a sequence of some values. Values can be anything connected to the problem, such as (real) numbers.

Genetic operators

The evolution algorithm is based on the reproduction of selected individuals in the current generation breeding a new generation composed of their offspring. New individuals are created using either sexual or asexual reproduction. In sexual reproduction, known as crossover, two parents are selected and DNA from both parents is inherited by the new individual. In asexual reproduction, known as mutation, the selected individual (parent) is simply copied, possibly with random changes.

Crossover operates on selected genes from parent DNA and creates new offspring. This is done by copying sequences alternately from each parent and the points where the copying crosses is chosen at random. For example, the new individual can be bred by copying everything before the crossover point from the first parent and then copy everything after the crossover point from the other parent. This kind of crossover is illustrated in **Figure 13** for the case of binary string encoding. There are other ways to make crossover, for example by choosing more crossover points. Crossover can be quite complicated and depends mainly on the encoding of DNA. Specific crossover made for a specific problem can improve performance of the GA.

Mutation is intended to prevent falling of all solutions in the population into a local optimum of the solved problem. Mutation operation randomly changes the offspring resulted from crossover. In case of binary encoding we can switch a few randomly chosen bits from 1 to 0 or from 0 to 1 (see **Figure 14**). The technique of mutation (as well as crossover) depends mainly on the encoding of chromosomes. For example when permutations problem encoding, mutation could be performed as an exchange of two genes.

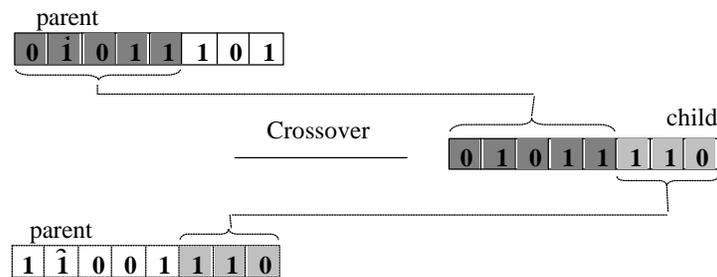


Figure 13 Genetic Algorithm - Crossover

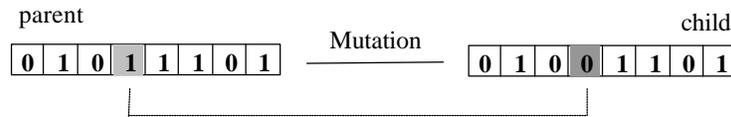


Figure 14 Genetic Algorithm - Mutation

Selection process

Individuals that participate in genetic operations are selected according to their fitness. Even though the main idea is to select the better parents in the hope that they will produce better offspring, the problem of how to do this selection remains. This can be done in many ways. We will describe briefly some of them. The (μ, λ) selection consists in breeding λ offspring from μ parents and then μ offspring will be selected for the next generation. In the Steady-State Selection, in every generation a few good (with higher fitness) individuals are selected for creating new offspring. Then some bad (with lower fitness) individuals are removed and replaced by the new offspring. The rest of population survives to new generation. In the tournament selection, a group of individuals is chosen randomly and the best individual of the group is selected for reproduction. This kind of selection allows giving a chance to some weak individual in the population, which could contain good genetic material (genes) to participate to reproduction if it is the best one in its group. Elitism selection aims at preserving the best individuals. So it first copies the best individuals to the new population. The rest of the population is constructed in ways described above. Elitism can rapidly increase the performance of GA, because it prevents a loss of the best-found solution.

Parameters of GA

The outline of basic GA is very general. There are many parameters and settings that can be implemented differently in various problems. One particularly important parameter is the population size. On the one hand, if the population contains too few individuals, GA has few possibilities to perform crossover and only a small part of search space is explored. On the other hand, if there are too many individuals, GA slows down. Another parameter to take into account is the number of generations, which can be included in the termination criterion.

For the evolution process of the GA, there are two basic parameters: crossover probability and mutation probability. The crossover probability indicates how often crossover will be performed. If there is no crossover, offspring are exact copies of parents. If there is crossover, offspring are made from parts of both parent's DNA. Crossover is made in hope that new chromosomes will contain good parts of old chromosomes and therefore the new chromosomes will be better. However, it is desirable to leave some part of the old population to survive into the next gen-

eration. The mutation probability indicates how often parts of chromosomes will be mutated. If there is no mutation, offspring are generated immediately after crossover (or directly copied) without any change. If mutation is performed, one or more parts of a chromosome are changed.

Genetic programming

Genetic programming (GP) is a technique pioneered by John Koza (Koza 1992), which enables computers to solve problems without being explicitly programmed. It is an extension of the conventional GA in which each individual in the population is a computer program. It works by using GAs to automatically generate computer programs that can be represented as linear structures, trees or graphs. Tree encoding is the most used form to represent the programs. Tree structure is composed of primitive functions and terminals appropriate to the problem domain. The functions may be arithmetic operations, programming commands, mathematical logical or domain-specific functions. To apply GP to a problem, we have to specify the set functions and terminals for the tree construction. Also, besides the parameters of the conventional GA, other parameters which are specific to the individual representation can be considered such as tree size, as an example.

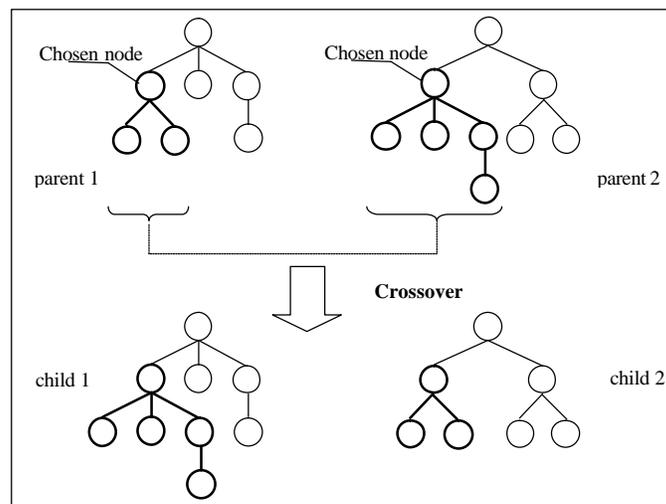


Figure 15 Genetic programming - Tree-encoding individual crossover.

Genetic operations are defined specifically for the type of encoding used to represent the individuals. In the case of tree encoding, new individuals are produced by removing branches from one tree and inserting them into another. This simple process ensures that the new individual is also a tree and so is also syntactically valid. The crossover and mutation operations are illustrated in **figures 15-16**. The mutation consists in randomly choosing a node in the selected tree, creating a

new individual and replacing the sub-tree rooted at the selected node by the created individual. The crossover operation is performed by randomly choosing nodes in the selected individuals (parents) and exchanging the sub-trees rooted at these nodes, which produce two new individuals (offspring).

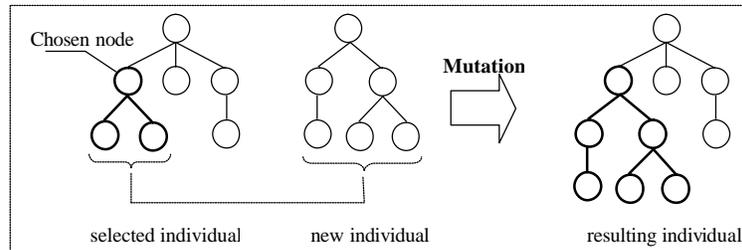


Figure 16 Genetic programming - Tree-encoding individual mutation

User's preferences learning using EC

We have introduced GA and GP in a previous section. In this section, we will proceed to describe their adaptation to our problem. Our aim is at learning the fuzzy-DSS parameters which are: 1) the weight vector (representing the user preferences is associated with the variables) that must be aggregated and, 2) the adequate decision tree (representing the combination of the aggregation operators) that have to be used.

Weights learning using GA

Weight vector being a linear structure, can be represented by a binary string in which weight values are converted to binary numbers. This binary string corresponds to the individual's DNA in the GA learning process. The goal is to find the optimal weighting of variables. A general GA module can be used by defining a specific fitness function for each application as shown in **Figure 17**.

Let's see the example of the University Admissions application. The corresponding fitness function is shown **Figure 18**. The fitness is computed based on a training data set composed of vectors $\bar{x}_1, \dots, \bar{x}_N$ of fuzzy values (x_{1j}, \dots, x_{kj}) for each \bar{x}_i . Each value of a fuzzy variable is constituted of a crisp value between 0 and 1 and a set of membership functions. During the evolution process, for each weighting vector (w_1, w_2, \dots, w_k) , the corresponding fitness function is computed as follows. Using these weights, a score is calculated for each vector. Afterward, these scores are ranked and compared with the actual ranking using similarity measure.

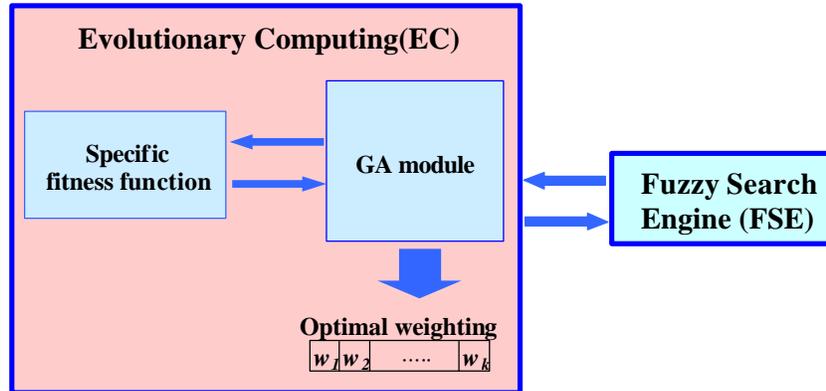


Figure 17 Evolutionary Computing Module: preferences learning.

Let's assume that we have N students and the goal is to select among them n students that will be admitted. Each student is then represented by value vector in the training data set. The similarity measure between the computed and the actual ranking could be the intersection between the n top vectors, which has to be maximized. We can also consider the intersection on a larger number $n_1 > n$ of top vectors. This measure can be combined to the first one with different degrees of importance. In this case, the Fitness value will be a weighted sum of these two similarity measures.

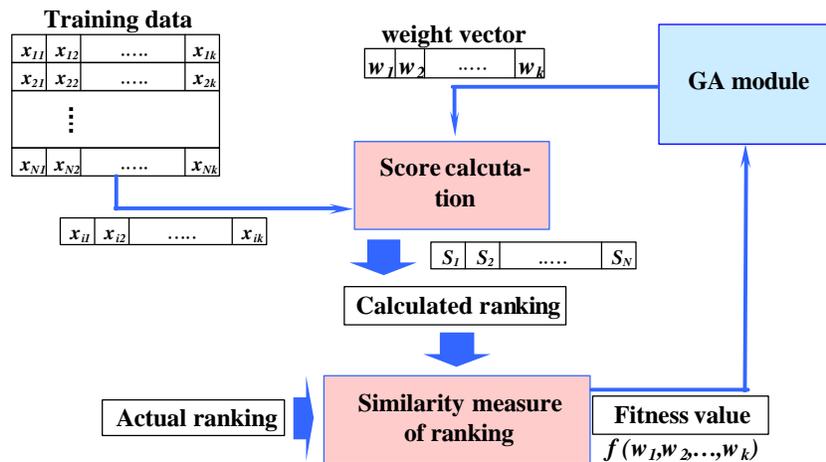


Figure 18 EC Module: Specific fitness function for the “University Admissions Application”.

Aggregation tree learning using GP

We have seen the learning of the weights representing the user preferences regarding the fuzzy variables. However, the aggregators that are used are fixed in the application or chosen by the user. But it is more interesting to adjust these aggregators automatically. We propose to include this adjustment in the GA learning process.

Aggregators can be combined in the form of tree structure, which can be built using a Genetic Programming learning module. It consists in evolving a population of individuals represented by tree structures. The evolution principle remains the same as in a conventional GP module but the DNA encoding needs to be defined according to the considered problem. We propose to define an encoding for aggregation trees which is more complex than for classical trees and which is common to all considered applications. As shown in **Figure 19**, we need to define a specific encoding in addition to the fitness function specification.

We need to specify the functions (tree nodes) and terminals that are used to build aggregation trees. Functions correspond to aggregation operators and terminals (leaves) are the fuzzy variables that have to be aggregated. Usually, in GP the used functions have a fixed number of arguments. In our case, we prefer not to fix the number of arguments for the aggregators. We might however define some restrictions such as specifying minimal and maximal number of arguments. These numbers can be considered as parameters of the learning process. This encoding property allows a largest search space to solve our problem. Moreover, instead of finding weights only for the fuzzy variables, we have to fix them also at each level of their hierarchical combination, which allows using weighted aggregation operators in the whole structure.

Tree structures are generated randomly as in the conventional GP. But, since these trees are augmented according to the properties defined above, the generation process has to be updated. So, we decided to randomly generate the number of arguments when choosing an aggregator as a node in the tree structure. And for the weights, we chose to generate them randomly for each node during its creation.

Concerning the fitness function, it is based on performing the aggregation operation at the root node of the tree that has to be evaluated. For the university admissions application, the result of the root execution corresponds to the score that has to be computed for each value vector in the training data set. The fitness function, as in the GA learning of the user preferences, consists in simple or combined similarity measures. In addition, we can include to the fitness function a complementary measure that represents the individual's size, which has to be minimized in order to avoid over-sized trees.

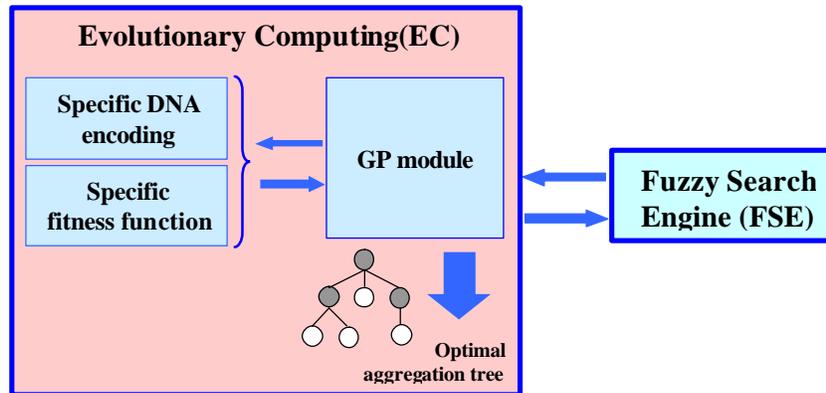


Figure 19 Evolutionary Computing Module: aggregation tree learning.

10 Conclusion

In this study, we introduced fuzzy query and fuzzy aggregation and the BISC decision support system as an alternative for ranking and predicting the risk for credit scoring, university admissions, and several other applications, which currently utilize an imprecise and subjective process. The BISC decision support system key features are 1) intelligent tools to assist decision-makers in assessing the consequences of decision made in an environment of imprecision, uncertainty, and partial truth and providing a systematic risk analysis, 2) intelligent tools to be used to assist decision-makers answer “What if Questions”, examine numerous alternatives very quickly and find the value of the inputs to achieve a desired level of output, and 3) intelligent tools to be used with human interaction and feedback to achieve a capability to learn and adapt through time. In addition, the following important points have been found in this study 1) no single ranking function works well for all contexts, 2) most similarity measures work about the same regardless of the model, 3) there is little overlap between successful ranking functions, and 4) the same model can be used for other applications such as the design of a more intelligent search engine which includes the user's preferences and profile (Nikravesh 2001a, 2001b).

Acknowledgement

Funding for this research was provided by the British Telecommunication (BT) and the BISC Program of UC Berkeley.

References

- (Banzhaf et al 1998) Banzhaf, W. (1998) Nordin, P., Keller, R.E. and Francone, F.D., Genetic Programming : An Introduction On the Automatic Evolution of Computer Programs and Its Applications, dpunkt.verlag and Morgan Kaufmann Publishers, San Francisco, CA, USA, 1998, 470 pages.
- (Detyniecki 2000) Detyniecki, M., Mathematical Aggregation Operators and their Application to Video Querying, Ph.D. thesis, University of Paris VI, 2000.
- (Fagin 1998) Fagin, R. Fuzzy Queries in Multimedia Database Systems, Proc. ACM Symposium on Principles of Database Systems, 1998, pp. 1-10.
- (Grabisch 1996) Grabisch, M., K-order additive fuzzy measures. In Proc of 6th intl Conf on Information Processing and Management of Uncertainty in Knowledge-based Systems, Spain, 1996, pp 1345-50.
- (Grabisch et al 2000) Grabisch, M., Murofushi, T., Sugeno, M. Fuzzy Measures and Integrals: Theory and Applications, Physica-Verlag, NY, 2000.
- (Holland 1975) Holland, J. H., Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. MIT Press, 1992. First Published by University of Michigan Press 1975.
- (Kohonen 1987) Kohonen, T. Self-Organization and Associate Memory, 2nd Edition, Springer Verlag, Berlin, 1987.
- (Koza 1992) Koza, J. R. Genetic Programming : On the Programming of Computers by Means of Natural Selection, Cambridge, Mass. : MIT Press, USA 1992, 819 pages.
- (Loia 2002) Loia, V. et al., Fuzzy Logic and the Internet, *Journal of Soft Computing*, Special Issue, Springer Verlag, Vol. 6, No. 5; August 2002.
- (Loia et al 2003) Loia, V. et al., "Fuzzy Logic an the Internet", to be published in the Series Studies in Fuzziness and Soft Computing, Physica-Verlag, Springer, August 2003.
- (Mizumoto 1989) Mizumoto, M., Pictorial Representations of Fuzzy Connectives, Part I: Cases of T -norms, T-conorms and Averaging Operators, *Fuzzy Sets and Systems* (31): pp. 217-242, 1989.
- (Murofushi and Sugeno 1989) Murofushi, T. and Sugeno, M., An interpretation of fuzzy measure and the Choquet integral as an integral with respect to a fuzzy measure. *Fuzzy Sets and Systems*, (29): pp 202-27, 1989.
- (Nikraves 2001a) Nikraves, M., Perception-based information processing and retrieval: application to user profiling, 2001 research summary, EECS, ERL, University of California, Berkeley, BT-BISC Project. (<http://zadeh.cs.berkeley.edu/> & <http://www.cs.berkeley.edu/~nikraves/> & <http://www-bisc.cs.berkeley.edu/>), 2001.
- (Nikraves 2001b) Nikraves, M., Credit Scoring for Billions of Financing Decisions, Joint 9th IFSA World Congress and 20th NAFIPS International Conference. IFSA/NAFIPS 2001 "Fuzziness and Soft Computing in the New Millenium", Vancouver, Canada, July 25-28, 2001.
- (Nikraves 2002) Nikraves M., Fuzzy Conceptual-Based Search Engine using Conceptual Semantic Indexing, NAFIPS-FLINT, June 27-29, New Orleans, LA, USA 2002.
- (Nikraves and Azvine 2001) Nikraves, M. and Azvine, B., FLINT 2001, New Directions in Enhancing the Power of the Internet, UC Berkeley Electronics Research Laboratory, Memorandum No. UCB/ERL M01/28, August 2001.
- (Nikraves and Azvine 2002) Nikraves, M. and Azvine, B., Fuzzy Queries, Search, and Decision Support System, *Journal of Soft Computing*, Volum 6, # 5, August 2002.

-
- (Nikravesh and Choi 2003) Nikravesh, M. and Choi, D-Y., Perception-Based Information Processing, UC Berkeley Electronics Research Laboratory, Memorandum No. UCB/ERL M03/20, June 2003.
- (Nikravesh et al 2003a) Nikravesh, M., Azvine, B., Yagar, R. and Zadeh, L. A. (2003) "New Directions in Enhancing the power of the Internet", to be published in the Series Studies in Fuzziness and Soft Computing, Physica-Verlag, Springer, August 2003.
- (Nikravesh et al. 2002) Nikravesh, M. et al., Fuzzy logic and the Internet (FLINT), Internet, World Wide Web, and Search Engines, *Journal of Soft Computing*, Special Issue; fuzzy Logic and the Internet, Springer Verlag, Vol. 6, No. 5; August 2002.
- (Nikravesh et al. 2003b) Nikravesh, M. et al., Perception-Based Decision processing and Analysis, UC Berkeley Electronics Research Laboratory, Memorandum No. UCB/ERL M03/21, June 2003.
- (Nikravesh et al. 2003c) Nikravesh, M. et al., Web Intelligence: Conceptual-Based Model, UC Berkeley Electronics Research Laboratory, Memorandum No. UCB/ERL M03/19, June 2003.
- (Sugeno 1974) Sugeno, M. Theory of fuzzy integrals and its applications. Ph.D. Dissertation, Tokyo Institute of Technology, 1974.
- (Takagi et al. 2002a), Takagi, T. et al. Exposure of Illegal Website using Conceptual Fuzzy Sets based Information Filtering System, the North American Fuzzy Information Processing Society - The Special Interest Group on Fuzzy Logic and the Internet NAFIPS-FLINT 2002, 327-332 (2002a)
- (Takagi et al. 2002b), Takagi, T. et al Conceptual Fuzzy Sets-Based Menu Navigation System for Yahoo!, the North American Fuzzy Information Processing Society - The Special Interest Group on Fuzzy Logic and the Internet NAFIPS-FLINT 2002, 274-279 (2002b)
- (Yager 1988) Yager, R. On ordered weighted averaging aggregation operators in multi-criteria decision making, IEEE transactions on Systems, Man and Cybernetics, (18): 183-190, 1988.