ARAB REPUBLIC OF EGYPT

THE INSTITUTE OF NATIONAL PLANNING

Memo No (1642)

Genetic Algorithms and Programming Introducing the concept and Domain Of Applications

Prepared by

Prof.Dr.M.ELKAFRAWY
Prof.of Operations Research,
Institute of National Planning
&
Prof.Dr.I.A.ISMAIl
Dean of the Faculty of
Computers& informatics,
MISR International Univ.

يوليه ٢٠١٠

CAIRO
SALAH SALEM St-NASR CITY

Genetic Algorithms and Programming

Introducing the Concept and Domain of Applications

By

Prof.Dr.M.ELKAFRAWY
Prof.of Operations Research,
Institute of National Planning

&

Prof .Dr.I.A.ISMAIl
Deanof the faculty of
computers & informatics,
MISR International Univ.

CONTENTS

When there is nothing to do

When you are bored to the bone

When you are missing your friends

When you are feeling alone

It's time to start studying Mathematics.

1. Introduction

On these notes we introduce the most important modern methods that are in use today. These notes comprise some applications to the genetic algorithms together with the written programs for each of these applications.
There are of course much more topics for applying the genetic algorithms than the ones explained here. Nevertheless, once, we get hold of the main concepts presented here we would be capable of applying the GA ideas to more and more applications.
The topics dealt with in these notes are :-

1- Genetic algorithms and related topics (1.2).
2- Solving game theory optimization problems using genetic algorithms.
3- Solving systems of linear equations using genetic algorithms.
4-Solving unconstrained minimization problems using genetic algorithms.
5- Image registration using genetic algorithms.

We then move to the next important topic which are genetic programming.

One give a general description of the method.

Programming and applications of this technique are solvable using GA methods,

2. *Genetic algorithms and related items*

2.1 Introduction : Genetic algorithms is a technique based on simulating nature. On the other hand, nature runs according to the biological rule, survival for the fittest found out by Darwen. To mimic and simulate on a computer, we follow the same rule of the survival for the fittest. This is done by specifying a fitness function, representing each argument of the function by a chromosome represented by a string of bits having a specified length.

To find which string is superior to the other, a number of operations is done on the strings to arrive at the best string that gives the best or the optimum fitness function.

Those operations are explained as follows:

(1) Reproduction Process:

The best off springs are chosen from the given net of strings each with the best values for the fitness function.

(2) The cross-over process:

Assuming that we have two chromosomes under consideration

a_1	a_2	a_3	a_4	a_5	a_6	.....	a_{16}

And

b_1	b_2	b_3	b_4	b_5	b_6	.....	b_{16}

and it is required to do the mating between them. To do that, we do the following :

(i) Decide on the point at which the cut and interchange is to be done.

(ii) Replace the bits to the right of that point in string 1, by there to the right of the same point in string 2.

So that the two new strings will look as shown bellow:

a_1	a_2	a_3	a_4	b_5	b_6	.....	b_{16}

b_1	b_2	b_3	b_4	a_5	a_6	.....	a_{16}

(iii) Mutation :

After a large number of iterations, we can apply the mutation process not to all strings under consideration but to some specified percentage and not to whole string but to some specified percentage of this string.

This mutation is explained by the following example.

Having a string of the following form :

0	1	1	1	0	1	1	0	1	1

If we are doing 10% mutation to this string , we flip only one of its bits from 0 to 1 or from 1 to 0. then

0	0	1	1	0	1	1	0	1	1

Is a 10% mutation

Now, the question that may jump to your mind, what has this to do with data mining.

Genetic algorithms and data mining:

It is required to use a genetic algorithm to reach a base to discover some relationships between data items. For example, given a database, the aim is to determine the four items, that are most often bought together. For example if one buys spaghetti, he also buys minced, tomato and salt. Therefore, the data base and the genetic algorithm tells us what the related items are.

2.2 Synthetic Data Sets and Data Set Generation

Since the focus of this article is on the development and verification of a genetic algorithm-based data mining search approach, an actual database with " real" data was not used nor was it required to demonstrate the usefulness of the genetic algorithm. Instead, " Synthetic" data were generated to demonstrate the application of the genetic algorithm. As described earlier, a synthetically generated data set consists of X transactions. Each transaction consists of a transaction number (Trans-Num), the number of items purchased in the transaction (N), and a list of N items, where each item is specified by a number (i.e.0=Pretzels, 1=Aspirin, 2=Beer, 3=Bread, etc.). Also, N can be any of 100 different items. The format of a transaction and an example of a transaction containing fifteen items are shown below:

Transaction Format : **Trans - Num N** $Item_0$ $Item_1$ ..... $Item_N$

Example Transaction : **01512310195542412283202087926**

Synthetic data sets were generated using a C Program that allowed the user, through command line input, to specify data characteristics. Allowing the user control over data set characteristics facilitated verification of the genetic algorithm search results, since known relationships were embedded within the data sets. Then, the relationships that were discovered by the genetic search were confirmed against the expected results.

The synthetic data generator program required the user to provide at least one option to specify the number of transactions to generate. Other options allowed the user to specify individual items and the probability with which each of the specified items appeared within the synthetically generated data set.

2.3 GENETIC ALGORITHM SPECIFICS

The coding scheme, fitness function, and each of the three genetic operators (reproduction, crossover, and mutation) can be implemented in a variety of ways depending on the problem to which a genetic algorithm is being applied. Various implementation alternatives of these for the data mining problem examined in this article are discussed below.

2.4 Parameter Coding

A main difference between genetic algorithms and more traditional optimization and search algorithms is that genetic algorithms work with a coding of the parameter set and not the parameters themselves. Thus, before any type of genetic search can be performed, a coding scheme must be determined to represent the parameters in the problem at hand. In the data mining problem addressed by this article, the parameters of interest are simply four, potentially related, item numbers. Therefore, a coding scheme for four item numbers.
was determined considering the following factors:

- A multi-parameter coding, consisting of four sub-strings, is required to code each of the four items into a single string.
- Each sub-string needs to represent one of a hundred(0 through 99) possible items
- There are 100 choose 4 (3,921,225) possible combinations of items that need to be represented.

Given these factors, several coding schemes were considered. First, a scheme using a base-10 coded sub-string was examined. This coding allows 100 items to be represented, and no codings correspond to non-existent items (all possible sub-string codings would represent valid item numbers, 0 through 99). This coding results in an 8-digit string where each digit (d_x) is between 0 and 9, as follows:

$d_1 d_0$	$d_1 d_0$	$d_1 d_0$	$d_1 d_0$
item 1	item 2	item 3	item 4

A coding of this sort, however, limits the number of schemata that are available for the genetic algorithm to exploit. Therefore, since we want to maximize the number of schemata, a binary coding, as shown below was considered.

$b_6 b_5 b_4 b_3 b_2 b_1 b_0$	$b_6 b_5 b_4 b_3 b_2 b_1 b_0$	$b_6 b_5 b_4 b_3 b_2 b_1 b_0$	$b_6 b_5 b_4 b_3 b_2 b_1 b_0$
item 1	item 2	item 3	item 4

Here, each digit (b_x) is a 1 or 0. One problem with this coding is that, in order to represent 100 items, a sub-string must consist of seven bits $((2^7 = 128)$. This means that $28^4 (128 - 100 = 28$ values in each sub-string), or 614,656 strings could be manipulated by the genetic algorithm which don't correspond to actual solutions.

Another possibility, when considering the fact that there are 3,921,225 combinations of items that need to be represented, was to code the items as a 22-bit binary string. This corresponds to 2^{22}, *or* 4,194,304 possible solution representations, reducing the number of non-solution strings to 273,079(4,194,304 - 3,921,225). Using this approach, however, requires a mapping from a " combination number" to the actual four items represented by that combination since the item numbers are not specifically embedded within the string. If this combination-mapping approach was used, it would be difficult for the genetic algorithm to exploit similarities between strings since two combination numbers would not necessarily have anything in common, even if the items represented by the combination numbers were similar. Therefore, this coding approach was deemed unacceptable.

Given these possibilities, and considering the implementation of the crossover operator which is discussed in the section (2.6), "Reproduction, Crossover, and Mutation Operators," the decimal coding appeared to be the best alternative and was therefore implemented for the data mining problem.

2.5 Fitness function

If a string of four items is coded as just described, the fitness for such a string can be determined based on the frequency with which the set of four items appear within the transactions contained in the database under investigation. In addition, if a fractional number of the items in the item list appear within a transaction, the fitness value will also reflect that fraction. For example, if the item list for which a fitness value is to be determined is as follows :

01	22	68	07

And if the database, for the purpose of this discussion, contains only two transactions, as follows :

transaction 1 : 34 22 55 01 68 99 02 07 42 24
transaction 2 : 76 01 86 99 02 07 42 24

Then transaction 1 contains all four items (01,22,68,and 07)and transaction 2 contains only two of the four items (01 and 07). If $I(t_i)$ represents the fraction of items that are contained within a single transaction t_i then $I(t_1)$ would equal 1.0 and $I(t_2)$ would equal 0.5 (2/4). If all the $I(t_i)$s are summed over all transactions, and this sum is divided by the total number of transaction, N, then a relationship (referred to as the base fitness) between the item set and the frequency with which the items appear within the database can be obtained. Thus, the base fitness value for the item combination of 01, 22, 68, and 07 for the example database of two transactions would be:

$$(I(t_1) + I(t_2))/N \Rightarrow \qquad (1.0 + 0.5)/2 = 0.75$$

This yields a base fitness function that can be represented as follows:

Assume:

N = total number of transactions in the database file (sdg. db)

t_i = transaction number

$I(t_i)$ = fraction of items that are contained in (t_i)

M = sum of all $I(t_i)$ over all transactions

Then M is :

$$M = \sum_{i=1}^{N} I(t_i)$$

and the fitness, referred to as F1, for a given string of items is :

$$F1 = base\ \ fitness\ \ (item\ \ string) = \frac{M}{N}$$

Furthermore, in an effort to avoid possible convergence, and to promote competition between strings throughout a simulation, a scaled fitness, $F2$, was implemented and tested. $F2$ is expressed as,

$$F2 = a^{*} F1 + b$$

where F2 is the scaled fitness, F1 is the raw fitness, and a and b are determined based on the maximum and average raw fitness values. In the case of the data mining problem, the maximum possible fitness (if every transaction in the data base contained all four items of interest)was 1.0 and the average raw fitness was found to be around 0.4.

In a third alternative, the fitness function was based on the following additional assumptions : Since the goal is to determine the four items that are most often purchased together, the fitness function should yield a higher value if more of the items in the item set are contained together within the database transactions. For example, if one item set yielded a base fitness value of 0.25 (implying that, on average, one of the four items in the item set were contained in each of the database transactions)and a second item set yielded a base fitness value of 0.75 (implying that, on average, three of the four items in the item set were contained in each of the database transactions), it may be desirable to give more than a simple linear emphasis to the second item set value since it is closer to the goal of four. Thus, a third fitness, referred to as F3, is expressed as

$$F3 = \frac{M^{x}}{N}$$

where x was chosen to be 3.

While this type of fitness expression gives more emphasis to item sets that are closer to the four item goal, it must be realized that a single occurrence of four items purchased together within a large database will not yield a higher fitness than three of the items purchased together

many times. Since we are seeking trends within a large set of database transactions, it was felt to be more important to recognize major trends (such as three items purchased together many times) which might lead to a trend of four items purchased together.

The results of simulations run with each of the fitness expressions just described (F1,F2, and F3) are discussed in section, " Fitness Function Simulation Results." Those results show that fitness function F3 provided very good performance, better than both F1 and F2.

2.6 Reproduction, Crossover, and Mutation Operators

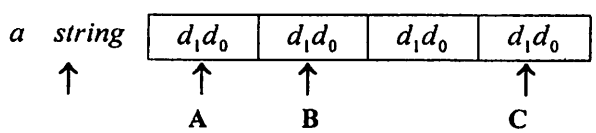
The reproduction, crossover, and mutation operators that were implemented to support the eight-digit decimal coding are discussed next.

Reproduction Operator

A basic roulette wheel reproduction operator was implemented for this data mining application. In this type of reproduction, each string in the population is given a roulette wheel slot sized in proportion to its fitness. Then, by " spinning" the wheel N times (the population size), N new offspring's are created for the next generation. By having a weighted, or "biased" roulette wheel, it is more probable that higher fit strings receive more copies in subsequent generations.

Crossover Operator

Crossover is used to improve the population fitness by introducing new strings into the population. Thus, the crossover operator for this problem had to introduce new item combination strings into the population. Due to the fact that the sub-strings (item numbers) within a coded string must be preserved, possible cross sites exist at the item boundaries within a string, as shown by points A, B, and C below :



In determining a crossover operator for this application, it was important to consider the fact that duplicate item numbers cannot exist

within a string since this would represent an invalid combination of items. For example, if the following two strings were crossed using simple crossover and a cross site as noted by A,

*string*1	04	21	06	15
*string*2	03	93	24	04

A

Strings 1· and 2· would result, as shown :

*string*1·	04	21	24	04
*string*2·	03	93	06	15

Note that the resulting string 1· only contains three different item numbers (04, 21, and 24) instead of four.

Given this consideration, it was obvious that a simple single point crossover operator would not be adequate. Crossover operators such as partially matched crossover, or PMX[7], and ordered crossover [7] ensure that duplicates do not occur in children strings. However, these crossover operators require that each parent string contain the same characters. The situation in the data mining problem, however, is slightly different in that every item in the first string may, or may not, be in the second string. Considering this fact, two different crossover operators were developed for examination : aligned single-point crossover (ASPX) and unmatched crossover with single child offspring (UXSCO), each of which is described below. The results of simulations run with each of these crossover operators are discussed in the section, " Crossover Operator Simulation Results." Those results show that the ASPX crossover operator provided very good performance, better than the UXSCO operator.

Aligned Single-Point Crossover (ASPX)

Recall that the main goal of our crossover operator is that it produces children strings which do not contain duplicate item numbers. This goal can be achieved with aligned single point crossover as follows : Two parent strings, such as those shown below, are chosen at random from the current population.