Combining Fuzzy Sets and Genetic Algorithms for improving strategies in Game Playing

A. CINCOTTI, V. CUTELLO, G. SORACE
Department of Mathematics and Computer Science
University of Catania
V.le A. Doria 6, 95125 Catania
ITALY

Abstract: - In this paper, we present a genetic algorithm for tuning weights in a decisional fuzzified process. The genetic algorithms will work using a multi-point dynamic crossover function. As a case study, we will test the behavior of the proposed GA on a simple yet interesting game, where a winning strategy can only be based on a multi-criteria computational process.

Key-Words: - Genetic Algorithms, Fuzzy Logic, Game playing

1 Introduction
Genetic Algorithms can be very naturally used in decision-making frameworks where alternatives are evaluated by means of some linear static evaluation functions. Such functions characterize the intelligence of the heuristic guiding the direction of the search in the space of the possible outcomes. This is the case for instance for some game programs, where the static evaluation function has the general form

\[ h = \sum_{i=1}^{n} w_i f_i \]

and the values \( f_i \) represent some specific features of the game, whereas the values \( w_i \) represent their relative importance in the game. Using the heuristic \( h \) one can apply classical search algorithms such as "minimax" and alfa-beta pruning, to improve the overall computational performance (see [8] for a complete explanation of these concepts).

An interesting framework for combining the powerful evolutionary search mechanism with fuzzy logic in games has been proposed in [6]. In the next sections, we will describe such a framework, we will define dynamic multi-point crossover and finally we will see the behavior of a generated program on an interesting computer game (see [5,7] for a deep explanations of concepts related to genetic algorithms).

2 Playing games with a coach
Let us now describe the framework for using a genetic algorithm to choose the best weights when playing fuzzified games. Such a framework has been introduced in [6].

2.1 How is the game fuzzified
Most games (and real life application which can be formalized as "games") can be fuzzily divided into parts: beginning, middle-stage, ending, etc. Chess players are aware of the different opening strategies (a well defined and well studied small sequence of moves). Good chess players already know whether their opponent is a "good" player by their opening sequence. A good opening can give you a big advantage or it can already put you in a very week position. The middle-stage of the game is, to many extents the most decisive one. It is at this stage that strengths and strategies of the players are usually of the utmost importance and become the decisive factor. In contrast, openings and ending are usually standard and well defined. In other words, openings can only make you loose if you do not know a good sequence, and endings cannot change your fate if your opponent knows what to do. Such a simple fuzzy division of the game history turns out to be quite good in most cases. What is needed is a good definition of the membership functions to the fuzzy sets "beginning", "middle" and "ending".

2.2 Strategies for the different parts of the game
To define the strategies for the game we need to define some specific features, which can be assigned a numeric value. If we have \( k \) features their corresponding numeric values will be denoted by \( f_1, ..., f_k \). These features will be combined by weights \( w_1, ..., w_k \), giving a global value \( h = \sum_{i=1}^{k} w_i f_i \) which represents the "desirability degree" of that specific game stage. Once we set the weights, we clearly have a strategy for that part of the game:

- at any instant, we will choose the move which maximizes for us the value of \( h \),
- and at the same time it minimizes it for the opponents.

How many and what features to choose is obviously strongly dependent on the specific game and it must be fixed a priori.
Finally, given the membership functions, $i_1$, $i_2$, $i_3$, which associate to a game phase a membership value to the three parts of the game above described, the total static evaluation of the desirability degree of a game phase $t$ can be given by the weighted average

$$h(t) = \sum_{k=1}^{3} \left[ \mu_k(t) \sum_{j=1}^{n} w_{i,j} f_j \right]$$

### 2.3 What is chromosome

Using the above described division, a chromosome is an ordered list of 3 elements. The first element represents the weights for the beginning part of the game, the second element the weights for the middle part of the game and, finally, the third elements represents the weights for the ending part of the game. This is called a triploid coding scheme. In particular, if we have $k$ weights for each of the three parts of the triploid, these weights will be transformed in binary and put in a sequence one after the other. How many bits to use for these representation, i.e. what level of precision we want depends obviously upon the fitness function we will be using.

### 2.4 Fitness

How do we compute the fitness value of a chromosome? We recall that a chromosome is basically a computer program that is playing the game with some specific strategies built-in by means of the weights. Its fitness value can only be computed by letting it play and see how it performs. To do this, we use the idea of a coach (or even multiple coaches).

A coach is a program as well with some specific, maybe common-sense or "greedy" strategies built-in. The two programs (coach and chromosome) will play against each other for a certain number of games, we call this the length of training phase. The number of games won by the chromosome will be its fitness value.

If we are dealing with games that allow a final score, then an alternative fitness definition could be the sum of all the scores obtained by the chromosome minus the scores obtained by the coach.

Note that the length of the training phase is fixed for all the chromosomes and is, obviously, another parameter which must be fixed a priori and whose choice can influence heavily the behavior of the GA.

### 3 Dynamic crossover

Let us recall the Schema Theorem, a fundamental theorem in GA. Such a theorem, tells us that short and high-fitness schemata, i.e. specific sequences of binary bits and "don't care" symbols, characterizing entire subclasses of genes, can produce exponentially many new offspring in the future generations. Multipoint crossover with a high value of $n$ can produce new (hopefully good) genetic combinations, but it can also break (with probability higher than single point crossover) high fitness schemata which would be better to keep as genetic patrimony for the future generations. We could then think of using a crossover function which chooses the number of points according to the fitness function of the two chromosomes involved. The rule of thumb is the following: the higher their fitness values, the smaller the number of points chosen for the crossover. To be able to apply this rule we need to:

- set the maximum number $n$ of crossover points allowed. The choice is obviously dependent on the size of the chromosomes and on the problem we are dealing with;
- map the fitness values to the set of values \{1, 2, ..., $n$\};
- impose a "class reproduction system": only pairs of chromosomes with very similar fitness values can be chosen for reproduction.

The choice of the best possible crossover function is one of the most studied and, to some extent, controversial topic in the GA field. We obviously do not claim that our dynamic choice is the best. We will however show with a specific example in the next section that the overall performance of the GA in producing a "good game player" with dynamic crossover is better than the one with static crossover.

Since the value of $n$ is application dependent, let us try to analyze the other two problems we have to solve. The two problems are highly interrelated and a solution for one will automatically impose a solution for the other.

There are basically two (simple) ways to solve the mapping problem:

- sort the $m$ chromosome in a population in decreasing order with respect to their fitness value. Put the first $m/n$ in one class, the second $m/n$ in a second class and so on. At the end we will obtain $n$ classes of equal size. Only elements in one class will be allowed to mate. The elements in the first class will use single point crossover, and, in general, the elements in the $i$-th class will use $i$-point crossover. Such a method of dividing up the population into a certain number of subclasses of equal sizes can be made more general. In principle, we could think of using $n'$ different crossover methods, the first with $n_1'$ points, the second with $n_2'$ points and so on. Obviously all these numbers will be greater than equal to 1 and smaller than or equal to $n$. Then we divide the population size into $n'$ sub-population of equal size and act as before.

- Another way is to divide the elements of the populations into clusters whose elements have all roughly equal fitness values. These clusters will not have necessarily the same number of elements and, it might even be the case that at every generation we may have a different number of clusters. Obviously, such an approach implies a clustering sub-routine which "quickly" acts on the population right after the new generation is formed.
4 Chromatic field

We tested the combined concept of fuzzy set theory and GA’s in the domain of Chromatic field which is a game of reasonable complexity for the task of learning because of its moderate branching factor. Let us first briefly describe the game. The game is played by two players, P1 and P2, on an 45 by 45 board (it could obviously be played on boards of any side or shapes). Initially every square on the board is painted with a random color taken from a set of 7 colors. Again, it could be any number of colors, but we think that less than 5 will make the game not interesting.

P1 starts from the upper-left corner of the board and P2 from the lower-right corner. P1 starts the game by choosing one of the square adjacent to the starting one. By adjacent we mean any square which intersects either on one side or just simply a corner point. After this move, P1 captures all the squares of the same color bordering the chosen one (in general it will capture all the square of the same color bordering any of its squares). All the squares owned by P1 are set with this new color. There is only one restriction: the color chosen by one player must be different from the current opponent’s color. When a player does not have any legal moves, he or she loses a turn. The players take turns setting colors until neither player can make another move. The player with the highest number of squares on the board is the winner.

We also produced a modified version of the game. In this other version, we give different values to different colors. Moreover, the number of squares of a specific color in the board, is inversely proportional to the value of the color. We observe that the first version of the game is a particular case of the second version in which all the squares have the same value. It is now more difficult to find a good strategy for the game because we must get the maximum score to win the game and the maximum expansion factor for capturing more squares as possible but, clearly, these two objectives are in contrast with each other.

4.1 Some implementation details

We will describe here some of the implementation decision we made for our case study.

We start by mentioning that the mutation operator is implemented by flipping a bit with very low probability: 0.01.

We also chose to always keep the top two players in the older generation when each new population is generated. This principle is usually called elitism.

Each generation in out GA implementation contains 60 players and the evolution process is repeated for 100 generations. Coach used in the GA training process (summarized in Figure 1) is a player that uses a simple greedy strategy.

The key features of the game are three:

- **Square value**: that is to say either the number of squares owned or the total value of the squared which are owned, if numeric values are assigned to squares of different colors. In particular, we will use a square value difference: the number of squares owned minus the number of square owned by the opponent. Therefore, this value could be negative.

- **Capturing square value**: that is to say the number of squares which could be captured minus the number of squares which could be captured by the opponent.

- **Expansion factor**: this the average of the distances of the squares from the starting point (i.e. the corner of the board). We take into account the difference between the expansion factor and the opponent's expansion factor.

The weights are coded using 8 bits. Therefore the maximum value is 255. Tables 1 and 2 show the weights for the coach and the best player (who wins the game with an average advantage of 64 squares) obtained after one hundred generations. It is interesting to notice that the greedy approach codified by the coach "always make the move that increases the number of squares" is changed by evolution. The more games are played the more apparent becomes that Expansion factor and Capturing Square Advantage start playing a very important role in the design of a winning strategy. Therefore, their weights change and they also do so according to the what stage of the game is being played.

To conclude our case-study discussion we also mention the

---

1 This game used to be present in an old version of Linux. It is not contained in the new versions and we do not recall its original name. We apologize to the author(s).
fact that we implemented both versions of the game: the one where all the colors have equal weights and are equally likely to be randomly drawn on the board and the one where the colors have different numeric values and the probability of appearing in the board is inversely proportional to their numeric values. While the genetically evolved strategy is basically unbeatable by a human in the first case (even after about 60 generations), in the second case instead it takes a lot more generations (about 100) to produce a player very competitive with a human player. The reason behind such results is very simple: in the first case we have a game board which looks to a human completely random. In the second case, a human player starts picking up patterns and can use them to design a good overall strategies.

<table>
<thead>
<tr>
<th>Squares advantage</th>
<th>Open</th>
<th>Mid</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>255</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>Expansion factor</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Capturing squares advantage</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1: weights for the coach

<table>
<thead>
<tr>
<th>Squares advantage</th>
<th>Open</th>
<th>Mid</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>255</td>
<td>233</td>
<td>190</td>
</tr>
<tr>
<td>Expansion factor</td>
<td>243</td>
<td>245</td>
<td>111</td>
</tr>
<tr>
<td>Capturing squares advantage</td>
<td>31</td>
<td>110</td>
<td>228</td>
</tr>
</tbody>
</table>

Table 2: weights for the best player

4 Conclusion

In this paper we applied the idea of dynamic crossover in the framework of fuzzified games. Chromosomes represent weighted criteria for choosing the best possible moves. The evaluation of the fitness value is a simple linear function of the weights and the membership values for the stages of the game. We plan to extend the simple model introduced here in many directions, let us mention a few

- We intend to introduce more complex aggregation functions for the fitness value (see for instance [2]).
- We also plan to attach to the fitness value a concept of rationality. That is to say, the weights obtained through evolution may give rise to an "irrational" decision mechanism. Rationality will be defined using the concept of fuzzy rationality measure already present in literature (see [3,4]).
- Finally, we would like to study games where more than three simple stages can be defined and for which then, there is a need of using several well tuned membership functions. Tuning of membership functions can take place by means of statistical sample drawn from underlying (possibly unknown) probability distributions and using the learning framework proposed in [1].

References: