In Search of No-loss Strategies for the Game of Tic-Tac-Toe using a Customized Genetic Algorithm

Anurag Bhatt, Pratul Varshney, and Kalyanmoy Deb
Kanpur Genetic Algorithms Laboratory (KanGAL)
Indian Institute of Technology Kanpur
Kanpur, PIN 208016, India
{anuragb,pratul,deb}@iitk.ac.in

ABSTRACT

The game of Tic-tac-toe is one of the most commonly known games. This game does not allow one to win all the time and a significant proportion of games played results in a draw. Thus, the best a player can hope is to not lose the game. This study is aimed at evolving a number of no-loss strategies using genetic algorithms and comparing them with existing methodologies. To efficiently evolve no-loss strategies, we have developed innovative ways of representing and evaluating a solution, initializing the GA population, developing GA operators including an elite preserving scheme. Interestingly, our GA implementation is able to find more than 72 thousands no-loss strategies for playing the game. Moreover, an analysis of these solutions has given us insights about how to play the game to not lose it. Based on this experience, we have developed specialized efficient strategies having a high win-to-draw ratio. The study and its results are interesting and can be encouraging for the techniques to be applied to other board games for finding efficient strategies.

Categories and Subject Descriptors

I.2.8 [**Computing Methodologies**]: Problem Solving, Control Methods, and Search

General Terms

Algorithms

Keywords

Tic-tac-toe, evolutionary games, learning strategies.

1. INTRODUCTION

The game of Tic-tac-toe is a commonly-played game. Many who play the game develop some strategies on their own which usually do not let the player lose the game. However, in a significant proportion of the games played, the game ends with a draw. In the past, researchers have studied computing methods to generate efficient strategies for not having to lose the game even once. Hochmuth [5] demonstrated how a genetic algorithm (GA) can be used to evolve a perfect Tic-tac-toe strategy, which never loses a game it plays. He concluded that there are 827 unique game-states that are encountered during game play and concentrated on finding a *single* no-loss strategy. The study reported a single no-loss strategy, but did not provide the description of that strategy to know its properties. Soedarmadji [7] suggested a decentralized decision-making procedure to find a competent strategy which forces a draw or a win depending on the proficiency of the opponent player. Although such a goal should result in a no-loss game-playing strategy, we observed that the resulting strategy reported in the study loses in at least three different scenarios (which we have highlighted in Section 4.5).

In this paper, our goal is to follow an identical analysis procedure as reported in the literature and revisit the use of GAs in finding not one but as many no-loss strategies as possible, so that we can make an attempt to unfold what causes such a strategy to not lose. For this purpose, we use a representation scheme similar to that in [5] and design new ways of evaluating a solution through matrix processing in MATLAB, a new initialization scheme, customized GA operators with a controlled elite preservation scheme and a two-tier GA procedure. Interestingly, our study is able to find as many as 72,657 no-loss strategies for playing the game of Tic-tac-toe, of which only a fraction (827) were reported earlier. Furthermore, we analyze these no-loss solutions to arrive at a number of efficient strategies which produce excellent win-to-draw ratio, a matter which has not also been paid much attention in the past. Although the game of Tic-tac-toe does not implicate much in practice, it is still a popular game and importantly the search of optimized strategies involve phenomenally large search space. The use of GAs as an optimization tool, its need for customization and execution of various inter-linked concepts of optimization used in this study are interesting and should motivate readers to perform similar studies for other games and problems.

2. REPRESENTATION OF A STRATEGY FOR TIC-TAC-TOE

Before discussing how we apply a genetic algorithm to find good strategies for playing the game of Tic-tac-toe, we first describe a representation scheme used to define a playing strategy. The procedure involves a number of definitions, which we make in the following:

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.

1. **Number of unique game-states:** Although it is possible to fill the nine squares on a Tic-tac-toe board in 3^9 or 19,683 ways, we realize that all of these are not feasible game-states. The conditions for feasibility of a game-state are as follows:

   (a) The number of `X` in any given state would be either one more than or equal to the number of `O` placed on the board, as `X` is assumed to play first.

   (b) Either player would not move if the other has won.

   Using these conditions, the infeasible game-states are removed from further consideration. Some such infeasible game-states are shown in Figure 1.

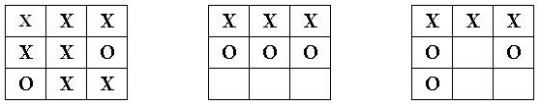

Figure 1: Some examples of infeasible game-states.

Another notable feature of the Tic-tac-toe board is the symmetry of the game-states. Any game-state, when rotated by 90, 180 or 270 degrees, or when reflected along the horizontal and vertical axes represents game-states which are equivalent to each other. We called these game-states as *equivalent game-states*. An example of eight equivalent game-states are shown in Figure 2. Out of eight equivalent game-states, we select one as the *base-case* game-state and suggest an empirical method of identifying it (we describe this method below).

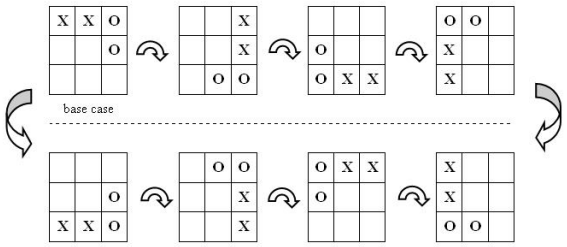

Figure 2: An example of eight equivalent game-states.

To reduce the number of overall useful game-states, we remove the repeated equivalents. By this process, we end up having a total of 765 unique game-states for the game of Tic-tac-toe. This number was also arrived at by using an enumerative method of creating all possible game-states on a computer and keeping only the feasible and non-repeating game-states. Other studies have also confirmed this number [6]. It is not to be confused with that the total number of solutions of the game of tic-tac-toe is only 765 and one can do an exhaustive search to choose the best solution. At every move, a player can choose at most 765 game-states, thereby making an astronomically large number of possible strategies of playing the game. The exact number of such feasible strategies is difficult to compute as a latter move depends on earlier moves.

2. **Selection of a base-case from the equivalent game-states:** In this paragraph, we describe a method for selecting one of the eight equivalent game-states for our further consideration. For this purpose, we first assign a weight to each of the nine positions, as shown in Figure 3. Thereafter, for all game-states in a particular equivalent class, we calculate four metrics and use a lexicographic evaluation scheme. The highest priority is given to a metric which is the sum of the weights for positions occupied by `X`. The game-state with the minimum value of this metric is chosen as the base-case game-state. In case, there is a tie on the minimum metric value among more than one game-states, we compute the next metric. The next priority is given to the sum of weights for positions occupied by `O`. The next is to the product of weights for positions occupied by `X` and finally the last priority is given to the product of weights for positions occupied by `O`. On the basis of this hierarchy, the game-state with minimum metric value is chosen as the base-case game-state. For example, in the example shown in Figure 2, the first game state (marked as 'base-case') is found to be the base-case among all eight game-states.

1	2	3
4	5	6
7	8	9

Figure 3: Weights for different positions on a board.

3. **Development of a game-base:** Next, we develop a game-base which stores information about all independent 765 game-states. For this purpose, we develop a 765×10 matrix to store these game-states. We start our game-base with an empty board of Tic-tac-toe and start filling it alternately with `X` and `O` (henceforth, we represent them by `1` and `2`, respectively) at possible places. Once a game-state is created, its base-case is evaluated. If it is not found in the game-base created so far, this base-case is added to it. The first nine elements of each row in the game-base matrix store the game-state in terms of `0`, `1` and `2` (`1` and `2` representing the respective player's moves and `0` representing blanks). The 10th element of each row stores the number of filled positions (either `1` or `2`) in the board. We call this value as the *game-level*. A few starting rows representing a few game-states are illustrated in Figure 4. For example, the eighth row in the matrix has two entries at positions 1 and 6, which are reflected at the first and the sixth column. The 10th column shows the total number of entries for this game-state (two in this case).

4. **Representing a strategy as a solution:** We are now ready to present a representation scheme of a strategy which we would use as a solution in our proposed genetic algorithm procedure. For this purpose,

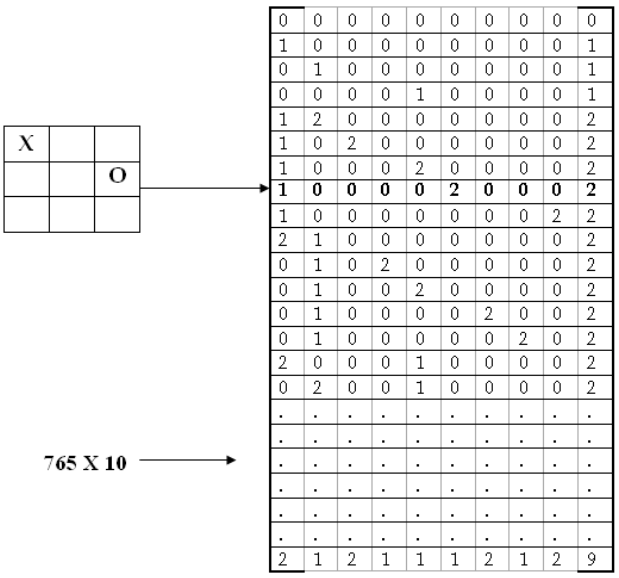

Figure 4: Starting rows of the game-base matrix.

we define a 765-length vector with the following meaning to its entries:

- The position of an entry in the vector signifies the row number of the game-base. We give an example a little later.
- The value of entry is the row number of the game-base to which the game should move if it encounters this position during the play.

Let us consider the strategy shown in Figure 5. The

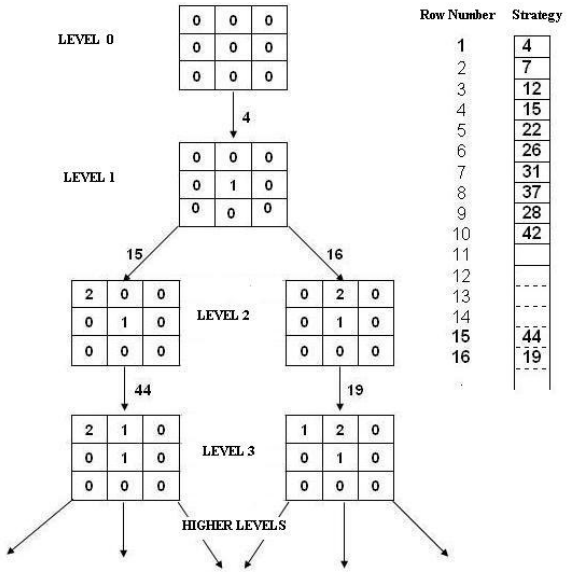

Figure 5: An illustration of a play of a strategy.

first level is always level zero, in which there is an empty board. This case is represented by row 1 of the game-base. The overall outcome of the game will depend on whether the computer plays as the 'first' or 'second' player. Say, the computer is the first player. The strategy shown in the figure has an entry 4 on the first position. This means that the computer should play a move dictated by the fourth row of the game-base. This move is to put a 1 at position 5, as shown in the 'Level 1' move of the figure. Now, it is the turn of the opponent. For this case, there are only two unique base-case moves (each having four equivalent game-states). These two base-case game-states are represented by rows 15 and 16 in the game-base. If the opponent chooses the game-state dictated by row 15, the next move of the computer can be found from the strategy vector at position 15. The figure shows an entry of 44 at this position. This means that the computer has chosen a game-state dictated by the 44th row in the game-base. This game-state belongs to 'Level 3' and is shown in the figure. On the other hand, if the opponent chooses game-state 16, the strategy dictates that the computer should use the game-state 19. This game-state taken from the game-base is shown as one of the moves under 'Level 3' in the figure. The game continues in this fashion until a result (win, lose or draw) is encountered and no further moves are made.

3. A CUSTOMIZED GENETIC ALGORITHM

The first task in GA is to create a set of random initial population of solutions or strategies. Next, GA operators (selection, recombination and mutation) are to be designed for handling the game playing problem. It is also important to realize that the GA developed here is far from being a naive one, as to solve a problem having an astronomically large search space, problem information must have to be used in initialization and GA operator design. Since an efficient and problem specific representation mechanism is used here, it allows us to choose a simplistic recombination operator. But to maintain diversity in the GA population for creating better solutions, we also use an innovative niching mechanism (we called a controlled elite preserving operator). We also use a two-tier GA approach. We describe them in the following paragraphs.

3.1 Creation of Random Initial Population

Each entry in a strategy cannot take any arbitrary integer between 1 and 765. The entries should be such that a hierarchy of increasing level is maintained. Recall that the first position of a strategy vector corresponds to row 1 in the game-base. We substitute a 1 (as this would be the first player's move) in place of a 0 on the entries of the first row of the game-base and collect all resulting game-states. Then, we find all base-cases corresponding to these game-states and observe that there are three distinct base-cases (marked by rows 2, 3 and 4 in the game-base). We simply choose one at random as the first entry of the strategy. Next, we play the above procedure for the second row of the game-base. After substituting 2 (as this belongs to level 1 and would be the second player's move) in place of 0 one by one in the entries of second row in the game-base and finding the base-cases covering all such game-states, we observe that only rows 5 to 9 can be achieved. Thus, we choose a number randomly between 5 to 9 to fill the second position of the strategy. For the third entry in the strategy, we replace all

0 entries by 2 in row 3 of the game-base and determine the base-cases. They turn out to be the rows 10 to 14 in the game-base. The strategy shown in Figure 5 has chosen 12 as an entry. Similarly, for the fourth entry, a row out of 15 and 16 can only be chosen. Now all moves of the strategy from level 1 to level 2 have been defined. Next, we define moves from level 2 to level 3. We shall start from changing 0 entries of row 5 to 1 (as it is now the turn of player 1) and find base-cases where a move is possible. Thereafter, one of these base-cases can be chosen at random.

Since this procedure has to continue for all 765 entries of the strategy, after a while moves may not be possible to be made, simply because that the outcome of the game is already determined by the previous moves. In such a case, we simply assign a dummy value higher than 765 as an indication to abort the game.

3.2 Fitness Evaluation

Our objective is to evolve several Tic-tac-toe strategies which never lose (meaning a draw or a win by the computer). This makes the problem to have a single objective of minimizing the number of losses. The evaluation of fitness of any strategy is done by first allowing it to play all possible games it could play, both as a first player and as a second player. For example, note from Figure 5 that there are two possible ways a game can move for the first player from level 1 to level 2, depending on whether the opponent made the left or the right side move indicated in the figure. Our evaluation procedure considers all such intermediate possibilities an opponent can have and count the total number of possible games resulting in wins, draws and losses. This is continued for the above strategy to be played as the second player. The total number of games lost in both cases as a first player and a second player is calculated. The fitness function is then defined as the fraction of games lost out of the total number of games played (both as a first and as a second player):

$$\text{Fitness} = \frac{\text{Number of games lost}}{\text{Number of games played}} \qquad (1)$$

It is clear that a strategy with fitness value equal to zero would be a perfect no-loss strategy.

3.3 Selection Operator

First, we employ the usual binary tournament selection operator, but it turns out to be already too greedy to proceed towards no-loss strategies. Due to the presence of a huge search space and existence of a few useful solutions in the search space, the population was found to lose its diversity very quickly and converge to a futile sub-optimal solution within a few generations. Instead of using a probabilistic tournament selection operator [4], which demands a probability parameter, next we decide to use the stochastic uniform selection (SUS) operator [3] along with a parameterless niching concept to maintain diversity of solutions in a GA population and to have a low selection pressure of current best solutions. Since, SUS inherently maximizes, we convert the above fitness function to the following form:

$$\text{Fitness} = \frac{1}{m}\left(1 - \frac{\text{Number of games lost}}{\text{Number of games played}}\right), \qquad (2)$$

where m is the number of population members having the same proportion of losses to game played. The effect of dividing by m provides a niching effect and help maintain multiple niches around good solutions in the population.

3.4 Crossover Operator

This study provides an excellent example in which the effort of designing a useful recombination operator can be reduced by choosing an innovative representation scheme. As every strategy encoded with the procedure described in Section 2 has meaningful entries in respective positions, a standard multi-point crossover operator can be applied to the selected individuals to create meaningful child solutions. A schematic diagram of the crossover operator is shown in Figure 6. The number of cross-sites is fixed at 50 (out of

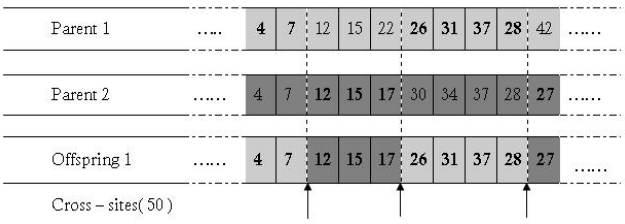

Figure 6: A schematic diagram of the crossover operator.

764 possible locations) to ensure the generation of diverse solutions. The crossover probability p_c (fraction of population members undergoing crossover) is chosen to be one, thereby making all population members to participate in the crossover operator.

3.5 Mutation Operator

Once again, due to the use of an innovative representation scheme, a standard mutation operator can be implemented with a caution. Every entry to be mutated can only be changed to a valid row number indicating the game-state in which it is allowed to move. We have discussed this matter while describing the creation of the initial population in Section 3. We link the mutation probability at a generation with the minimum fitness (as in equation 1) of the population directly. This way, a population having worse solutions has a larger mutation probability and vice versa. Specifically, we use p_m to be exactly the minimum of the fitness values (equation 1) of all population members. The number of entries (n_m) mutated in a strategy is defined as follows:

$$n_m = \lceil 250 p_m + 10 \rceil. \qquad (3)$$

To implement, n_m entries out of 765 positions are chosen at random and mutated to an allowable value at each position.

3.6 Controlled Elite Preservation

The normal method of elite preservation selects the best N individuals out of the parents and offspring and sends them to the next generation. This method was found ineffective in maintaining diversity in the population whether we used SUS or tournament selection. Therefore, we developed a new way of selecting individuals from the combined population: We combine three intermediate populations of size N each: (i) population before selection, (ii) population after crossover operation, and (iii) population after mutation operation. The combined population of size $3N$ is sorted

in worse values of fitness and select N solutions which are distributed according to arithmetic progression in the difference between two consecutive solutions. The solutions with sorted index of j defined below are chosen:

$$j(i) = i + 2N\frac{(i-1)(i-2)}{(N-1)(N-2)}, \quad i = 1, 2 \ldots, N. \qquad (4)$$

This procedures always includes the first and $3N$-th entry and $(N-2)$ other entries with increasing differences between two consecutive solutions. A little calculation will show that the above will ensure around 50% solutions from the best N of $3N$ combined members. The restoration of remaining 50% of solutions from the other two-third of the combined population helps maintain diversity in the population.

4. SIMULATION RESULTS

The code for implementing the above-mentioned genetic algorithm is written in MATLAB (version 7.0.1) and run on machines with AMD 64-bit processor and 512 Mb RAM. We used MATLAB because we defined all our data structures (including the game-base) as matrices.

4.1 Initial Studies

First, we tried using the binary tournament selection with a single-best preserving strategy, which led to quick loss in diversity and resulted in a convergence to a sub-optimal solution. Thereafter, we used SUS with a single-best elite preservation scheme. This strategy maintained diversity for a substantially longer amount of time, but the elite preservation was found to force it to converge to a sub-optimal solution as well. Both the above trials did not succeed in finding a single no-loss strategy. Figure 7 shows the variation of the population-best fitness with generation counter with the SUS operator. The GA parameters used in this study are as follows: population size = 100, maximum number of generations = 200. The figure shows that a no-loss solution is not found till 200 generations by this procedure (zero fitness value is not found!) and after about 40 generations, the population-best fitness did not change. We ran

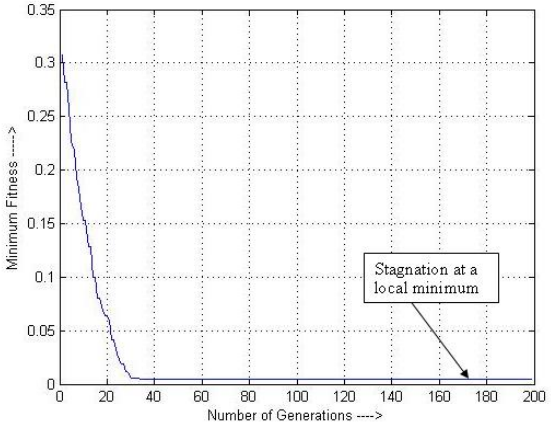

Figure 7: Population-best fitness with SUS and single-elite preservation strategy.

the above GA till 500 generations and no improvement was observed.

4.2 SUS with Controlled Elite Preservation

Next, we used SUS with the controlled elite preservation technique described above. With a population size of 100, the GA is run for 500 generations. Figure 8 shows a simulation run in which a no-loss strategy was finally discovered at 322nd generation. Thus, the controlled elite preservation seems to be an improvement, the number of generations required to find a no-loss strategy is large. We now suggest a two-tier strategy which used further problem information and is more efficient than this procedure. In handling large

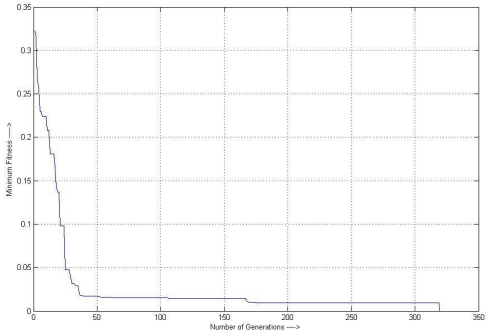

Figure 8: Population-best fitness with SUS and controlled elite preservation strategy.

and difficult optimization problems, such inclusion of problem information through initialization and operator design becomes an important task [2].

4.3 Two-Tier GAs: Independent Simulations as First and Second Player

It is somewhat intuitive that the outcome of the game of Tic-tac-toe depends on whether a player plays as a first or as a second player. There is a greater chance of winning the game by the first player (with a ratio of about 1.68:1). In the course of our study, we also notice that strategies as first and second player are mutually exclusive to each other. The moves made as a first player will never be made when it plays as a second player and vice versa. Therefore, we decide to use this knowledge in developing a strategy in our next study. The GA is run separately for evolving solutions for the first and the second player. This decreases the computational time considerably. Other than that, no-loss strategies start appearing in lesser number of generations in both cases.

To find more than one no-loss strategy, if exists, by our GA, we use the following procedure by maintaining an archive of no-loss strategies. As soon as a no-loss strategy is evolved, we move it from the population and save it in the archive. We use 100 population members and run our GA for 500 generations for each case (as a first player and as a second player). We run both the cases 11 times from different initial populations and tabulate in Table 1 the number of generations (best, median and worst) needed to find a single no-loss strategy. It is interesting to note that for developing a strategy for the first player, as few as four generations are enough. However, for developing a no-loss strategy for the second player more generations (minimum of 46) are needed. As mentioned above, the game of Tic-tac-toe is biased for

the first player and our simulation results agree with this fact.

Table 1: Number of generations required for first no-loss strategy to evolve in 11 independent runs.

	Number of generations		
	Best	Median	Worst
First player	4	7	10
Second player	46	278	432

Figures 9 and 10 show typical variations of population-best fitness with generation counter for both the cases. In

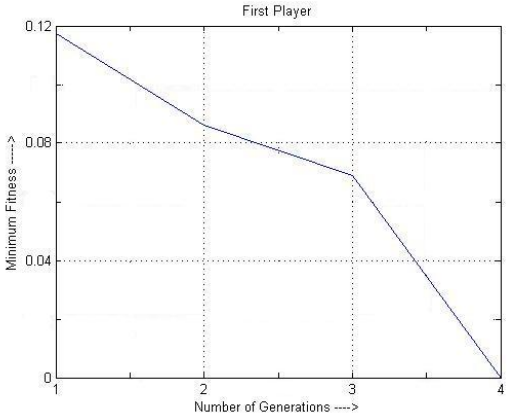

Figure 9: Population-best fitness for the game in which strategies are evolved for the first player.

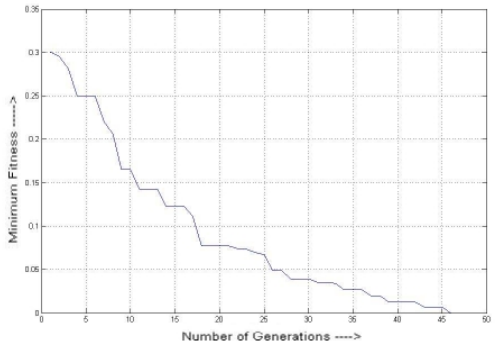

Figure 10: Population-best fitness for the game in which strategies are evolved for the second player.

all 11 runs, we found a no-loss strategy for the first player within a maximum of 10 generations. However, in 8 out of 11 times our GA finds a no-loss strategy for the second player in a maximum of 432 generations and in the remaining 3 runs, our GA was not able to find a no-loss strategy in 500 generations.

At the end of both runs, we have two groups of mutually exclusive solutions: one, which would never lose as a first player and the other as the never-losing second player. The solutions from these groups are then blended together taking the moves of respective players from their individual solutions. There are a few repeated solutions which we remove and arrive at 117 solutions for the first player and 621 for the second player. Therefore, we have a total of $117 \times 621 = 72,657$ different no-loss strategies[1] as first and second player when played against an opponent. To choose one particular solution from this set is a decision-making task which requires a further analysis.

4.4 Analyzing the obtained solution set

In the above optimization task, we have considered a fitness function which only considers the fraction of losses among total possible games. All 72,657 combined strategies to be played as a first and a second player will never lose in all possible games played against any opponent. But we may be interested in knowing the proportion of wins and draws for all these no-loss strategies. For this purpose, we record the number of wins and draws which these strategies make against all opponents and plot them in Figure 11. We observe that the number of wins vary from 89 to 156 and number of draws varies from 38 to 110. Clearly we are interested in strategies with more wins and less draws. The number of points appearing on the plot are conspicuously less than 72,657, which implies that there are multiple strategies to achieve an identical number of wins and draws. In fact, we find that there are only 2,263 independent combinations of wins and draws available in our data set. To make the situation more clear, we plot a contour graph of the above plot with the frequency of number of solutions at a point. From now on, we will refer to strategies with same number of total wins and draws as *similar* strategies. The granular gray region is the background where no solution was found. The strategies with at least 25 similar strategies are shown in white. As we move inwards, the darkness of the color increases with increasing frequency of similar strategies. The small black regions in the most inner parts of the plot are strategies with more than 150 similar solutions in our data set. The figure gives us an idea about the effectiveness of our fitness function and the nature of search of our GA. Obviously, strategies with lesser number of draws and higher number of wins would be considered as the final desired strategies with no-loss being a constraint for all of them. Therefore, the strategies in the upper left region of our plot are more important than the others. A non-dominated set of our data has been shown by a connected line in Figure 11. As shown in the Figure 12, these strategies are very few in number. On the contrary, the mode of the data lies somewhere in the middle of the plot. The darker regions in the center are those where a GA search with only minimizing the proportion of loss will be directed. Thus, we realize that in addition to using the no-loss constraint, we need to perform a two-objective optimization [1] of proportion of wins and draws. We defer this study for another time and discuss some properties of the obtained high-performing solutions.

The first glance at these solutions reveals that many of

[1]Even though all 72,657 strategies are different from each other, during game-play, many of them lead to the same game-states and all may not turn out to be distinct in actual game simulations. This happens because the state in which they are different is never reached during actual game-play. However, strategy-wise they are all different from each other.

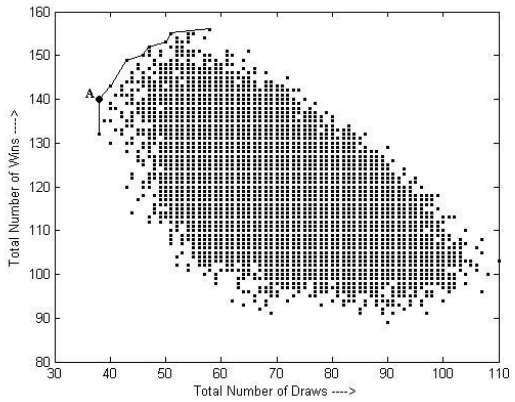

Figure 11: Solution set in terms of wins and draws.

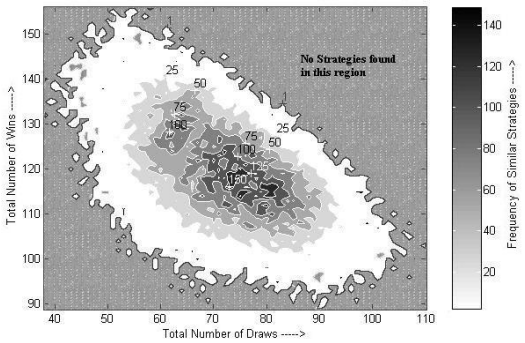

Figure 12: Contour plot of win-draw set by the frequency of solutions.

them differ from others by not more than 20 moves. Hence, we select a subset of these solutions for further analysis. For the first player, we find solutions which differ from each other in at least 80 positions in GA strings. There are 13 such solutions out of 117 total strategies found by minimizing the proportion of loss for the first player. Similarly, for the second player we find 27 solutions out of 621 which differ from each other in at least 90 positions. The combination of these solutions leads us to have 351 widely varying no-loss strategies. These 351 strategies are compared with each other on the basis of their moves and their state-trees. Some commonly-observed moves are listed below:

1. If the opponent is one short of a win, block it.
2. Occupy the center if it is empty.
3. If center is filled, occupy the corner and edge-center in that order.

Those who are used to playing the game of Tic-tac-toe will recognize these rules as good moves for not losing a game. It is interesting that we rediscover these basic moves through an analysis of computer-generated set of no-loss strategies.

Another observation is that the solutions strive to counter the opponent's moves rather than ensuring their own win. This is because we minimized the number of losses and did not explicitly maximize the number of wins.

4.5 Creating Efficient Solutions

We choose the set of strategies with the maximum number of similar solutions occurring among 72,657 solutions found by our GAs. This set (we call the mode-set) of similar solutions is likely to be picked by a GA as a no-loss strategy, but may not result in most possible wins. Table 2 shows that one of these mode-set strategies result in 41 wins and 5 draws as first player and 75 wins and 68 draws as second player out of 189 total possible games, providing a win-to-draw ratio of 1.59 for this strategy.

Table 2: Good Strategies for playing the game of Tic-tac-toe.

Strategy	As first player		As second player		Win/
	Wins	Draws	Wins	Draws	Draw
Mode	41	5	75	68	1.59
Mode-move	44	8	88	61	1.91
Best W-t-D	42	3	98	35	3.68
Mod. heuristic	18	1	68	21	3.91

Next, we create a strategy move-wise by finding the maximally occurring moves from going from one level to the other among all 72,657 solutions found by our GA. Table 2 shows that this resulting 'mode-move' strategy is better than the mode-set strategies discussed above and has a better win-to-draw ratio.

To select an efficient strategy directly from the set of 72,657 solutions found by our GAs, we choose the strategies having the highest win-to-draw ratio. We find that there are two such strategies (having identical win-to-draw ratio) which are marked in Figure 11 by 'A'. The table shows that the win-to-draw ratio for this solution (marked as 'Best W-t-D') is 3.684, which is much better than the above two solutions.

Soedarmadji [7] reported a heuristic solution which we have implemented in our code and found that it actually loses in at least three scenarios. These game-states in which the heuristic solution loses the game are shown in Figure 13. In the figure, `X` is played by the heuristic solution. In the first case, `X` should have been played in any of the edge-centers.

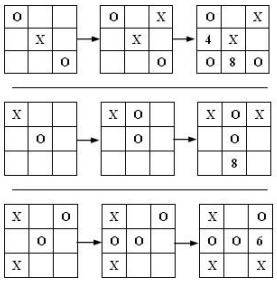

Figure 13: Three cases in which Soedarmadji's heuristic solution loses the game. `X` is played by the heuristic solution.

Since the heuristic solution played the top-right corner, O has played it towards completing a win. The second case is more obvious, as X should have blocked the bottom edge-center to stop the opponent from winning. Similarly, the win of the opponent could have been avoided by playing at position 6. Based on the insights obtained by analyzing 72,657 solutions found by our GAs, we modify this heuristic solution manually to fix these problems and create a modified heuristic solution. The modified strategy as a first player is shown in Figure 14. Out of 19 cases, this strategy wins in 18 of them, as shown in the figure. This solution has a win-to-draw ratio of 3.91, which is better than all other strategies found above.

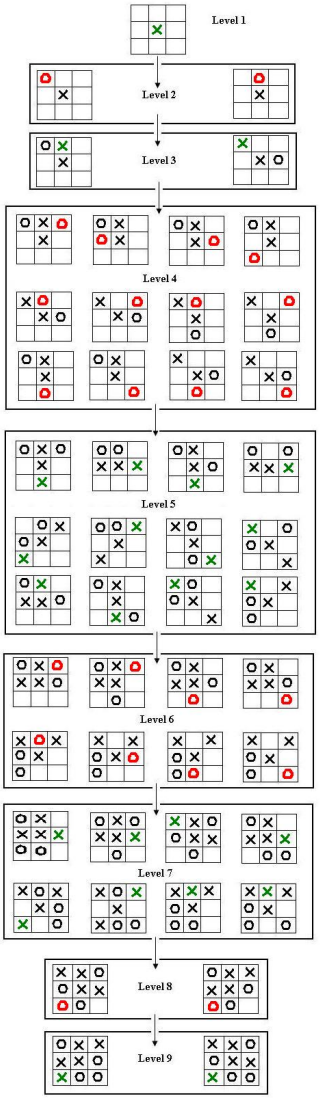

Figure 14: The strategy as a first player of the modified heuristic solution.

5. CONCLUSIONS

In this paper, we have applied a genetic algorithm for evolving no-loss strategies for the game of Tic-tac-toe. First, we have described a scheme for representing a strategy in a GA and discussed the initialization scheme and GA operators for the minimization of proportion of losses in *all* possible game scenarios. By carefully designing the GA procedure, we have been able to find not only one, but as many as 72,657 different strategies which will never lose a game. This is a remarkable result for the game of Tic-tac-toe. Further, we have analyzed these solutions and come up with four different specific no-loss strategies which also cause a large win-to-draw ratio. An analysis of these solutions has also resulted in a number of basic principles of playing the game of Tic-tac-toe for never losing the game. There are a number of other conclusions which we make from the study:

- Considering symmetry, all possible game-states in Tic-tac-toe are 765, as opposed to 827 mentioned in the literature [5]. This is also supported by another study [6].
- Since the moves made by the first and second player belong to disjoint sets, therefore their corresponding strategies are mutually exclusive.
- A GA can be successfully applied to evolve several no-loss strategies using above problem-specific information. Increase in population size above 100 has not shown any appreciable effect on the results. Tournament selection is found to be not effective as it loses diversity very fast. A similar observation has been made for a single-best elite preservation operator as well.
- The chance of obtaining a no-loss strategy for first player is high, which agrees with the fact that the first player has an advantage in this game.

The study can be extended by treating the optimization task as a bi-objective one of maximizing the number of wins and minimizing the number of losses subject to a no-loss constraint. Nevertheless, this study has revealed a number of interesting aspects of evolving no-loss strategies for playing the game of Tic-tac-toe, which may motivate other researchers to try similar studies for other games.

6. REFERENCES

[1] K. Deb. *Multi-objective optimization using evolutionary algorithms*. Chichester, UK: Wiley, 2001.

[2] K. Deb, A. R. Reddy, and G. Singh. Optimal scheduling of casting sequence using genetic algorithms. *Journal of Materials and Manufacturing Processes*, 18(3):409–432, 2003.

[3] D. E. Goldberg. *Genetic Algorithms for Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[4] D. E. Goldberg and K. Deb. A comparison of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms 1 (FOGA-1)*, pages 69–93, 1991.

[5] G. Hochmuth. *On the genetic evolution of a perfect Tic-tac-toe strategy*, pages 75–82. Stanford University Book Store http://www.genetic-programming.org/studentpapers2003.html, 2003.

[6] Mathrec contributors. Mathematical recreations, repository for mathematical diversions, http://www.mathrec.org/old/2002jan/solutions.htm, 2002.

[7] E. Soedarmadji. Decentralized decision making in the game of tic-tac-toe. In *Proceedings of the 2006 IEEE Symposium on Computational Intelligence and Games*, pages 34–38, 2005.