An Application of Genetic Algorithm to the Game of Checkers

Gabriella A. B. Barros Leonardo F. B. S. Carvalho Vitor R. M. Silva Roberta V. V. Lopes*

Universidade Federal de Alagoas, Instituto de Computação, Brazil

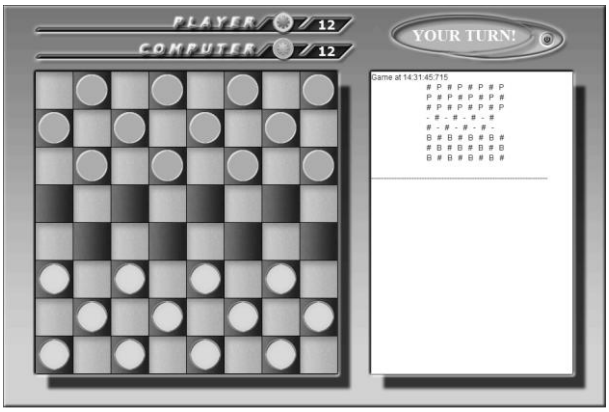

Figure 1: Evolutionary game of Checkers developed in this paper.

Abstract

The difficulty of a game's Artificial Intelligence is one of the determining factors of its success or failure. To define how the machine-opponent will behave is a hard and complex task. This paper aims to present an application of the Game of Checkers that tries to solve the matter of presenting an adversary which is convincing and of a similar level to the player, using Genetic Algorithm's techniques.

Keywords: Artificial Intelligence, Genetic Algorithm, Chess Games, Games

Authors' contact:
{gabyshini,lfilipebsc,vrafael1812}@gmail.com
*rv2l@hotmail.com

1. Introduction

In the world of electronic games, a smart and innovative interaction with the user is one of the biggest motives of a game's acceptance. Driven by the billions of dollars that the entertainment global industry moves every year, there was a great amount of attention towards researches that focus on improving the choices of the machine, increasing its learning ability. The idea of a computer assisting the learning of students was also a strong motivation, and this thought was proven correct in more than one occasion [Silva 2008].

The use of evolutionary algorithms to improve the performance of machine controlled characters, known as NPCs (Non-Player Characters), was suggested and researched before [Carvalho 2008]. This behavior, called AI, can be understood as the part of the code responsible for computationally controlling the movements of opponents and allies, making it appear as this last two are making intelligent decisions about the world around them, when in reality it is a predefined option for certain situation [Taitai 2003, Scwab 2004].

Although the use of evolutionary algorithms is applicable in many game styles, it was chosen to work with a game among those with strategy based on turns. Some games of such kind that were previously researched, successfully or not, are Tic Tac Toe, Chess, Checkers and Go. The AI techniques applied to them were: Neural Networks [Kecman 2001], Genetic Algorithms [Lopes 2003], Alpha-Beta Pruning [Russel and Novig 2004], and Finite State Machine [Wagner et al 2006].

This paper aims to propose the utilization of Genetic Algorithms (GA) to create an evolutionary game of Checkers. However, it is important to highlight that the goal is not to obtain an invincible machine-opponent, but one that allow the player to improve its abilities. It would, too, decrease the chances of the game becoming boring to the user, since the difficulty would always be changing to challenge the gamer.

This paper is organized into seven sections. Section 2 presents the set of rules used in the developed game of Checkers. Section 3 presents the application itself, and its functionalities; while section 4 discusses technical aspects of the application and the Genetic Algorithm used in the development of the game. Section 5 contains the results, which are discussed in section 6. Finally, section 7 presents a brief conclusion about the work.

2. Rules of the game of Checkers

The developed system uses the official rules of Brazil. These are described as follows:

1- The game is played on a chess board with 64 intercalated dark and light squares. The longest diagonal of dark squares of the board should be at the left side of each player. The goal is to immobilize or capture all the pieces of the opponent.

2- Two contestants play the game, each one with 12 pieces. One player uses white pieces, while the other uses Black ones. The first movement belongs to the white pieces. Figure 2 illustrates this situation.

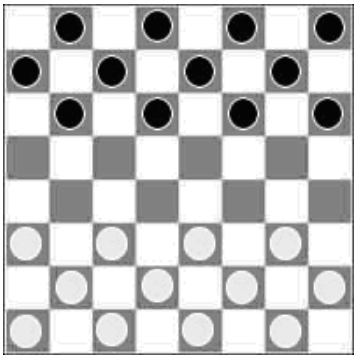

Figure 2: Inicial arrangement of the board.

3- The man, an uncrowned piece also called a Checker, moves itself diagonally, and always forward, one square at a time. When it reaches the farthest row of the board - the 8^{th} row – it is promoted to king.

4- The king has a bigger movement, being able to move backward and forward as many squares as it wants. However, it cannot jump over a piece of its same color.

5- The capture is mandatory. Two or more pieces side by side at the same diagonal cannot be captured. Figure 3 shows both scenarios, being that the scenario of an unpermitted capture is show by the king (marked with a D on top of it, from "Dama", the name used in Brazil).

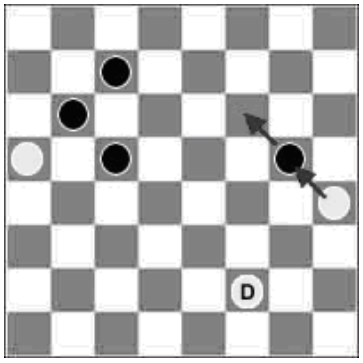

Figure 3: Capture of pieces at the game.

6- The man captures the king and the king captures the man. Man and king have the same value to capture or to be captured.

7 – The man and the king can capture both forwards and backwards one or more pieces.

8- The man that jumps over a crowning row during the capture of various opponents, but does not ends it movement at that row, shall not be promoted to king, as can be seen in Figure 4.

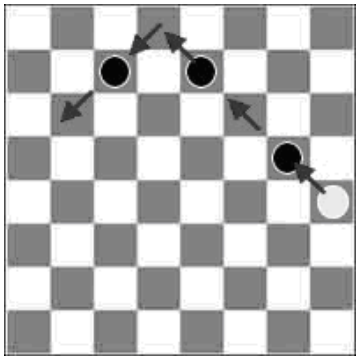

Figure 4: Capture jumping through a crowning row.

9- While performing the capture of multiple pieces, it is allowed to jump over the same empty square, but it is not possible to capture twice the same piece. It can be seen in Figure 5.

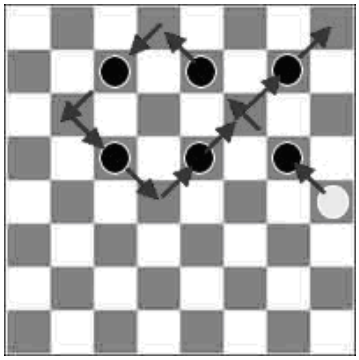

Figure 5: Capture of several pieces jumping over a same empty square.

10- The captured pieces shall not be removed from the board until the move has been finished, as shown in Figure 6.

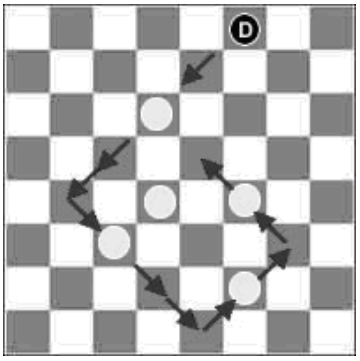

Figure 6: Capture and removal of several pieces.

11- After the kings have been played 20 successively times, without a man have been played or without a capture, the match is declared a draw.

12- End of matches of 2 kings against 2 kings; 2 kings against one king; 2 kings against one king and one man; one king against one king; or one king against one king and one man are declared a draw after five consecutive plays.

There is one last rule, called "rule of majority" in Brazil. If in a single playing movement it is possible to capture in more than one way, it's compulsory to execute the move that captures the most number of pieces. This rule was not implemented in order to allow the player and the genetic algorithm a greater freedom of choice of action to take in order to better assess whether the action it chooses is really the one that will bring more benefits.

3. Application

3.1 User Interface

When the application is launched, the user must write one username and choose either to perform or not the first move. The username will be used to maintain a configuration file about the successful movements of the player.

The user can, then, interact with the graphical interface, dragging the pieces from their origins to desired destinations. The moves are validated using the rules described in section 2.

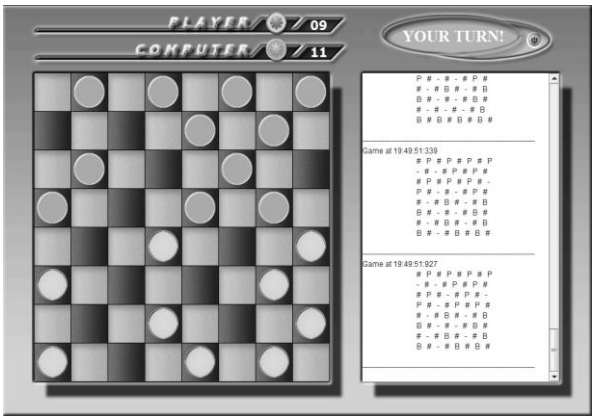

Figure 7: Application's Graphic User Interface

The right middle section of the interface shows the match move's history according to characters that will be presented in section 4. The upper left section shows the current number of pieces of each player. In the upper right section is shown the turn owner or the match result at its end. This is all shown in Figure 7.

3.2 Control of current turn and end of match

A model of finite state machine (FSM) is used to control the match's turns. The turns are defined as states of the application. The FSM of the application is in accordance with the proposals of Moore and Mealy, because it contradicts the traditional model, which is a model of acceptance/recognition and does not work with actions. The formal definition of the FSM is a tuple $<\Sigma, \Gamma, S, S_0, \delta, \omega>$, where [Wagner et al. 2006]:

- Σ is the input alphabet: a non-empty set of finite symbols;
- Γ is the output alphabet: a non-empty set of finite symbols;
- S is a non-empty set of finite states;
- S_0 is the initial state, and is one element of S;
- δ is the transition function for the states: $S \times \Sigma \rightarrow S$;
- ω is the output function. For Mealy model, the output function depends on the input alphabet (ω: $S \times \rightarrow \Gamma$) and the current state. For Moore Model, the output depends only on the current state (ω: $S \rightarrow \Gamma$).

The state machine developed in this application accords with Moore model. It has four game states, defined as: Game Start, User's Turn, Computer's Turn and Game Over.

The Game Start is the initial state of the application. In it is asked the username and who should have the first move. This response is taken as input and based on this it defines who has the first move. Reset the game is one action that destroys the current executions, forces the restart of the application and switches the value of the last initialization, changing the order of players' turns.

In the User's Turn, the FSM awaits the trigger of a transition to the corresponding state through a valid move. The state, then, shall make the move, which was, until then, merely a visual representation. It shall, also, verify if the turn is composed by more than one movement, validating its conclusion or not. The next step is to remove the captures pieces from the board, and check whether the match is over, using as grounds the rules in Section 2.

During the Computer's Turn there will be executed tasks relevant to the computer's movements. Initially, the state will make a selection of the best move, which characterizes the decision making of the machine. This is achieved through a genetic algorithm that shall be discussed further along in section 4. Similarly to the User's Turn, it is also verified if the match is over.

The Game Over state is achieved when the User's Turn state or the Computer's Turn state verifies, positively, that the match ended according to the game

rules. The following actions are, then, remove captured pieces, that are still visible, from the board, perform the animation relative to the match's finish, and end the FSM.

It is important to highlight that the transition function connecting states is not explicit. In the application's domain, the set of operations that evaluates a move is considered a transition function. This evaluation operations use the rules presented previously and the verification of the match's end, whether it is a victory or a draw.

4. The Genetic Algorithm

The Genetic Algorithm, or GA, is an evolutionary approach that uses a metaphor regarding its basic genetic material. It searches to use principles found in nature, based on Darwin's natural selection model, in order to obtain solutions for algorithmic problems. That being said, gene and chromosome are used to represent inputs of the problem in question.

Genetic algorithms are indicated in situations that require the analysis of a large number of variables, or moments that allow one or more satisfactory solution [Lopes 2001].

In the developed application, it was used the following definitions [Carvalho 2008]:

- **Individual**: represents a possible configuration of the board;
- **Chromosome**: is the data structure that represents the individual, being the current configuration of the board;
- **Population**: vector of chromosomes that represents the set of possible outcomes;
- **Adaptation of the individual:** defines how good is the chromosome as a solution for a given problem;
- **Selection:** a set of criteria that determines which individuals in the population will be reused in the search for a result of higher quality.

The GA is applied to a population *P* of individuals that evolves through the action of genetic operators in order to obtain a population *P'*, which would be a descendent of *P* [Lopes 2003]. The genetic operators are selection, mutation, crossover, inversion and substitution. This application's GA uses only the mutation operator, using two different types of it: mutation by sub-exchange and mutation by replacement. The mutation operator guarantees the creation of features in the individuals of the population that did not exist before.

The mutation by sub-exchange [Lopes 2003] receives a chromosome $a = <a_1, a_2, ..., a_i, a_j, ..., a_{64}>$ and returns a chromosome $a' = <a_1, a_2, ..., a_j, a_i, ..., a_{64}>$, where $1 \leq x_1 < x_2 \leq 64$.

The mutation by replacement is an extension of the mutation proposed by Holland, which used a binary alphabet. It receives a chromosome $a = <a_1, a_2, ...,, a_{64}>$ and an alphabet *U*, and returns a chromosome $\boldsymbol{a'} = <a_1, a_2, ..., \boldsymbol{a_i}, ..., \boldsymbol{a_j}, ..., a_{64}>$, where *i* and *j* are random positions in the vector with $1 \leq i < j \leq 64$. The bold characters correspond to the chromosome complement of the characters a_i and a_j, in the *U* alphabet.

4.1 About the application's GA

The initial population of the game consists of only one chromosome, due to the fact that the chromosome is a representation of the configuration of the board. In the implemented GA, the chromosomes are represented by a vector of 64 characters. It was agreed that the characters would be as follows: "**B**" would be a square with a white man in it; "**P**" would be a black man; "**A**" represents a white king; "**E**" would be a black king; "**-**" represents an empty square; "**#**" is a square that cannot be occupied; "**X**" is a square occupied by a computer's piece that was captures; and "**Y**" is a square occupied by an user's piece that was captured. They would obey the following restrictions:

- The maximum number of white or black pieces is 12;
- Each character occupies a position *i* from the vector that represents the chromosome, such that $0 \leq i \leq 64$;
- If the ratio of *i* per 8 is even, the odd positions of the board will be occupied by the character "**#**", and the even positions will be occupied by the remaining characters;
- If the ratio of *i* per 8 is odd, the even positions of the board will be occupied by the character "**#**", and the odd positions will be occupied by the remaining characters.

Thus, the chromosome corresponding to the initial configuration of the board may be represented as <*#, P, #, P, #, P, P, #, P, #, P, #, P, #, #, P, #, P, #, P, #, P, -, #, -, #, -, #, -, #, #, -, #, -, #, -, #, -, B, #, B, #, B, #, B, #, #, B, #, B, #, B, #, B, B, #, B, #, B, #, B, #, B, #>*, which is shown in Figure 1.

4.2 Creation of new chromosomes and adaptation factor of the individual

During the creation of new chromosomes, the GA performs a few tasks. Initially, it analyses whether or

not it is possible to move a piece. Then, it verifies if it is possible to capture an opponents' piece. The second action has priority over the first and, if it is satisfied, there will be generated only chromosomes that perform a capture. Finally, the most adapted individual will correspond to the computer's move.

Since the application lies in the domain of Checkers, it is necessary to modify some operations of the GA, in order to allow certain peculiarities of the game. For instance, in the context of the game, a move must always be executed. Although it may seem obvious, for the GA it means that a new chromosome always shall be generated to replace the current one. This replacement has to be made even when the new individual creates a bad scenario – of piece loss -, or else it will incapacitate the end of the match.

The adaptation factor of the individual is calculated in order to evaluate how good is the movement represented by the chromosome. For such, the GA considers the weights described in Table 1. Such values were assigned according to tests that aimed the best performance of the application.

Table 1: Weights W of the application

MOVE	WEIGHT
Simple move	1
Move that results in the lost of a piece to the opponent	-5
Move that results in saving a piece from being captured	2
Move that results in the capture of a piece	3

The developed game uses a storage system to allow a bigger evolution of the AI. Initially, the computer may only follow one of these strategies, in order of priority: to capture the greater number of opponents' pieces that results in the minimum number of computer's losses; to lose the minimum amount of pieces possible; or to make a random move.

When the computer starts playing against the user, its database is filled. The database, in the beginning stage of developments, was a simple archive system. Each line in the archive is divided into two columns. The first contain the initial configuration of the board, and the second has the configuration achieved after making the move. These configurations of size *t*, $1 < t \leq 64$, would be windows of parts of the board that were affected by moving the pieces.

When the user makes a move that results in the capture of a virtual player's piece, the algorithm would save the configuration before and after that specific movement.

Due to the fact that the pieces of the players chance color – and representations – at the end of each match, there were agreed standard values to represent the pieces in the saved files. The pieces of the computer would change from "**B**" and "**P**" to "**c**" – lower case – and from "**A**" and "**E**" to "**C**" – upper case. Similarly, the user's pieces would go from "**B**" and "**P**" to "**j**" and "**A**" and "**E**" to "**J**". These characters, when compared to the chromosomes of the game, would change back to their original values.

With these data, the GA's system can maximize the number of computer's pieces in the board, and minimize the number of user's pieces. To achieve this, it is used Equation 1.

$$W - \sum_{i=1}^{n} X \times N, with\ N = \begin{cases} 1, & if\ n\ is\ the\ configuration \\ 0, & else \end{cases} \quad (1)$$

Where W is the weight that represents how good is a move, according to Table 1. The chromosome fitness, calculated through Equation 1, shall be maximized when the move is one of capture, and there is not such a configuration in the database that result in a computer's piece being lost. The value of *X* is defined in Equation 2:

$$X = \frac{number\ of\ occurrences\ of\ configuration\ i}{total\ number\ of\ configurations\ \times 100} \quad (2)$$

This implies that the evaluation process, using the database, can be defined as follows:

1. Scan the database retrieving all records;
2. For each record found:
   a. While there are new records or the match is not over, do:
      i. If there is an initial scenario equals to the given final scenario, recursion to item 1 with this final scenario as initial scenario;
      ii. Otherwise, apply Equation 1 and return the results;
3. Count the total number of pieces of each contestant on the final scenario. Apply Equation 1 and return the results.

When applied to a game of Checkers, the algorithm aims to minimize the number of pieces lost by the computer; avoiding moves that have previously lead to an undesirable outcome. In other words, its objective is to maximize the computer's pieces ("**C**" e "**c**")) and minimize the user's ones ("**J**", "**j**" e "**-**"). The empty square "-" is included in the minimization due to the increasing of empty spaces be a consequence of the capture.

5. Results

There were performed two series of tests with this algorithm. The first consisted of using a human player against the AI. There were played 40 games, in which the computer won 52,5% of times. In Table 2 can be

found the number of configurations in the database at the beginning of each match; the average evaluation of the chromosomes, with and without Equation 1; the average evaluation of the selected individuals of population; and, finally, the game's result.

The second series of tests used a different approach of Genetic Algorithm to compare performance and average time obtained, in milliseconds, of the algorithms. The algorithm was the one that originated this study, and was proposed by Carvalho [2008]. In Table 3 is described who initiated the game, the average time of this paper's algorithm, the average time of Carvalho's algorithm and the result of the game, which is Defeat if Carvalho's win, and Victory if this paper's agent win.

Table 3: Test's Results

First Move	Average Time	Carvalho's Average Time	Result
Our algorithm	10.13	76.56	Defeat
Carvalho's	46.45	42.27	Defeat
Our algorithm	91.16	0.78	Victory
Carvalho's	62.74	28.39	Victory
Our algorithm	13.85	11.29	Victory
Carvalho's	70.77	39.85	Victory
Our algorithm	0.57	79.09	Defeat
Carvalho's	52.67	59.86	Defeat
Our algorithm	4.25	71.54	Defeat
Carvalho's	0.61	95.83	Defeat
Our algorithm	0.64	92.20	Defeat
Carvalho's	61.29	0.032	Victory
Our algorithm	76.65	17.42	Victory
Carvalho's	45.59	25.73	Victory
Our algorithm	79.95	0.00	Victory
Carvalho's	138.26	20.33	Victory
Our algorithm	33.46	30.60	Defeat
Carvalho's	1.0	3.20	Victory
Our algorithm	121.0	24.44	Victory
Carvalho's	40.68	38.39	Victory

The final results, regarding the games played against others AI techniques and against human players, are described in Table 4.

Table 4: Test's Results

Regarding games against other AI techniques	
Number of victories of the computer against other AI techniques	12
Number of defeats of the computer against other AI techniques	8
Total	20
Regarding games against human players	
Number of victories of the computer against human player	21
Number of defeats of the computer against human player	19
Victories in which there were differences between columns A and B (Table 2)	11
Defeats in which there were differences between columns A and B (Table 2)	9
Victories in which there were not differences between columns A and B (Table 2)	10
Defeats in which there were not differences between columns A and B (Table 2)	10
Total	40

6. Discussions

The application showed a good performance, being able to confront nicely a common user. It also showed a significant improvement when using the decision rules along with the weights of the GA, rather than just using the weights to evaluate chromosomes.

About the selection of pieces, the application showed a better performance with the moves of man instead of kings. This is because the range of action of the man is smaller, therefore being easier to predict the possible movements. It made clear that the algorithm can generate moves in which the computer just delivers its king to be captured, even though it was possible to perform a different movement. However, when using Equation 1 these situations became far less frequent.

Table 2 presents the results of 40 matches. Initially the database had 267 board configurations saved. This quantity increased during tests, and turned into 579 by the end of analysis. After a few games, it was possible to notice that the average evaluation of the chromosomes, with and without Equation 1, began to differentiate. The values obtained using the equation were superior to those obtained without it.

In the tests executed by a human player the machine began winning the matches. As the user improved, the machine's and the user's levels began to normalize. Even still, the virtual opponent presented a satisfactory result, showing that the system served its purpose of assisting the gamer to improve its abilities.

7. Conclusion

This paper proposed the use of genetic algorithm to improve the artificial intelligence of a virtual adversary, and, so, improve the user's experience. This thought lead to the development of a game of Checkers that used the GA in its decision-making process.

It was achieved, then, a tool that compel the user to evolve as he/she plays, having a progressive improvement of his/hers abilities. Although the results were satisfactory, the random factor of the algorithm makes the application unstable, being able or not to perform the best movement for a certain configuration of the board. Since the goal, however, was not to obtain an invincible game, but one that helps the user to develop, this fault shows little importance.

It stays as a future goal to continue to develop the system in order to make it able to classify the user

according to his/hers abilities in game, and to use this information to improve the realism and entertainment levels of games. The next approach would be increasing the trees of movements, since this methodology only used one level of search. The performance of the algorithm would significantly improve with a deeper tree search, therefore obtaining a better analysis of which piece to move and where.

References

CARVALHO, L.F. B. S. 2008. Um Jogo de Damas Evolutivo. Trabalho de Conclusão de curso, Instituto de Computação, Universidade Federal de Alagoas. Maceió.

KECMAN, V. 2001. Learning And Soft Computing - Support Vector Machines, Neural Networks, And Fuzzy Logic Models, The MIT Press.

LOPES, R. V. V. 2001. Um algoritmo genético em hardware para monitoramento de sinais vitais, Dissertação de doutorado em informática, Centro de Informática, Universidade Federal de Pernambuco.

LOPES, R. V. V. 2003. Um Algoritmo Genético Baseado em Tipos Abstratos de Dados e sua Especificação em Z. Tese de Doutorado. Universidade Federal de Pernambuco. Recife, PE.

RUSSUEL, S. AND NORVIG, P. 2004, Inteligência Artificial, 2ª Edição, Editora Campos.

SCHWAB, B. 2004. AI Game Engine Programming, Charles River Media.

SILVA, M. R. 2008. O Jogo De Damas para a Educação. Dissertação Pós Graduação, Universidade de Franca – UNIFRAN, Franca.

TATAI, V. K. 2003, Técnicas de sistemas inteligentes aplicadas ao desenvolvimento de jogos de computador, Dissertação de mestrado em engenharia elétrica, Faculdade de Engenharia Elétrica e de Computação, Universidade de Campinas.

WAGNER, F. AND SCHMUKI, R. AND WAGNER, T. AND WOLSTENHOLME, P. 2006. Modeling Software with Finite State Machines: A Practical Approach. CRC Press.

Xadrez Regional. 2002, 'Jogo de Damas - Regras Oficias'. Available in: http://www.xadrezregional.com.br/regrasdm.html, last accessed on August 03, 2011

Table 2: Algorithm performances against a human player

Configura-tions	Configura-tions without repetition	A: Average evaluation of the chromosomes using only the weights	B: Average evaluation of the chromosomes using Equation 1	Average evaluation of the selected chromosomes	Computer Results
267	243	-2.04	-2.04	1.0	Victory
269	245	-3.13	-3.13	-0.70	Defeat
279	254	-2.65	-2.65	0.41	Victory
282	257	-2.95	-2.97	-0.19	Defeat
291	264	-3.57	-3.57	0.47	Victory
295	268	-3.29	-3.60	-0.66	Defeat
304	277	-4.66	-4.66	-0.75	Defeat
311	284	-2.90	-2.90	-0.88	Defeat
319	292	-2.89	-2.89	-0.65	Victory
324	297	-1.21	-1.22	0.81	Victory
329	302	-3.80	-3.80	0.67	Defeat
338	311	-2.31	-2.42	0.35	Victory
347	320	-2.68	-2.68	-0.24	Defeat
355	328	-1.49	-1.49	0.68	Victory
360	333	-3.87	-3.87	0.00	Defeat
367	340	3.73	3.73	0.10	Defeat
376	348	-3.83	-3.86	-0.48	Defeat
383	354	-3.02	-3.17	-0.62	Defeat
391	362	-2.82	-2.82	0.07	Victory
399	368	-1.40	-1.43	0.21	Victory
458	421	-2.65	-2.65	0.56	Victory
458	422	-4.30	-4.30	-0.65	Defeat
468	429	-2.17	-2.17	0.54	Victory
470	431	-2.58	-2.58	1.09	Victory
474	435	-2.41	-2.42	1.02	Victory
477	438	-2.47	-2.47	1.08	Victory
480	441	-2.94	-2.95	0.57	Defeat
488	446	-1.43	-1.44	1.19	Victory
498	456	-4.88	-4.89	-1.18	Defeat
506	464	-2.54	-2.55	0.48	Victory
509	466	-4.91	-4.91	-0.82	Defeat
519	473	-1.79	-1.80	0.93	Victory
532	483	-3.86	-3.86	0.16	Victory
541	491	-2.59	-2.78	1.13	Victory
545	494	-3.70	-4.09	-0.18	Defeat
554	501	-4.56	-4.77	-0.17	Defeat
561	506	-1.13	-1.39	0.21	Defeat
570	515	-3.68	-3.84	-0.72	Defeat
577	521	-4.21	-4.65	-0.23	Victory
579	523	-2.68	-2.89	0.40	Victory