Discovering Combos in Fighting Games with Evolutionary Algorithms

Gianlucca L. Zuin*
Departamento de Ciência da Computação
UFMG
gzuin@dcc.ufmg.br

Yuri P. A. Macedo*
Departamento de Ciência da Computação
UFMG
ypamacedo@gmail.com.br

Luiz Chaimowicz
Departamento de Ciência da Computação
UFMG
chaimo@dcc.ufmg.br

Gisele L. Pappa
Departamento de Ciência da Computação
UFMG
glpappa@dcc.ufmg.br

ABSTRACT

In fighting games, players can perform many different actions at each instant of time, leading to an exponential number of possible sequences of actions. Some of these combinations can lead to unexpected behaviors, which can compromise the game design. One example of these unexpected behaviors is the occurrence of long or infinite combos, a long sequence of actions that does not allow any reactions from the opponent. Finding these sequences is essential to ensure fairness in fighting games, but evaluating all possible sequences is a time consuming task. In this paper, we propose the use of an evolutionary algorithm to find combos on a fighting game. The main idea is to use a genetic algorithm to evolve a population composed of sequences of inputs and, using an adequate fitness function, select the ones that are more suitable to be considered combos. We performed a series of experiments and the results show that the proposed approach was not only successful in finding combos, managing to find unexpected sequences, but also superior to previous methods.

1. INTRODUCTION

One of the tasks in game design is to define an environment with a predetermined set of rules that dictate how the player should interact with the game and evolve its state. In video games, programming these rules is a non trivial task and is subject to design flaws and bugs. Moreover, testing can be very time consuming in game development, and various titles end up being delivered with some design errors.

In many games that rely on physics simulations, even though the designed environment follows a set of predetermined rules, predicting all emergent behaviors can be a difficult task. Sometimes these unexpected behaviors can be embraced by designers, creating features that can be expanded and designed upon. But they may also break player's immersion, and maybe even go as far as to ruin the game for that player. This is commonly observed in some game genres, such as the fighting games.

Fighting games are very popular and have been around for many years. The largest fighting game competition, Evolution Championship Series (EVO)[1], formerly Battle by the Bay, was founded in 1996 and, in 2015, had over 19 million views on the streaming website twitch.tv [15]. Due to their competitive nature, they require a careful balancing between the characters, such that the gameplay can be perceived as fair by the both players. Although these characters are overall designed to be equally efficient, sometimes they may feature some unexpected behaviors due to the large number of play combinations. One example of these unexpected behaviors is the occurrence of long or infinite combos.

In fighting games, whenever a player strikes a blow to another, the player taking the hit becomes temporarily stunned and unable to take actions. Should the attacking player perform another attack that is fast enough to hit an opponent before he is able to recover, this sequence of attacks becomes a *combo*. More specifically, a combo [18] is a sequence of attacks by a player that, while damaging an opponent, does not allow it to take any actions. While this sequence of attacks should eventually end allowing the opponent to retaliate, infinite combos, as the name suggests, do not end as long as the attacking player does not miss his correct inputs. These types of combos are usually the result of unexpected interactions between a character's actions, where a sequence of attacks loops with itself indefinitely.

Despite how negative infinite or long combos can be to a player's experience, it is not correct to say that a combo is an undesired feature in fighting games. Combos are an integral part of many action games, rewarding precise execution of commands and encouraging players to keep on practicing. Competitive fighting game players strive to perform long combos that take the most out of any of their opponents' vulnerable moments. As fighting games are also an electronic sport that is commonly watched live or through

GECCO '16, July 20-24, 2016, Denver, CO, USA
© 2016 ACM. ISBN 978-1-4503-4206-3/16/07…$15.00
DOI: http://dx.doi.org/10.1145/2908812.2908908

[1]http://evo.shoryuken.com/

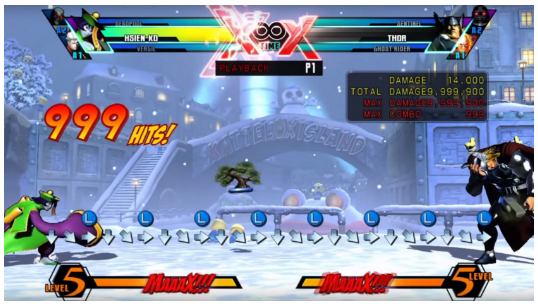


Figure 1: An infinite combo from the game *Ultimate Marvel Vs Capcom 3*. Combos in this game are longer than in most games, but even then it suffers from combos that have made some characters be considered much superior to others.

online streaming, combos can also can be very entertaining for viewers of competitive matches.

An important task for adequately balancing fighting games is to discover these combos. The biggest problem, however, lies on the exponential number of combinations of attacks, which makes the task of predicting all of their outcomes, and consequently detecting combos, a very hard one.

In this paper, we propose the use of an evolutionary algorithm to successfully find combos on an open source fighting game, by discovering which sequences of player inputs are likely to generate combos. The main idea is to use a genetic algorithm to evolve a population composed of sequences of inputs and, using an adequate fitness function, select the ones that are more suitable to be considered combos.

Evolutionary algorithms allow us to trim the exponentially large search space of attack combinations, making the search for longer combos much easier. Also, by automating this task, we are able to attempt combinations of actions that normally would be counter-intuitive for manual testing standards, allowing the discovery of new combos.

The remainder of this paper is organized as follows: Section 2 discusses some works regarding Artificial Intelligence techniques applied to fighting games and non-traditional areas of game design. Section 3 presents the problem definition while section 4 describes our proposed methodology. In section 5, we present our experiments and discuss the obtained results. Finally, Section 6 brings the conclusion and directions for future work.

2. RELATED WORK

Several works on Game AI rely on evolutionary algorithms, but most of them focus on how to improve the artificial intelligence of computer controlled players. As stated by Yannakakis and Togelius [17], the use of AI techniques in non-traditional areas of game design seems to be a new trend in academic research. For example, the authors of [12] discuss procedural content generation tailored to player experience in platform games, specifically the Infinite Mario Bros. Single-layer and multi-layer perceptron networks (SLP and MLP) are used to approximate the affective state of the players, select relevant features and improve the game. In [1], the authors propose the use of a general game system not only to synthesize but also evaluate new games. Heavily relying on previous works concerning what makes a game fun or interesting, they establish several metrics to achieve their goal. Another example is [14], which illustrates an attempt to use computational intelligence to generate a game. Through an evolutionary algorithm, the game controllers and rules are improved.

When it comes to fighting games, it is difficult to find any academic research that does not focus on improving the game AI. The authors of [11], for example, focus on creating a bot capable of mimicking a human player. The bot uses a Naive Bayes classifier over all possible actions and a finite state machine to choose a set of action that would better mimic a player. The different sets attempt to reproduce different kinds of human behavior during a match, like playing more aggressively when ahead or conservatively if overpowered. The bot developed by Yamamoto et al. [16], in turn, attempts to predict its adversary course of actions through the K-Nearest algorithm. With that information and the knowledge that some actions are advantageous against another, it deploys the appropriate countermeasures. Graepel, Herbrich and Gold [4] explore the use of reinforced learning with different reward functions trying to learn relevant data by inference in a Markov decision process. Their objective is to develop an AI with good policies in a commercial game.

In previous work [18], we tackled the problem of finding large combos in fighting games using Hidden Markov Models [10]. Both supervised and unsupervised learning algorithms were used but due to the excess of noise and particularities of the implemented model, we were unable to successfully predict combos. Changing the minimal discrete time interval to a player action, rather than a game frame, we were able to identify small combos lasting up to 112 frames. In this paper, we propose a different methodology, and use an evolutionary method to discover combos.

3. PROBLEM DEFINITION

A common pattern in game programming is the definition of a "game loop" [9] where, at each turn, inputs are processed, the game state is updated, and the game is rendered. This loop executes once for every frame, which is the shortest discrete time interval in a game.

Most fighting games display a fully deterministic behavior, which means that, under the same circumstances, whenever the same action is performed, the outcome is the same. It also means that no event relevant to the gameplay relies on any sort of random timer or choice. For fighting games that follow this behavior, any state within the game can be described by the sequence inputs, one at each frame, that led to that point. Therefore, by giving a specific set of inputs to a fighting game, that set of inputs translates into actions that may or may not result in a combo.

Every game requires some sort of input from one or more players. This can be done by the pressing of a button, the click of a mouse, a voice or motion command, and so on. In fighting games this is usually given in the form of directional and button commands, that are performed within a match. The former is determined by the direction that a player is pushing a directional pad, analog stick, arrow keys, or other interface for directional movement. The latter refers to every other button a player can press that causes his character to perform an action when pressed by itself or with another combination of buttons or a directional command.

This input has to be processed by the game somehow. For each frame, the player must feed the game with a command, even if that command is "do nothing". Many games that are multiplatform redirect button commands into another class of input that is the same for all gaming platforms. Such as interpreting the "triangle button" from a console, and the "Y button" from a another console, into the same "Medium Attack" input. This is extremely useful to allow players to study and play the game regardless of its platform, as well as being an organized way for the game to allow its players to redefine their button settings. Game franchises such as *Guilty Gear* and *Street Fighter* kept the same input name patterns across multiple console generations for convenience purposes.

Through this representation, the problem then becomes finding a sequence of n player inputs that generates a large combo for n frames or less. Each singular input is completely independent of the others. This imposes an exponential search space that represents each possible input combination in the game. For example, if a fighting game has 36 possible inputs in each frame, a sequence of 300 frames would allow for 36^{300} possible input sequences within a 300 frame interval. Thus, the complexity of the input search space for any fighting game can be represented by c^n, where c is a constant that differs for each fighting game and represents the number of possible inputs, and n is the number of frames which we are interested in evaluating.

In this paper we focus on finding combos for a game we developed in [18], which is a replica of *Street Fighter 2* developed in the Gamemaker engine.

4. METHODOLOGY

This section describes the evolutionary algorithm modeled to find combos in fighting games. We start by given a brief description of the game platform where we find combos (Section 4.1). Considering this platform, each individual represents a sequence of input commands given by a player, as detailed in Section 4.2. Individuals are evaluated by executing the game for 300 frames, where combos are identified (Section 4.3). After fitness evaluation, individuals are selected to undergo crossover and mutation operators through a tournament selection. The best individual is inserted into the new population without any modifications (elitism). This process goes on until a maximum number of generations is reached.

4.1 Game Platform

The game platform we used in our experiments is detailed described in [18]. It was developed as an open source fighting game for academic research purposes[2]. The game is implemented with Gamemaker, a tool for game development that has both free and proprietary versions. Either version can be used to edit any of the game's resources, such as sprites and code. It follows the *Street Fighter* formula, mixed with a simplified version of the *Marvel vs Capcom 3* control scheme, which includes directional commands and three buttons for attacks. These attacks are classified as 'light', 'medium' and 'heavy', and each mapped to a button. Each attack has variations according to the attacking character's state, including variations that require a directional input before pressing a button. Special moves can be performed through a sequence of commands, which includes both directional commands and attack buttons.

[2]Available at https://github.com/GZuin/finding-combo

4.2 Individual Representation

In our methodology, each individual represents a sequence of commands given by a player. The game we focus on uses directional commands represented using the "numpad notation", which is the notation used within most fighting game communities. This notation assigns numbers from 1 to 9 to directions as with the arrows of most keyboards with a numerical pad *(1 = down-left; 2 = down; 3 down-forward; 4 = left; 5 = neutral (no direction pressed); 6 = right; 7 = up-left; 8 = up; 9 = up-right)*. As for the button commands, the game identifies four types of inputs: a light attack *(l)*, a medium attack *(m)*, a heavy attack *(h)*, no button pressed.

Figure 2 shows the genotype representation. We used a fixed length representation, where each input corresponds to a gene. Since an input consists of a directional command *(1 to 9)*, and a button command *(- for no button pressed, l, m or h)*, there are 36 possible values for each gene. Considering that the game runs at 30 frames per second, if we were to look for a combo that lasted, for example, ten seconds, each individual would be a sequence of 300 genes. Most of our tests are interested in finding combos in this interval of time. Should a combo actually last that long in the game, the opponent should be defeated with no chance of reaction. Of course, this is different for every fighting game, and the ideal number of frames required for testing is something that should be decided by the designer or by someone with extensive knowledge of the game mechanics.

The sequence of inputs from a individual is translated into game actions, also illustrated in Figure 2 (phenotype vector). However, note that we are not interested into the characters' actions themselves, but on whether the opponent reacted to these actions. Hence, during the fitness evaluation process, after the elapsed time (determined by the number of genes) ends, the game feeds back to the algorithm the information on how well that sequence performed. This information is mainly the state of the opponent at each frame: (i) "0" meaning that the opponent was not in a combo during that frame, and therefore could act, (ii) "1" to represent that the opponent is currently unable to act due to being in a combo.

Figure 2 illustrates these ideas and the relations between frames, showing how inputs might translate into attacks by the player, and how these attacks change the opponent's state when they hit. In the player's current state, a 5l input translates into a "Standing Light Attack", while the 2h input translates into a "Crouching Heavy Attack". The 5- inputs after the attack translate into the player not performing any actions. The opponent's state during these attacks is reflected by the combo sequence of zeroes and ones.

4.3 Fitness Evaluation

For each individual, the fitness evaluation procedure involves executing the game. Even though the game state is defined by both players' actions, in our experiments only player one receives inputs to perform actions, while player two remains idle during the whole test (all inputs for player two are the game's equivalent to "do nothing"). This is a limitation that we have imposed to restrict our search space, as different combos can be performed depending on the receiving player's current action. For instance, if the receiving player is jumping when the combo starts, the attacks that hit him may have different effects. This is also true for in-

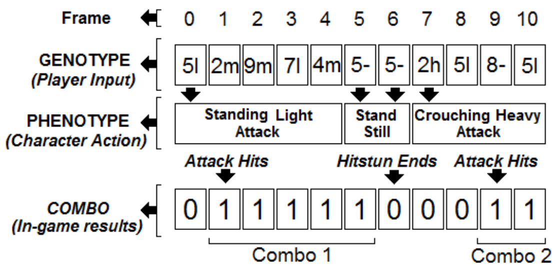


Figure 2: "Combo 1" is a 5 frame combo that begins when the first attack hits. The 5l input is pressed again at frame 8, but it has no effect as the player is busy with another attack that started on frame 7.

stances where the receiving player is on distinct positions in the fighting stage.

The search space for every combo in a fighting game includes every possible position and action being performed by the receiving player when a combo begins, even if many of these actions have little or no influence on the game state when the combo starts. While we are certainly interested on further expanding our experiments to cover more of this space, in this work we focus on every instance where the receiving player is standing at the 'corner' or 'edge' of the fighting stage, which represents one of the best situations in which a combo can be performed.

In order to evaluate an individual, the input reading function of the game was changed to feed player 1 with commands from a file containing inputs for every frame. We also write at every execution of the game an output file containing the frames in where player 2 is stuck in a combo caused by player 1 (one character per frame, "0" for "not in combo", and "1" for "in combo"). To optimize training and execution times, the game was stripped of most of its graphical and sound resources and its speed was increased up to 330 times the original game speed, resulting in the maximum number of frames per second allowed by the Gamemaker engine.

The algorithm uses this information to determine the fitness of the individual by taking into account the following, during the time frame determined by the number of genes (number of inputs in the sequence):

1. How long is the longest combo.
2. How long are the other combos in the same sequence.
3. How easily would it be to append each combo to its adjacent combos in the same sequence.

In order to better understand the concept behind the third clause, it is necessary to comprehend that a large combo is not more than a sequence of smaller combos. For example, a punch that stuns the opponent for 10 frames is completely eligible to be considered a combo of 10 frames. If that punch allows for the attacker to hit with a kick that stuns for other 7 frames at the last possible moment, then this sequence of attacks is considered a combo of 17 frames. What this means is that a sequence of inputs, specially random and sub optimized ones, is likely to generate long sequences of 0's that represent moments where the opponent is idle and unharmed, interlaced with long sequences of 1's that represent the amount of stunning that opponent received from an attack. Knowing this, we can assume that the smaller the sequence of 0's between two sequences of 1's, the easier it should be for these two separate attacks to be performed without allowing the opponent to recover. Our fitness function makes use of this, evaluating not only the largest combo generated by a sequence of inputs, but also the other combos that could likely be appended. Therefore, the fitness of a given combo can be evaluated as the sum of the fitness of each of these sequences of 1's, as described in the following equation:

$$F = \sum_{i=1}^{n} f(i), \quad f(i) = \begin{cases} \dfrac{ComboSize}{(1 + LeftZeroes)^2} & \text{if right of longest} \\[2ex] \dfrac{ComboSize}{(1 + RightZeroes)^2} & \text{if left of longest} \\[2ex] ComboSize & \text{if the longest} \end{cases}$$

where $f(i)$ denotes the fitness of each sequence of ones, and n denotes the number of sequences of ones. For the longest combo, $f(i)$ corresponds to its size. Otherwise its value is decreased by the square of 1 plus the number of zeroes until another sequence of ones (towards the largest sequence).

Basically, when the game returns the state of the opponent at each frame, we first search this string for the longest sequence of 1's and determine how long is the longest combo generated by the input sequence. Then, we look for each adjacent combo, initially setting a value equal to its size. Each combo to the left of the largest one has its value decreased exponentially by the size of the streak of 0's to its right, while each to the right of the largest has its value decreased by the size of the sequence of 0's to its left. This way, only long combos that are separated by very small intervals of 0's are truly valuable to the fitness, in the hopes that these might be appended to the largest combo in future generations.

4.4 Genetic Operators

Since a combo is a sequence of inputs and changing neighbors bits might destroy combos, we decided to use a one-point crossover. As the immediate neighbors of a gene do not interfere as much in the actions as farther ones might, crossover can potentially change a relevant input that is not contributing to a combo. A uniform mutation operator was applied.

4.5 Dealing with Neutrality

The neutrality phenomenon was first addressed by Kimura, and *"assumes that only a minute fraction of DNA changes in evolution are adaptive in nature, while the great majority of phenotypically silent molecular substitutions exert no significant influence on survival and reproduction and drift randomly through the species"* [7].

This scenario fits the problem we address in this paper. One particular characteristic of our approach is that the representation we use allows for a many-to-one genotype-phenotype mapping, which end up generating many neutral mutations [5]. As explained before, the genotype of our individuals are sequences of inputs performed by the player, while the phenotype is the resulting actions performed by the character inside the game. Figure 3 illustrates these two characteristics of our problem: in the first set of inputs we make a "Standing Light attack" (5L). We then press the buttons for a "Backward Heavy Attack", two backward jumps, and another "Standing Light attack", (4H, 7-, 7- and 5L respectively). However, during the time that the inputs 4H, 7-, and 7- were given, the character was stuck in the animation for the first attack. This made the following four inputs to be ignored, and only after four frames the character was able to perform another action, which would be the input 5L. The second set of inputs, although different, resulted in the same set of actions by the same logic.

Another particularity of the problem is that some specific inputs can make this 'ignored' interval shorter. In this very example, the animation of the move "Standing Light Attack" (5L) can be canceled with another "Standing Light Attack" after four frames, unlike most moves in the game (usually, it would take 6 frames to perform another action after a "Standing Light Attack"). If a player were to press other inputs rather than 5L, the character would most likely not finish its animation from the first attack for at least two more frames.

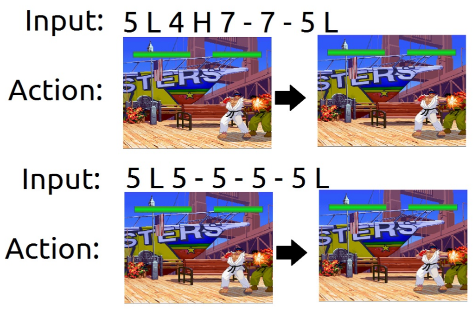


Figure 3: Two sets of inputs and the resulting actions when fed to the game. Even though both input sets are different, the resulting actions are the same.

Because of the scenarios aforementioned, the problem we tackle presents a high number of introns: inputs that do not contribute to the game state since they did not result in any in-game action. This issue presented serious challenges in our previous probabilistic machine learning approach [18]. In the proposed approach, the effects of neutrality [7] in the search are those expected. The fitness of the best individual goes through long periods of steadiness, where the population seems not to be doing anything useful (the equilibrium phases) [13]. However, in most problems neutrality can provide a buffer to guide the population towards better individuals [2], so that the search switches to the transient phase, as illustrated in Figure 4.

Many works in the literature have been developed concerning neutrality [2, 6, 13], and the general consensus as how to handle it seems to be a high mutation rate or by applying local search operators to skip local optimum. We propose a simple local search operator to easy the effects of neutrality, but that is cheap enough to be frequently applied.

The local search operator selects a segment of n genes from one individual and evaluates all possible resulting combinations of changing one gene. Since there are 36 different kinds of inputs in the game, the operator analyses $35n$ sequences (there are only 35 options for each gene, as the 36^{th} would yield the same sequence). The local search is applied only to the best individual and does not necessarily occur at every generation.

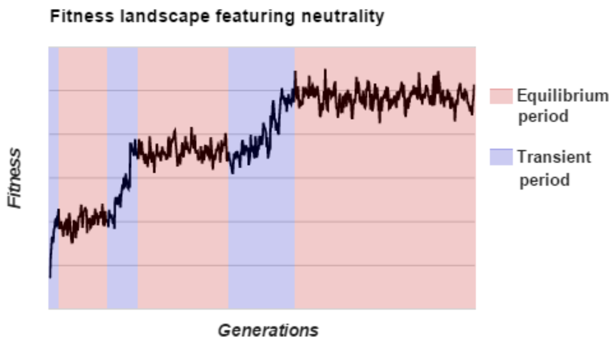


Figure 4: Example of a fitness function in an experiment featuring neutrality. Long periods of equilibrium where little is gained are followed by brief transient periods, where a quick growth in fitness is observed.

We also experimented with very high levels of mutation to overcome the effects of neutrality. However, further investigation on alternative approaches to deal with this problem are left for future work.

5. EXPERIMENTAL RESULTS

This section presents the results obtained when running the proposed method to find combos in games. Throughout this section, we present different experiments performed in order to tune the algorithm parameters and understand the effects of neutrality. The method was compared to the approach proposed in [18], which uses HMM, and a greedy hill climber method. The hill climber (HC) was implemented considering the same number of solutions evaluations as the EA performed.

5.1 Parameter Tuning

As the fitness requires playing the game, we set the parameters of the method with its running cost in mind. The execution time depends of three parameters: the number of generations, the number of individuals and the number of frames. We also performed 10 executions for each experiment due to the stochastic nature of EA.

Due to neutrality, the mutation probability received high values for genetic algorithm standards in all experiments. In order to give an idea of the proportion of introns vs exons, it was usually between a 5 to 1 and a 10 to 1 ratio, meaning that the majority of individual's genes are introns.

Figure 5 shows the results of the experiments obtained with the best parameter after preliminary tuning: 200 individuals evolved for 300 generations with crossover probability (p_c) of 0.85, mutation probability (p_m) of 0.5 and tournament size (k) equals to four. Unless stated otherwise, the size of the individuals (which relates to the number of frames being evaluated) was set to 300. In the experiments reported in this section, the local search operator is not considered in the search. Note that for this first experiment the results obtained by the proposed method (EA-best) were better than those obtained by the HC but worse than those obtained by the HMM. We also show the value of the average fitness of all individuals in the EA population, to give an idea of convergence. Notice that after 100 generations only small changes in fitness values are observed for the best individual, and that the average fitness of the population does not present a great variation during the evolution.

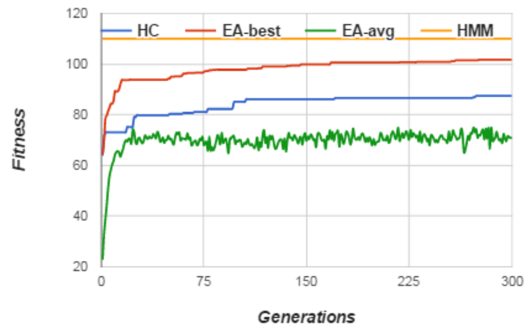


Figure 5: Results obtained by the EA with 300 generations, 200 individuals, $p_m = 0.5$, $p_c = 0.85$, $k = 4$. Hill-climbing (HC) is inferior to the evolutionary approach (EA-best), which is not far from the population average fitness (EA-avg).

Although this could be the effect of neutrality, by looking at the average fitness of all individuals from the population, we observed that it actually converged quickly to a small interval of values. We look at these individuals and observed that most of them represented small variations of the same combo, reflecting little or no improvement in fitness values. This reveals another characteristic of our problem: the combos we are interested in, which are the highest hills of the search space, are surrounded by valleys. The best combos known to our game (and for most games) usually occur due to sequences of two or more very specific actions that would be detrimental to the combo anywhere else. Having these moves within most combos would impact negatively into the fitness value.

5.2 Addressing Neutrality

Experiments in the last section showed that neutrality is strongly present in the search. We tried to deal with that in three ways: using the local search operator [3]; replacing part of the population to force new genetic material to be inserted into it; further increasing mutation rates.

The local search is applied after the results of the best individuals are not improved over m generations, were m was set as 20. However, due to computational costs, for a given individual, the local operator is applied in a subsequence of 10 genes. As observed in Figure 6, the results of the search process are very similar to those showed in Figure 5. However, around generation 150, once the EA gets out of the local peaks with the help of the local search operator, it quickly improves, finding new better individual. The biggest drawback in this approach is indeed the increase of computational effort. By looking at a segment of 10 genes we are evaluating 350 combo sequences.

In a second attempt to avoid local peaks, we changed the dynamics of the algorithm to allow population replacement. For each generation, we discarded the n% worse individuals and randomly generated new ones from scratch. Figure 7 illustrates the effect of this change when replacing 10% of the population (20 individuals). The other parameters were kept the same from the previous experiments. Note that the results produced are now better than those obtained by the baseline, although the average fitness drops due to the insertion of random individuals at each generation. An analysis shows that the periods of stability of the best fitness function now may be related to the effect of neutrality.

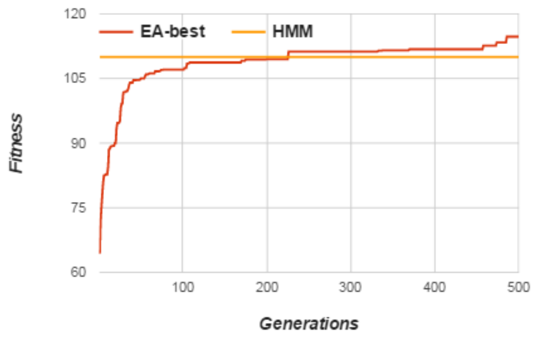


Figure 6: Effect of local search applied every 20 generations. The other EA parameters were left unchanged.

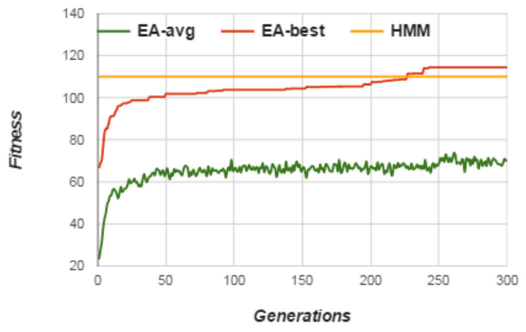


Figure 7: Results after 10% individual replacement (EA with 300 generations, 200 individuals, 10% pop. replacement, $p_m = 0.5$, $p_c = 0.85$, $k = 4$).

We also played with the individual replacement rate. We run a variety of tests with different values, and surprisingly high values of replacement (up to 50%) lead the EA out of the local points of the search space, as shown in Figure 8.

The results of this last experiment were a surprise: the algorithm discovered combos where two of the *Heavy-Shoryuken* special attack could be linked together, which caused our fitness to increase drastically. As a fact, in the combo before the first *Heavy-Shoryuken*, no other attack in the game could be used to extend it other than a second *Heavy-Shoryuken*. As the game developers, we were not aware that this action combination could be performed, specially in the way the individual performed it. The EA found this result on more than one execution. Compared to all other attacks, this one is known to be the best to finish a combo with. It is also the same attack that ended the combo in the best result obtained with the baseline. The fact that the algorithm chooses to perform this move is very promising, since this special attack requires the player to perform a minimum of

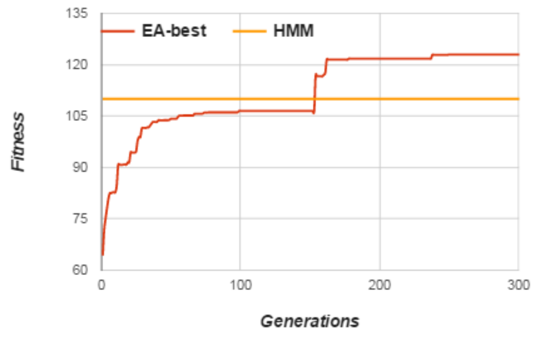


Figure 8: Results after 50% individual replacement (EA with 300 generations, 200 individuals, 10% pop. replacement, p_m = 0.5, p_c = 0.85, k = 4).

four very specific inputs in sequence. The EA was consistently able to create and reproduce this ideal sequence in most experiments.

Finally, aiming to improve even further the exploration of the search space, we increased even further the mutation rate. As previously mentioned, a high mutation probability can help to avoid local maximum and shorten the period of time the population goes through no changes due to neutral mutations (i.e. those that effect the genotype but not the phenotype). The results are presented in Figure 9, where we also show the effects of mutation rates more similar to those used in most evolutionary algorithms (0.10). Note that the results of low mutation rates are worse than those obtained with 0.5 or 0.8, but that the differences when increasing substantially the mutation (0.8) do not improve the results further.

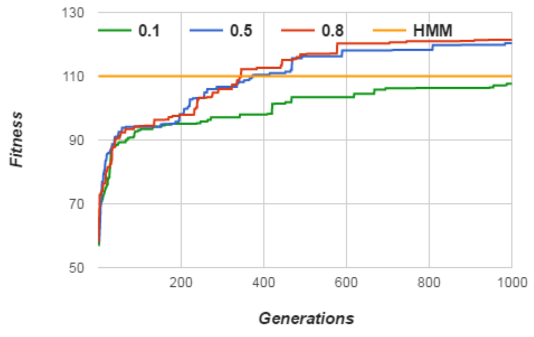


Figure 9: Effects of different mutation probabilities.

5.3 Effects of individual size

After introducing individual replacement to the population and improving significantly the results, we performed a different experiment. We tested different sizes of individuals, i.e., the number of frames from the game evaluated to find combos. Bigger individuals might lead to better results, as more frames also mean more chances to start a combo, once the chromosome is larger. However, evaluating them also takes more time. With that in mind, we reduced the population size, looking for a better trade-off between solution quality and computational complexity. However, note that this experiment have the same execution time as previous ones, where a total of 18,000,000 frames were evaluated (i.e. we have less individuals but more frames per individual). The results obtained are shown in Figure 10, and are very similar to those obtained with smaller individuals (see Figure 7). This indicates that the algorithm generalizes well, and is also appropriate to identify combos in larger time periods of the game.

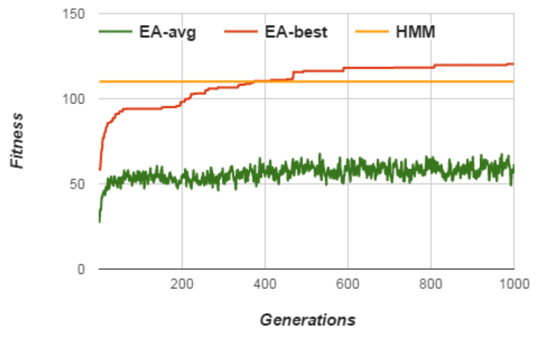


Figure 10: Variations of individual size (i.e. number of frames).

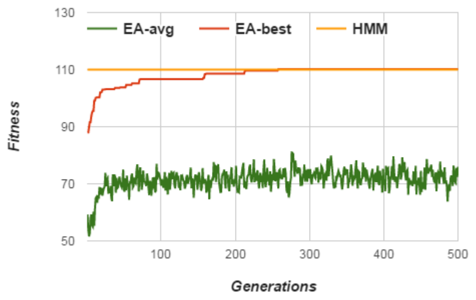
5.4 Adding Knowledge to search

Finally, we used our previous knowledge of the game as starting points for the search as an attempt to guide the algorithm towards promising areas of the search space at early generations. We developed 46 different combos lasting up to 70 frames, and built an initial population of 200 individuals as random concatenations of these combos until the 300 frame limit was met. Figure 11 shows the results obtained with the same parameters used in all experiments so far, but first without and then with individual replacement. As expected, with no individual replacement, the first generation has a very high average fitness (above 50, while in previous experiments this value never reached 50) but then gets stuck in local maximum. When individual replacement is used, in turn, the average fitness starts high but then drops quickly. Further investigation on the best way to combine previous knowledge about the game and random combos is left for future work.

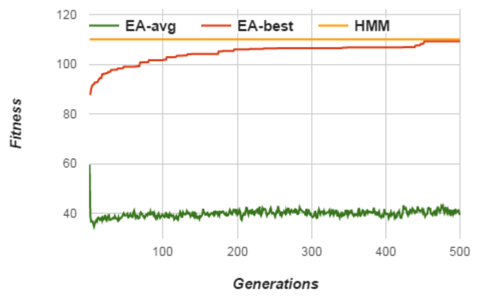
6. CONCLUSION AND FUTURE WORK

The application of AI techniques to non-traditional areas of game design is an important research topic. This is especially true in fighting games which lack this type of work. In this sense, this paper presented an evolutionary approach to detect combos in fighting games./ Basically, a population of individuals representing sequences of inputs was evolved using genetic operators such that the sequences representing longer combos would be selected. The proposed approach was consistently able to find combos under many different configurations of parameters. Some occasional results were even able to detect combos that were not known to be possible within the game platform. It is worth noting that combos are an important feature in fighting games and their correct identification can be crucial in developing a balanced game.

We believe that with proper parameter tuning, this method could be improved and adapted to other fighting games. For example, there are many possibilities to start and end a combo, making it relatively hard to obtain a better than average combo by chance. Therefore, creating a controlled



(a) Individual replacement.



(b) No individual replacement.

Figure 11: Effects of inserting previous knowledge to the initial population.

starting population may limit the potential of the algorithm biasing its result. One possible experiment could be the adaptation of the reinsertion operator to also use segments from a predefined combo list. This would result in a population that has randomly generated individuals as well as more controlled ones, possibly further promoting variability. The effects of neutrality also need to be further investigated, although the replacement of individuals and the local search helped in the exploration of the search space.

Another possibility is to use *niching*[8]. Not only this would promote diversity, but also help finding various large combo in different local optimum. Since each execution of the evolutionary method takes a long time, the optimal scenario is finding more than one large combo in a single execution. One of the main challenges in this scenario is defining when two combos belong to the same species. The exact same subset of actions can lead to a totally different set of states. We have identified that the combos are highly dependent of the first attack and that the evolutionary method rarely diverges from this first move, discarding most individuals that do so. This evidence could be a starting point for a classification method of different species. Finally, the fitness method could be modeled under a multi-objective approach, which would also improve diversity while returning a subset of individuals as potential combos.

7. ACKNOWLEDGMENTS

The authors would like to thank CAPES, CNPq and Fapemig for their financial support.

8. REFERENCES

[1] C. Browne and F. Maire. Evolutionary game design. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(1):1–16, 2010.

[2] E. Galván-López, R. Poli, A. Kattan, M. O'Neill, and A. Brabazon. Neutrality in evolutionary algorithms... what do we know? *Evolving Systems*, 2(3):145–163, 2011.

[3] R. Garg and S. Mittal. Effect of local search on the performance of genetic algorithm. *International Journal of Emerging Research in Management & Technology*, 3(6):404–407, 2014.

[4] T. Graepel, R. Herbrich, and J. Gold. Learning to fight. In *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, pages 193–200, 2004.

[5] M. A. Huynen, P. F. Stadler, and W. Fontana. Smoothness within ruggedness: the role of neutrality in adaptation. *Proceedings of the National Academy of Sciences*, 93(1):397–401, 1996.

[6] Y. Katada, K. Ohkura, and K. Ueda. An approach to evolutionary robotics using a genetic algorithm with a variable mutation rate strategy. In *Parallel Problem Solving from Nature-PPSN VIII*, pages 952–961, 2004.

[7] M. Kimura et al. Evolutionary rate at the molecular level. *Nature*, 217(5129):624–626, 1968.

[8] S. W. Mahfoud. Niching methods for genetic algorithms. *Urbana*, 51(95001):62–94, 1995.

[9] R. Nystrom. *Game programming patterns.* Genever Benning, 2014.

[10] L. R. Rabiner and B.-H. Juang. An introduction to hidden markov models. *ASSP Magazine, IEEE*, 3(1):4–16, 1986.

[11] S. S. Saini, C. W. Dawson, and P. W. Chung. Mimicking player strategies in fighting games. In *Games Innovation Conference (IGIC), 2011 IEEE International*, pages 44–47. IEEE, 2011.

[12] N. Shaker, G. N. Yannakakis, and J. Togelius. Towards automatic personalized content generation for platform games. In *AIIDE*, pages 63–68, 2010.

[13] T. Smith, P. Husbands, and M. O'Shea. Neutral networks and evolvability with complex genotype-phenotype mapping. In *Advances in Artificial Life*, pages 272–281. Springer, 2001.

[14] J. Togelius and J. Schmidhuber. An experiment in automatic game design. In *Computational Intelligence and Games. IEEE Symposium On*, pages 111–118, 2008.

[15] Twitch. Evolution 2015 smashes viewership records. https://blog.twitch.tv/evolution-2015-smashes-viewership-records-4447726169a2, 2015. Accessed: 2016-04-12.

[16] K. Yamamoto, S. Mizuno, C. Y. Chu, and R. Thawonmas. Deduction of fighting-game countermeasures using the k-nearest neighbor algorithm and a game simulator. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, pages 1–5, 2014.

[17] G. N. Yannakakis and J. Togelius. A panorama of artificial and computational intelligence in games. *IEEE Transactions on Computational Intelligence and AI in Games*, 2014.

[18] G. Zuin and Y. Macedo. Attempting to discover infinite combos in fighting games using hidden markov models. In *SBGames*, pages 149–157, 2015.