Rule-Based Agent for Hanabi

Gonzalez, Ricardo J

CE881 - Game Artifical Intelligence
University of Essex, UK
rg20332@essex.ac.uk

Abstract. The purpose of this research was to found the best Genetic Algorithm to create the best possible Rule-Based agent.
After tuning the functions and variables used for the GA, a Rule-Based agent who outperformed other sample agents.

1 Introduction

1.1 Why Game AI research?

Games have always been a source of challenges for AI researchers, due to the possibility of testing sophisticated decision making algorithms in an environment that provides very defined rules and restrictions. These algorithms end up being divided in rule-based agents, these will follow rules both doesn't learn from previous games played by itself or by others. Meanwhile the other approach is by learning agents, these agents are the ones that learn from previous matches played by itself or from the ones played by other agents or human players.

1.2 Research on Hanabi

These previously mentioned reasons led to several researches done in a wide variety of games, being Hanabi one of them. This games brings an interesting challenge due the cooperative nature that has the game. This had led to several researches attempting to find an agent able to perform well in the game, one of these is the research done by the DeepMind team, in which they researched how learning agents compares to rule-based agents and concluded that learning agents were lacking behind the rule-based agents [1].

Another popular research done on Hanabi was regarding the use of the Information Set Monte Carlo Tree Search (MCTS) technique. This allowed James Goodman to be the winner of the 2018 Hanabi competition. Overall what did was to create a non-complete decision tree (due to time limit per move in the competition). Using as a support a Bayesian opponent model, due to not being able to know the identity of the other players, those allowing overall being flexible enough to be able to handle playing with any other player [2].

1.3 My approach

I want to make an agent that would perform better than average, not focusing on a specific type of players who will play alongside myself. Based on this, and in addition of the researches already done, it seems like the learning agents despite the large amount of data that are being used to be trained, they don't perform very well.

Thus decided instead focusing on developing the best rule-based agent. For doing this I will develop a Genetic Algorithm, focused on finding the best set of rules and later on analyze the actual performance of the agent compare with other sample agents.

2 Hanabi

As mentioned in the Introduction, Hanabi is a cooperative game. Best described as the cooperative version of solitaire, in which each player holds a hand of four cards (five if playing with less than 4 players).

Each card has a rank, ranging from 1 to 5, and a color which is either red, blue, green, white or yellow. There are a total of 50 cards, 10 of each color: three 1s, two 2s, 3s and 4s, and just one 5. The goal of the game is to play form one stack of each color, consisting of consecutively ranked cards.

What makes Hanabi special is that contrary to other card games, the players can see the other players card, but not their own. What let the players know which card is their own is by the help with other players, who must give them **hints** about their cards

2.1 Hint

As the action of their turn a player can decide to give a hint to another player about the cards of their hand. This hint can be which card is of *x* color, or which card is of *y* rank. *Note* this hint should be exhaustive to all the cards, being that if you want to say that one card is blue and there is another blue card in the player hand, you must hint the player that those 2 cards are blue Eg. "Your second and third cards are a **2**," "Your fourth card is **blue**."

Nevertheless, giving a hint will not only consume the player's turn but it's also based on a limited resource. Only able to have more possibility of giving more hints by either **completing a stack** or **discarding a card**

2.2 Discard

As another possible action that a player can make in their turn, is discarding a card of their hand. When discarding a card, the player must leave the card face up, along side any other discarded and unsuccessfully played cards, visible to all the other players. After the discard the whole team gain the possibility of giving another hint and the player who discarded must draw a new card.

2.3 Playing a card

The final possible action that a player can make is attempting to play a card. Playing a card is successful if the rank of the card is the next one in the sequence of the stack of the color of the card. Eg. A player attempts to play a 2-Blue and the top of the blue stack is 1, in this situation, the play will be successful.

If the play is successful, the card is place on top of the corresponding stack. Otherwise the team receives a **strike** and the card will be discarded without gaining a hint, leading to the player who played the card to draw a new one.

When the players manage to complete a full stack, they receive an extra hint.

2.4 Game over

The game ends by either of this conditions:

1. The players have successfully complete all the stacks
2. The players have received three strikes by playing cards unsuccessfully
3. After a player draws the last card from the deck and everyone has already played one final turn

At the end of the game, the score is the amount of cards in the stacks, going to a maximum of 25 points. Nevertheless if the game was ended due to the receiving three strikes, the final score always is 0 points.

3 Background

3.1 Agent

An *Agent* is an AI controller which takes actions based on rules or by learning from previous data, this two divides an agent into a Rule-based approach or a Learning Agent. Nevertheless both types of agents could take in consideration the *state* of the *environment* to take in consideration their following action. Eg. In chess take in consideration the position of the current pieces in the board.

3.2 Genetic Algorithm

Genetic Algorithm's main purpose is to replicate the phenomenon of *natural selection*, this means that using a population of random individuals the characteristics of those best fitted will endure through the following generations. This happens because the most fitted are able to *mate* more and thus are able to pass some or must of their characteristics to their *offsprings* [3].

In Genetic Algorithm normally the characteristics are represented as *chromosomes*, which are normally represented as a list of numbers of values representing something in the environment the *individuals* will be competing on.

A Genetic Algorithm is formed of different steps/functions which it is based on

- Mutation
- Fitness function
- Crossover
- Selection

Mutation A mutation is a change of the original chromosomes that the individual has. Eg. Adding more values to the chromosome or changing one from its values to another, etc.

Fitness function The fitness function takes as an argument the individual's chromosome and its resulting value will be how fit or suited the individual is. The objective of the GA is normally to improve the overall fitness of the population.

Crossover Is the process in the mating of individuals in which their genetic material will combine. This could be either by techniques like k-point crossover in which the chromosomes of both parents will be divided in k+1 random sections and those sections will be joined. Some examples of these are shown in fig. 1 and fig. 2, being One Point Crossover and Two Point Crossover examples respectively

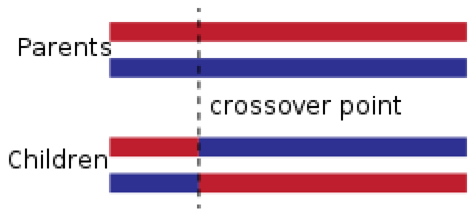

Fig. 1: One Point Crossover Example

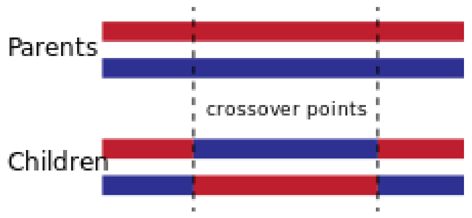

Fig. 2: Two Point Crossover Example

Selection For the creation of the next population for the next generation, a portion of the current one is selected to be able to breed the new generation. Is a common technique that the technique used to select who will be able to mate, is more probable those individuals who are fitter after using the *fitness function* to calculate it.

A popular form of selection and the one used, is the one called *tournament selection*. This type of selection gets a random sample of the population used of N size (popularly called tournament size), from this sample the individual who will be selected will be the fittest.

4 Techniques implemented

As mentioned before the main approach used is the use of Genetic Algorithm, with this GA the objective is finding the best set of rules for a Rule-Based Agent.

4.1 Constants

Some constants that will be used for the algorithm are the following:

- Population size: 60
- Generations limit: 40
- Games to play: 100
- Number of players: 4
- Tournament size: 5

4.2 Chromosome

The chromosomes used for the individuals will represent in this case the rules that the player will follow. These are all the rules that the Game Engine provides us, summing a total of **52 rules**. This rule's values are then represented with numbers ranging from 0 to 51 inclusive. Both this values were also arranged for simplicity to have the values of all the discard rules continuos to one another, ranging from 0 to 19 inclusive, these were also done for the tell and play rules.

Nevertheless we still must mention that a chromosome may be *invalid*, this means that with the set of rules that we have we're not able to make a move in certain scenarios.

4.3 Fitness Function

To calculate the fitness of a chromosome, we will create a 4 Rule-Based Agent that will try to execute those rules in the chromosome in the order that they are. With these agents we will play the game the N amount of times previously mentioned. If the runner notifies that in any game there were a disqualification being that the Agents got stuck, we will classify this chromosome as invalid and thus have a fitness of 0. Otherwise, the fitness will be the mean of the score of all the games.

4.4 Crossover function

There are two possible crossover functions that could be executed: A one point crossover or a custom crossover which is the concatenation of both chromosomes. The decision of using either of the crossover is by the following probabilities:

- One point crossover - 66.66%
- Concatenation crossover - 33.33%

After performing either of the crossover, there exists the possibility of duplicated rule values in the chromosome, this doesn't have a positive or negative impact in the fitness and just increases complexity of the chromosome. Thus the following step will be removing duplicate values, preserving only the first occurrence of the value.

As a final step there's a 67% chance that the offspring has a mutation. If this happens the mutation function will be triggered.

4.5 Mutation function

There exists 4 possible mutations: **Add** a new rule, **remove** a rule, **shuffle** the chromosome or **change** the rule for another. Nevertheless the mutation function has three possible branches, which affects which of the prior mutations may happen, and the probability of their execution, all of this depending of the length of the chromosome.

1. A length equals to 3 or less values
2. A length equals to the amount of rules
3. A length that doesn't fulfill neither of the above conditions

Table 1: Probabilities for each of these conditions

Condition	Add	Remove	Change	Shuffle
length <= 3	60%	0%	30%	10%
length == rules	0%	50%	0%	50%
3 < length < rules	30%	30%	30%	10%

4.6 All-time best population

For creating a new generation, instead of using the current population to create it, it was used instead an all-time best population. This population was added the top 10 individuals of each generation, if the size of this population exceeded the population size, then it would just kept the best currently in the population. This population was mainly used for the mating and for selecting which individual was the best of all-time.

4.7 Algorithm

Taking all of the previously commented in consideration what the algorithm does is the following:

1. Create an initial population A random initial population will be created, this will be done by creating random individuals which original chromosome values follow an specific pattern: All initial chromosomes should have 2 random rules of each of the rule types (discard, tell and play), thus all initial random chromosomes have a length of 6 rules. With this we avoid the possibility of repeated rules in the initial chromosome and probably avoid a lot of initial invalid chromosomes (though there will be some of them).

2. Run Generation With the current population we will run the generation, and calculate their fitnesses with the fitness function.

3. Store the best individuals of the generation We will keep the best 10 individuals of the current generation in a List which size will not be more than the population size. Due to the previous aspect, when the list surpass the limit, the best fitnesses will be kept while the ones with the lowest fitness will be removed from the list.

4. Create new generation Using the list of the best individuals of all time, the best 4 individuals of all times. Each of these individuals will have 15 breeding partners that will be selected by using a *tournament selection* with the mentioned above tournament size. With these selected individuals the best individuals will create a new individual with the *crossover* function.

5. Repeat The algorithm will repeat itself starting again from step 2 (running the generation) until the current generation is the generation limit.

5 Experimental Study

5.1 Amount of games

One of the initial variables that I changed from their initial values was the amount of games to get the fitness of the individual. This happened because initially I saw how there were some fitnesses (mean of game's scores) with 19+, but when testing again that individual with the same amount of games it gave me sometimes a completely different number. Thus, concluded that the individual may be very lucky sometimes, but their reality is very different.

To avoid this I decided to increase the amount of games from its initial value of 10 games to 100 games, Because even if this means a higher time complexity, I will have results more statistically valid and by conclusion will indeed help to find the best possible individual. Sadly this show that the algorithm was not as perfect as I thought as it changed from having a higher density of values between 17-19 to instead being between 14-16.

5.2 The importance of the order

Another change led to being more familiar to how the RuleBased agent that the engine gives, was how important it's the order of the values of the chromosome. Being possible to have a higher or lower fitness, just due to the order of the rules to be executed. Due to this the decision was made of adding the mutation for shuffling the values of the chromosome. Improving by a little bit the performance of the algorithm.

5.3 Why it takes longer each generation to be completed?

One thing that I noticed at the start is that each generation took more time to be completed compared to the last one. Initially I thought it was because the chromosomes were getting much better, thus the games lasted even longer. But after careful consideration and seeing how was the mean and std of the fitnesses of all the individuals I noticed that that was not the case.

The reality was that when doing the crossovers initially I didn't thought about repeated values or how long would the chromosome be, this led to *bug* when doing the mutation for adding more values. This happens because the mutation will recommend a random value that the individual doesn't have. This lead to the problem like having an already completed chromosome with all the rules and that the code will try to look for a random one that the chromosome will not have, ending in a deadlock. This problem would be amplified considering that there may be a lot of unnecessary values due to duplicated values.

After taking this in consideration the algorithm was changed for avoiding these problems, and as a test checked if once again it got itself in a deadlock after some generations, this didn't happened. Allowing me to then test how higher amount of generations would help my algorithm.

5.4 More generations

The initial limit of generations were 20 generations, a number chosen to avoid the problem formerly explained. But after solving it, it was decided that we could check how it would affect an increase of generations for the algorithm, in this case it was tested with 40 generations. The results were positive, as must of the time there would be a new all-time best or at least the standard deviation between the values would decreased. It was considered to double the amount of generations again, for checking the possible results, but 40 generations were already taking around 1 to 4 minutes, hence preferred the 40 generations to avoid long running times.

5.5 The use of an all-time high population

Initially the population used to create the new generation was the current generation, but as an experiment it was decided to see how the performance would change, using instead an all-time high population in which it would be added each generation the best 10 individuals containing max 60 of the best individuals of all-time. Using this as the population to create the new generation the results were favorably, increasing the possible max individual fitness.

5.6 Best initial random population

After analyzing the best chromosomes that have been created after running the algorithm 60+ times, I noticed that all of these have been of length of at least 6 rules. After this, I decided then to modify the initial creation of population which was only to create one random rule of each rule type. After making the change, even though there were no a lot of changes about the max fitness that the algorithm reaches, but the standard deviation that the results have are less. These means that the range of the best results have less outliers (previously there were a lot of outliers of *best individuals* with fitness less than 10), with this new rule for the creation of a random population, there were none of these outliers.

5.7 Once again it was shown that the order is important

One final change done was when discovering that the algorithm used to remove the duplicates of the chromosome made that the resulting chromosome changed the order of the values. This made that the GA performed inconsistently and the results were worse than it should be. This changed confirmed even more for me that the order indeed is very important for the better performance of the individuals.

6 Analysis

6.1 GA's performance

To analyze how the the GA performs it will be analyzed how in average the fitnesses progress through each generation and how it was the distribution of the best fitnesses after looping the algorithm several times.

Fitnesses progression To analyze how the fitnesses progress through each generation, we will loop the algorithm 60 times. In each loop of the algorithm it will be gathered the mean of the individuals fitnesses. For the formal analysis we will select a random 6 entries of these 60 loops, plotting the means as the y value and the generation in which it was that mean as the x value.

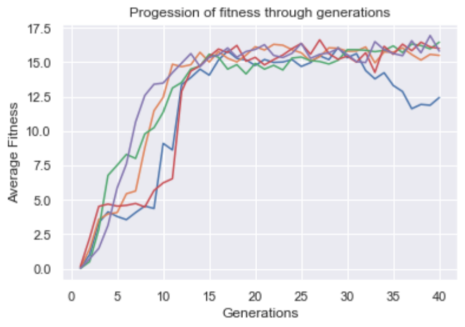

Fig. 3: Average fitness progression through each generation

After analyzing fig. 3, there are some insights that we can have of the algorithm:

- The average fitness of the population through the generations is overall in an upward trend through all the generations.
- There's a case where the avg fitness of a loop of 40 generations started going downwards starting around generation 30. This can be interpreted as that maybe the limit of generations should be reduced from 40 to 30.
- In addition to this it seems like the overall avg of the population is being kept stable starting around the 15th generation. Thus, in addition to the previous statement, it can be conclude that the generation limit could be between 15-30.
- Also, it seems like the avg fitness at the end of generation 40 is more than 12, which is a decent fitness overall, thus it seems like the GA is very effective.

Distribution of best fitnesses The fitnesses of the best individuals will be analyzed by plotting the density distribution of the fitnesses of all best individuals gather after looping 60 times the algorithm. It must be mentioned that instead of using the normal 100 games to know the fitness of the individual, in this case it was used 1000 games, after the gathering of the best individuals of those loops.

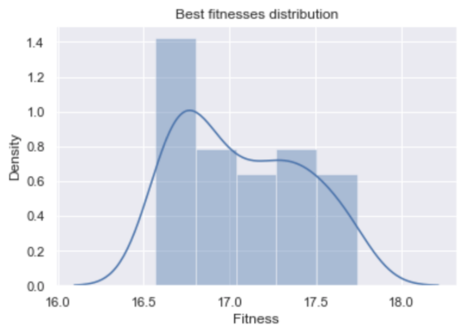

Fig. 4: Density distribution of the fitness of the best individuals

From fig. 4 we can conclude that most of the fitnesses are concentrated around the 16.5 value. Which is a very good average to have overall, thus we can be satisfied with this result.

6.2 Why does the best individual is the fittest?

The best fitness found has the following chromosome

```
[45,24,15,8,39,16,27,3,29,4,14,21,25,12,35,18,46]
```

After analyzing other best fitnesses maybe why this chromosome works is due to the following

- **Its length is overall shorter than average**. This was a surprised, as it seems like longer chromosomes are the ones that have the more chance of being better. Nevertheless the resulting best chromosome its length is 6 rules less than average. This could be seen like the performance is mainly due to the set of rules it has and the order of it.
- **The first 4 rules are very conservative**. Considering these rules will be the main rules will be triggered their values are very interesting. The first rule will be a conservative play rule, which makes the agent play if it has 70% chance that the card is safe. Later on comes the important rule of telling someone else if they have a useful card, allowing the game to continue onwards and let the player have overall more knowledge of its cards. And finally it comes the discard rules, which in this case are conservative plays of mainly discarding probably useless card or discarding the oldest card with no info.

- **There's a main concentration in discard rules**. I think this is something important, as discard rules normally are more safe and provide extra value to the play, compared to doing play games or an incorrect tell rule. And with a bigger concentration of these type of rules, it means that the player is mainly focused on discarding so that it could be use later on for hints.

6.3 Agent's performance

To analyze how's the performance of the Agent, it will be compared to other sample agents provided by the framework.

Playing against itself The first test would be running each agent (the sample ones and the GA's) 1000 times against themselves. The results are the ones show in Table 2

Table 2: Statistics of each player after playing 1000 times with itself

Agent	Average	Standard Deviation	Min	Max
GA Agent	17.827	1.7453	2	22
Piers	17.112	1.54	12	21
Iggi2	16.064	1.2293	11	20
Flawed	1.623	1.8853	0	13
Random	1.287	1.3184	0	8
MCTS	3.526	3.1024	0	18

Based on these results the following discoveries are the most outstanding:

- The performance of the GA's agent outperforms the other sample agent's behaviors. Nevertheless, this may just happen if all the players in the game, play with the same behavior, aka. same agent against itself.
- The use of the agent doesn't always lead to a good game, this can be shown with the range shown of 20 values of the agent. But this score of 2 seems more like an outlier, based on the standard deviation that the agent had.

Playing against other players The next test is about how the agent performs when playing against the other agents. Same as before it will be 1000 matches, against other random 3 unique agents that exclude the agent being tested itself. The results are the following:

Table 3: Statistics of each player after playing 1000 times with other random players

Agent	Average	Standard Deviation	Min	Max
GA Agent	5.608	3.6929	0	21
Piers	5.679	3.8095	0	21
Iggi2	5.644	3.8941	0	22

Agent	Average	Standard Deviation	Min	Max
Flawed	4.819	3.2635	0	19
Random	4.84	3.1474	0	20
MCTS	5.105	3.3878	0	20

These are the discoveries found based on these results:

– The GA's agent performs decently against other players compared to the other agents, ranking 3 out of 6 of all the agents. This means that there's still an area of opportunity for the agent.

7 Overall Conclusions

It was learned by analyzing the best individual that an overall conservative play-style may lead to higher performance, in addition that having more amount of rules in consideration also improves the average score than a rule-based agent can have.

The algorithm can be considered successful, as its performance led to very good statistics compared to other sample agents. Nevertheless, the chromosome considered as the best one could be considered an outlier or anomaly between all the best ones that the algorithm has brought. This can also be demonstrated with the analysis of the distribution of the best fitnesses of each generation shown in fig. 4, as most of the best individuals created by the GA had a fitnesses around 16.5.

Thus a further work that can be done may be trying to reduce the minimum fitness that the GA can have. Maybe this could be done by instead of doing an algorithm more focused on finding the best amount of rules and which rules specifically, created random orders of all the rules, thus mainly focusing only on the order of this rules.

In addition the current GA could be improved by changing the fitness function, to maybe also consider the average score when playing against other agents. This could help improve the average score when playing 1000 games with other players that are not the agent

References

1. N. Bard, J. N. Foerster, S. Chandar, N. Burch, M. Lanctot, H. F. Song, E. Parisotto, V. Dumoulin, S. Moitra, E. Hughes, I. Dunning, S. Mourad, H. Larochelle, M. G. Bellemare, and M. Bowling, "The hanabi challenge: A new frontier for ai research." *Artificial Intelligence*, vol. 280, 2019. [Online]. Available: http://0-search.ebscohost.com.biblioteca-ils.tec.mx/login.aspx?direct=true&db=edselp&AN=S0004370219300116&lang=es&site=eds-live&scope=site
2. J. Goodman, "Re-determinizing information set monte carlo tree search in hanabi," *CoRR*, vol. abs/1902.06075, 2019. [Online]. Available: http://arxiv.org/abs/1902.06075
3. W. Roetzel, D. Chen, and X. Luo, "Design and operation of heat exchangers and their networks," 2020, [Online; Accessed 10-11-2020].