Can Stacked Genetic Algorithms Following Stacked Rules Stack Cards Following Stacking Rules?

Rachel Lowe

This paper discusses an attempt to try evolving a variety of agents to play the game of *Hanabi*, with an end goal of comparing the performance of an agent which follows rules in different orders of priority in different situations to the performance of an agent which sticks to one fixed prioritized ruleset. Unfortunately, due to time constraints, the results are currently unknown, as, whilst a means of creating, evolving, and training these individuals, with somewhat coherent representations of their internal states being visible, has been implemented, not enough time was left for the process of using the aforementioned implementation to answer this question.

Introduction:

Hanabi is a game involving cooperating with other people to stack cards on top of each other, in a particular order, with the catch being that you can see everyone's cards but your own, and communication is heavily restricted [1]. This has since made it a source of interest for some AI research, with approaches such as genetic programming [2], reinforcement learning [3], and raw mathematics [4] being used to try to create an agent capable of playing *Hanabi*, and playing it well.

One gap in existing research regarding the application of genetic programming to *Hanabi* regards attempting to, not just train a single ordered list of 'rules' for an agent to apply, but instead train an agent with multiple prioritized lists of 'rules', context-dependent, and swapping between different lists of priorities based on certain conditions regarding the current gamestate. Existing literature by and for players of *Hanabi* do mention the merits of such an approach to *Hanabi* [5], due to the inherent coordination and implicit communication such an approach provides. Therefore, the intent here was to attempt creating these stacked genetic algorithms (or, rather, a genetic algorithm aiming to produce individuals with multiple optimized lists of rules) with stacked rules (due to the rules being ordered by priority, and, as there are multiple lists of rules, they're technically stacked that way) and getting them to stack cards following the rules by which the cards are stacked (or, in English, getting them to play *Hanabi*).

This variety of task, involving cooperation with limited information and limited communication, has applications outside of simply playing this game, for example, in the domain of getting multiple autonomous AI agents (perhaps a legion of overengineered roombas, perhaps a fleet of self-driving cars, or maybe even a spaceship of robots) which may not be able to easily communicate with each other to effectively coordinate with each other.

In this paper, I shall discuss my attempt at performing this investigation, along with my fully implemented mechanism for performing this investigation (which takes advantage of the *Hanabi Learning Environment* framework [3]). Unfortunately, as of right now, this experiment is not yet finished (as, whilst the code for it has been implemented, the code hasn't finished running), so the results (or, rather, the lack of) are very much inconclusive.

Hanabi

Hanabi is an imperfect information cooperative multiplayer board game, designed by Antoine Bauza, and first released in 2010 [1]. This game is for 2-5 players (the 4-player variant shall be the focus of this paper): each player has a hand of 4 or 5 cards (from a deck of 50), each card has a colour and a number, and the players, who can all see everyone else's cards, must work together to, for each colour, play cards 1-5 for that colour in order, without playing a card which has already been played, or a card which isn't playable yet. The catch is that players can't see their own hand, but they may communicate with other players: a player wishing to communicate must spend an 'information token' during their turn (tokens may only be gained when any player discards a card during their turn, and discarded cards cannot be salvaged), and this player may only tell the other player about *all* of their cards of a certain colour or number [6]. Critical reception to this game has, overall, been positive, and the unique, cooperative, imperfect information nature of the game has attracted some attention from AI researchers, due to the unique challenges posed by the need for agents to inform each other about their own hands, handle the inherent uncertainty from the shuffled deck of cards rendering brute-forcing an optimal approach practically impossible, and working together for a common goal [3].

There are two main approaches for the task of getting a computer to play *Hanabi* well; 'Self-play learning' (where one agent tries to learn how to play *Hanabi* by playing with (multiple copies of) itself) and 'Ad-hoc Teams' (where agents are trained with the end goal of making them perform well in a team, in which the other team members are completely different agents, which are also trying to perform well in a team of complete strangers) [3]. For purposes of this paper, the 'self-play learning' approach shall be used, due to reasons not entirely unrelated to the assignment brief, and due to it being simpler to set up and run, at least for the time being.

Of course, this attention from the AI field did not appear out of thin air; Prior research has been performed on *Hanabi*. A 'solution' was proposed for the two-player variation of the game in 2015 [4], and approaches to it using Genetic Programming have been investigated before [2], but the intent here is to try investigating that in a bit more depth.

Background:

As mentioned earlier, this agent is a 'self-play' agent, intended to play 4-player *Hanabi* (using a 5-suit deck and hand size of 4), cooperating with three other instances of itself. Whilst the *Hanabi* environment was designed to be tackled via reinforcement learning (presumably using the 'vectorized' item in the observation dictionary as the input for the agent, perhaps being a neural network-based agent), I have chosen to use a rules-based genetic programming approach, attempting to train an optimal configuration of rules for an agent to apply to the game. This is because, unlike the black-box approach of neural networks and other similar forms of machine learning which usually boil down to an incredibly abstracted view of some inexplicable mathematical wizardry, the outputs of this approach are, at very least, not entirely incomprehensible, which is a benefit for this use case.

The task here is to create an agent capable of playing the game *Hanabi* and playing it well. Whilst other people have attempted to try solving the problem of 'what is the best set and order of rules for an agent to apply to the game of *Hanabi* in order to solve it?' [2], I intend to try to take this concept further, by trying to compare different 'shapes' (for lack of a better term) for these rulesets. Upon looking at some strategies for playing *Hanabi*, written by players of *Hanabi* for players of *Hanabi*,

it appears that there is some merit in players adjusting their strategies based on the current state of the game. For example, within *The Hyphenated Conventions*, one such strategy guide, one key point covered in the beginner strategy section is how there is a distinct 'early game' phase (the period until the first player discards a card), in which the priority should be on giving every player who is in a position where they would benefit from a clue the necessary clues they need, and only discarding (and initiating the 'mid-game') either after exhausting all info tokens, or the current player not seeing any 'legal' (under these player-established conventions) hints they can give [5]. This approach to the game is worthy of further study, hence the attempt I made at investigating it further.

Techniques Implemented:

The agents I attempted to create were constructed via a somewhat indirect process of genetic programming (through the medium of a real-valued genetic algorithm). Whenever it is the agent's turn to play, they are provided with an analysis of the current observed gamestate, and the chromosome controls the behaviour of the agent by providing an ordered list of 'rules' which the agent will attempt to apply to the current situation. The agent also provides a summary of the analysis of the observation to the trained chromosome (such as how many information tokens are left (and whether discarding/telling are legal moves), how many lives are left, the deck size, state of the fireworks, whether any cards have been discarded, and the player index of the player), which can allow the chromosome to provide one of multiple rules lists to the agent, given the current situation.

The rules themselves are values of an enum (which the chromosome and the agent are able to read). The chromosome itself stores these ordered lists of rules as ndarrays of floats between 0 and 1, each value being associated with a 'priority' for a certain rule (with the chromosome class itself having a particular fixed tuple of 'rules', and these are then ordered in descending priority using numpy's argsort method (on a version of the priorities ndarray where everything is negative instead) to find out what order these indices must be returned in.

As the chromosomes effectively used a real-value genetic algorithm structure, most of the genetic operators used for the evolution process were relatively simple. Crossover is implemented in the form of N-dimensional lerp between two nominated 'parents', and mutation is implemented in the form of minor N-dimensional displacement; however, the values are kept in range via modulo, effectively causing invalid values to wrap around instead of crudely clipping them at the limits (which would increase the risk of an individual getting stuck in a local minimum by restricting the ability of the value to change as it gets closer to the bounds). Additionally, when crossover is chosen, there is a 25% chance that the crossover will effectively be forced to go the other way around (I'm not sure if there is a formal mathematical term for this, but by increasing the values of the copy of one parent by 0.5 before performing the lerp (and performing the wraparound after the lerp), it effectively causes the lerp to happen around the reverse side of the n-dimensional hypertorus search space), once again with the intent to avoid getting stuck in local minimums when two parents are not that different to each other. I also chose to try implementing a 'shuffle' operator; producing a new child which has the exact same values as the parent, but a random subset (of a random size) of the values from the parent are shuffled. I opted to implement that operator due to the nature of the chromosomes trying to find an optimal ordering; therefore, directly manipulating the ordering made sense (although, in hindsight, this operator may be a bit impractical for producing beneficial evolutions, as it's incredibly likely to produce children which are not entirely

unlike their parent, for better or more probably for worse). When chromosomes with more than one priority list have these operators applied to them, they are applied per-list within the individual (although, for the 4-player self-play chromosome, the shuffle operator has a 50% chance to reassign the lists to the players as well).

The genetic algorithm framework used to evolve these individuals is one of my own design, loosely based on a GA framework I initially created (from scratch) for use in the CE310 Evolutionary Computation and Genetic Programming module I took at this university last year. I have attempted to enhance it before applying it to this task, most notably providing proper support for a variety of chromosome types (instead of having only a single static fitness function and decode function belonging to the GA class which needed to be overwritten every time a new chromosome type was being tested, therefore rendering all prior existing chromosomes unevaluatable due to them using a reference to that method of the GA class), and other performance improvements such as replacing most of the python lists/tuples with ndarrays, and implementing some multithreading for the fitness evaluations (considering time constraints), and being able to produce graphs of agent performance over the course of evolution, and over the course of a single game of *Hanabi* (assuming, of course, that the population is able to conclude the evolution process).

The rules are a lot less pleasant to discuss.

The rules used by the agent used in the final version are ultimately based on particularly naïve approaches to playing *Hanabi*, notably not implementing several of the 'basic' techniques for playing *Hanabi*, such as the 'good touch principle' (the principle that, when giving a clue to tell a player about a certain card, one should try to only provide information about that single card [5]) and 'finesse'/'prompt' moves[1]. The reasoning for this is somewhat unfortunate; the final version submitted was a relatively last-minute attempt at restarting from scratch. Included in the files for the final submission is a copy of the Jupyter notebook containing the initial attempt at this, which had rules which actively attempted to follow conventions such as the 'good touch' principle. Ultimately, that version performed significantly worse than expected (especially considering that the relatively naïve approaches I used for the genetic algorithm agent I made for the 7th lab session for this module were able to get an average score ~10 times higher than the attempts at implementing these advanced tactics), which may have been due to a few mistakes in implementing some of the logic for those rules. However, as the result of ~50 hours of non-stop work on that was a score that paled in comparison to the average scores achieved by the agent that was provided as an example of an incredibly barebones approach to *Hanabi* (even with particularly desperate attempts at trying whatever I could to fix the potential causes of the problems), I ultimately opted to retry from scratch instead of trying to salvage that.

Despite this, some of the functionality for the agents does try to implement some aspects of these strategies, notably by getting the agents to only consider discarding their oldest card which they know nothing about, and several rules explicitly targeted towards informing a player, whose oldest unknown card is a card which is worth saving (due to it either being the only viewable instance of that particular not-useless card, or due to it being *the* only remaining non-discarded instance of that not-useless card), some information about that card, so it doesn't get discarded. Furthermore, a rule has been implemented which is explicitly intended to only tell a player about a useless card if the player only knows half of the info about that card (info which could mislead them into thinking that

[1] Explained rather well in *The Hyphenated Conventions* [5]

the card is useful when it isn't, in turn prompting them to potentially misplay it), intended to avoid any problems from accidentally saying too much (although this probably does not adequately compensate for the lack of a functional adherence to the 'good touch' principle).

Experimental Study:

Describe the experimental study performed to obtain the final submitted agent (e.g., parameter tuning, effect on the results of adding different functionality to the agent, etc.).

Analysis:

Analysis of the development process, (what things worked, what didn't work? Did something surprise you?)

Everything that could go wrong did go wrong.

I was late starting (due to taking too long on other assignments), the entire ~70+ hour development cycle on this assignment was condensed into one week, and there was a severe lack of sleep, which lead to many utterly terrible decisions being made throughout.

The agents I have produced are much worse than the agent I produced for lab 7. I have failed to achieve any of the aims I sought to achieve. I should have tried to get the other assignments finished sooner, I should not have wasted time overengineering the genetic algorithm runner and instead spent more time getting something quick and dirty up and running (and get it to finish running), and I probably should not have performed the utterly stupid decision of giving up on one implementation due to significantly worse than expected outcomes being reached, instead, I should have tried to fix the problem with that one, which may have wasted a lot less time, and I might have been able to start properly running it sooner, and getting this documentation written sooner. But no, I didn't, and that's all I can really say on the matter.

Overall conclusions:

Problems and difficulties found, main take-aways) and potential future work. Note that conclusions must not be a reflection of what did the assignment mean for you as a student, but a critical summary of the work and findings of the project.

Despite this investigation completely failing to achieve anything resembling the thing it was trying to achieve (with the only tangible outcome I can show being an agent incapable of the simple task of scoring 2 out of 25 on the agent tester on Moodle), I suppose I did at very least write a Jupyter notebook which, out-of-the-box, can allow anyone who wants to do so to run this experiment themselves, with minimal setup required (assuming they have the hanabi_learning_environment, scipy, numpy, matplotlib, and jupyter installed via pip and usable with python 3.8 or later).

However, as the experiment is unfinished, and the part of the experiment in the name of this experiment is yet to start (due to the slight problem of genetic algorithms taking time to run), I cannot, in good faith, make any conclusions, besides the redundant 'there isn't anything to conclude yet'

References

Bibliography

[1] A. Bauza, *Hanabi,* Montigny-lès-Cormeilles: Les 12 Singes, 2010.

[2] J. Walton-Rivers, P. R. Williams, R. Bartle, D. Perez-Liebana and S. M. Lucas, “Evaluating and Modelling Hanabi-Playing Agents,” in *Proceedings of the IEEE Conference on Evolutionary Computation*, 2017.

[3] 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, “[1902.00506v2] The Hanabi Challenge: A New Frontier for AI Research,” 6 December 2019. [Online]. Available: https://arxiv.org/abs/1902.00506v2. [Accessed 19 January 2022].

[4] H. Osawa, “Solving Hanabi: Estimating Hands by Opponent's Actions in Cooperative Game with Incomplete Information,” in *Papers from the 2015 AAAI Workshop*, Palo Alto, California, 2015.

[5] Zamiell and e. al, “Home | The Hyphenated Conventions,” 2018. [Online]. Available: https://hanabi.github.io/. [Accessed 19 January 2022].

[6] B. Formery et al, “Hanabi - Play Hanabi online with friends!,” hanabi.cards, 2020. [Online]. Available: https://hanabi.cards/. [Accessed 19 January 2022].

[7] BoardGameGeek, “Hanabi | Board Game | BoardGameGeek,” BoardGameGeek, [Online]. Available: https://www.boardgamegeek.com/boardgame/98778/hanabi. [Accessed 19 January 2022].

Anyway, here’s the performance of the bot on the moodle bot tester. Brace yourself for disappointment.

For Assignment 2, Your Hanabi agent scored an average of 1.1 points per game.

Part #	Outcome	
Class MyAgent is defined	Yes	✔
Agent Score (for Assignment 2)	1.1/25	✖

Partially correct

Marks for this submission: 1.10/25.00.