university of groningen / faculty of science and engineering

Reinforcement Learning and Evolutionary algorithms in the Stochastic Environment of Blackjack

Bachelor's Project Thesis

Thomas Vos, s3162443, t.j.vos@student.rug.nl,
Supervisor: Dr. M. Sabatelli

Abstract: The casino game blackjack requires player's actions to play. These actions have an optimal strategy which we learn with two different reinforcement learning (RL) algorithms (Q-learning and QV-learning) and 2 different evolutionary algorithms (genetic algorithm (GA) and particle swarm optimization (PSO)). The maximum win rate for blackjack is around 42,5%, and our best performing algorithm (QV-learning) achieved a 42,47% win rate with the best exploration policy. Q-learning and GA both achieved a 42,30% win rate with their best strategies. Whereas PSO performed badly only achieving 21.2% which indicates it is not suited for the problem.

1 Introduction

Blackjack is a casino card game with only a few elementary actions, yet it has an unfathomable amount of possibilities (Thorp, 1966). All the rules for blackjack are explained by Garvie (2017). In brief, the aim of the game is to achieve or get as close to 21 points as possible. The player starts the game with two cards, and decides whether to take another card (hit) or not (stand). If the player goes over 21, they lose. If the dealer has over 21, the player wins. If the player has a higher total than the dealer they also win. The other possible actions are split, which is allowed only when there are two of the same card in the hand of the player and double down, which is only allowed when the cards total 9,10 or 11. By splitting, the game effectively splits into 2 games at once and both games proceed as normal. With double down only one more card is added, but it doubles the bet for that game.

The game is dependant on luck and does not have a strategy with which one could always win, however there are ways to play that increase the odds of winning compared to a sub-optimal or random play. Since blackjack is a casino game, it is designed such that the house always has a slight advantage. According to other research (Summerville, 2019), (Thorp, 1966) the optimal strategy for blackjack has around 42.5% win rate, the maximum win rate achieved differs per source. This difference in results stems from the fact that blackjack is stochastic. This paper explores blackjack and attempts to find this optimal strategy by using different reinforcement learning (RL) and evolutionary algorithms. The main aim of this paper is comparing different RL and evolutionary algorithms to find which approach achieves a higher win-rate with their found optimal policy. Using Q-learning (Watkins, 1989) and QV-learning (Wiering, 2005) each with three different exploration policies (greedy, ϵ-greedy and softmax) (Sutton & Barto, 2018) as RL algorithms. A genetic algorithm (GA) (Man et al., 2001) with two different parent selection and crossover methods, and particle swarm optimization (PSO) (Wang et al., 2018) are used to explore the evolutionary algorithms. The main research question of this paper is the following, which approach (RL or GA) achieves a higher win rate/better optimal policy for blackjack? However within this search, comparisons will also be made within the two approaches to find the better algorithm within the approach. On top of that within each algorithm the different exploration policies will be compared for the RL algorithms and crossover/parent selection for the GA.

2 Related works

This paper is not the first to explore blackjack with either type of algorithm. A lot of research has been done to learn the optimal policy for blackjack, mainly using reinforcement learning or some adaption of it (Kakvi, 2009). Evolutionary algorithms have been used to learn blackjack as well (Summerville, 2019). The results obtained by them come really close to the optimal policy, however the run time is longer for the genetic algorithm than that of reinforcement learning implementations. These researches show that blackjack can be learned in either way but it is very difficult to obtain the perfect solution with either method.

The comparison between RL and evolutionary algorithms have been made countless of times before, which converges to the solution the fastest? which is more reliable to give the optimal solution? Tijjani & Ozkaya (2014) compared multiple different RL algorithms to two different evolutionary algorithm, after which they concluded that most RL algorithms converged faster to the solution than the evolutionary algorithms. Yanes Luis et al. (2021) concluded that evolutionary algorithms converged slower to the solution than RL methods for high dimensional problems. However both methods converged to similar results. Another research by Zhang & Zaïane (2017) compared multiple evolutionary strategies to deep RL algorithms on multiple different tasks, and could not conclude which type of algorithm was better overall. Both methods had their own tasks in which they performed better.

Reinforcement learning and evolutionary methods have also been combined to attempt to obtain better results than by using either method alone. Pathak & Kapila (2018) is such an example. Although it is good to realize that this option might be appropriate for the blackjack environment, it is not the focus of this paper.

3 Methods

In order to have algorithms learn blackjack, a python program* was made to simulate blackjack which agents can interact with. The game has been implemented from scratch. The implementation of blackjack follows the official rules played in casinos Garvie (2017), this ensures that the dealer has no choice in its actions and has to follow set rules for their actions. In order for an algorithm to choose an action, it needs a set of states with preferences for actions. These states are split into two main objects. The first one represents every possible value the hand of the player can have, which ranges from three to 21 as well as the first card of the dealer, since this is the only card of the dealer known to the player while playing. Each state has a value for all possible actions in that state. The second object is similar but represents only the states in which the player can split, which is every hand that has two of the same card. These states are separate since the other state object does not keep track of individual cards but only of the total value of all cards in the hand of the player. These states represent the policy. Therefore each algorithm will adapt the states according to their rules to improve the found policy to optimize the win rate.

As mentioned before comparisons will be made between four different algorithms. Two RL algorithms and two evolutionary algorithms, the implementations are explained in the following sections. The two different RL algorithms used are Q-learning and QV-learning.

3.1 Reinforcement learning

The Q-learning algorithm makes use of Q-values and the QV-learning algorithm makes use of both Q- and V-values. These values represent the goodness. High values of a V-state corresponds to a good fit for that state, this means that being in that state likely results in a good reward. Whereas Q-values represent not only the state but also the action taken when in that state. A high Q-value suggests that being in that state and taken the specified action likely results in a good reward.

For both RL algorithms, exploration policies are used to determine which action gets chosen at each time point. Three exploration policies have been implemented, greedy, ϵ-greedy and softmax. Greedy is the simplest exploration policy and simply takes the action with the highest Q-value in the state. ϵ-greedy is based on greedy but rather than always taking the best action it has a small chance to take a random action instead (see Formula 3.4).

*The code of the custom program can be found here: https://github.com/firevos/blackjack

This randomness ensures exploration of all actions when enough iterations are used and can prevent the algorithm getting stuck in sub-optimal actions.

$$\pi_\epsilon = \begin{cases} \max_{a \in \mathcal{A}}(Q(s_{t+1}, a_t)) & \text{with prob.} \quad 1-\epsilon \\ \text{random } a \in \mathcal{A} & \text{with prob.} \quad \epsilon \end{cases} \tag{3.1}$$

Here π_ϵ represents the policy (the action chosen) found with the ϵ-greedy exploration policy. ϵ is the exploration constant which is set to 0.05. Note that with an ϵ of 0, ϵ-greedy effectively becomes the same exploration policy as greedy. An alternative version of ϵ-greedy is used which converges ϵ to 0 over time. This ensure that the agent will only be encouraged to explore early on and choose the actions greedily later on. ϵ is updated each iteration as follows:

$$\epsilon = \epsilon - \frac{E}{\text{epochs}} \tag{3.2}$$

Each iteration ϵ gets reduced by a small margin. E denotes the starting value of ϵ and epochs are the total amount of epochs the agent runs for.

The third exploration policy, softmax is governed by the following formula:

$$\pi_{\text{softmax}} = \frac{e^{Q(s_t,a_t)}}{\sum_{k=0}^{N} e^{Q(s_t,a^k)}} \tag{3.3}$$

$\pi_{softmax}$ represents the policy found by the softmax exploration policy. N denotes the amount of different possible actions in state s_t. The softmax formula (see Formula 3.6) essentially turns all Q-values of a state into probabilities, giving higher probabilities to actions with higher Q-values and low probabilities to lower values. This exploration policy encourages exploration just as ϵ-greedy however not based on randomness but rather based on the goodness of each action.

Greedy has been chosen to act as a baseline compared to the other exploration policies, which have encouraged exploration. The encouraged exploration is important in the stochastic environment of blackjack since it is easy for an agent to get stuck in a sub-optimal action otherwise due to a random streak of luck. Both RL algorithms make use of the same exploration policies ensuring comparable results.

3.2 Q-learning

Q-learning (Watkins, 1989) is an off-policy (Suran, 2020) learning algorithm and works by keeping track of Q-states, these states are represented by the state objects explained in the previous section. After an action has been made and the game has advanced, the Q-states of the agent making the action is updated with the following formula:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_t + \gamma \cdot \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{3.4}$$

$Q(s_t, a_t)$ denotes the state-action pair at time point t. α denotes the learning rate, which represents how much the current choice influences the Q-value. α is set to 0.5. r_t represents the reward received for taking the action in state s at time t. γ denotes the discount factor which is set to 0.9. $\mathcal{A}$ denotes the set of all possible actions. $Q(s_{t+1}, a)$ represents the maximum expected future reward.

3.3 QV-learning

The other RL algorithm implemented is QV-learning (Wiering, 2005), which is an on-policy (Suran, 2020) algorithm. It works similar to Q-learning, but rather than keeping track of only the Q-values it also keeps track of the V-values and updates the Q-values based on the V-values. The V-values are updated as follows after every action:

$$V(s_t) = V(s_t) + \alpha[r_t + \gamma V(s_{t+1}) - V(s_t)] \tag{3.5}$$

Again, α denotes the learning rate and γ the discount factor. The Q-values are then updated using the same formula as for Q-learning (see Formula 3.4) with a slight altercation, using the V-values rather than the expected future reward:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_t + \gamma \cdot V(s_{t+1}) - Q(s_t, a_t)] \tag{3.6}$$

3.4 Genetic algorithm

The Genetic Algorithm (GA) is the first evolutionary algorithm implemented (Man et al., 2001). It works by initializing a set of agents with a random policy each. The state objects mentioned earlier are used by the GA to set the policy of an agent. This is done by setting the state-action pair to 1 if that action in that state belongs in the assigned policy and to 0 if it does not belong in the policy. Actions are then chosen greedily to ensure the agent always follows the set policy. The GA learns by firstly evaluating the set of agents based on the win rate returned by the agent after 100 000 epochs after which the next generation of agents is determined/created. There are two distinct methods within the creation of the next generation. The first is the parent selection and the second is the crossover ratio. There are two different parent selections implemented, tournament selection and ranked selection (Hassanat et al., 2019). There are also two implemented crossover ratios, firstly single point crossover (Man et al., 2001) and ranked based crossover. These different methods have been implemented to investigate whether a specific method would influence the results. The parent selection methods determines which agents generate offspring for the next generation and the crossover ratio determines how these offspring are created from the parents. Once the offspring has been created there is a small chance in each state for a mutation, this means that a random action will be set as the policy in that state rather than the inherited action. This chance is governed by the mutateChance variable which is set to 0.05 and reduces with the same rate as ϵ does for the ϵ-greedy exploration policy.

$$\text{mutateChance} = \text{mutateChance} - \frac{\text{mutateStart}}{\text{generations}} \tag{3.7}$$

Here mutateStart stands for the initial value of mutateChance and generations denotes the total amount of generations the GA will run for.

The stopping criteria for the GA is determined by a predefined amount of generations, which is set to 200. After the maximum generation has been reached the GA will terminate. The results are being tracked for each agent in each generation.

3.5 Particle swarm optimization

The Particle Swarm Optimization (PSO) is the second evolutionary algorithm implemented (Wang et al., 2018). As the name suggests, this algorithm works by exploring a search space with a swarm of particles which can communicate with each other. The dimensionality of the search space is determined by the amount of different states in the problem. Which in this case is rather large (>400) Since split states are counted separately from regular states and each combination of the player's hand and dealer's hand is counted as a separate state. The search space is in a continuous state, this means that the discontinuous state-action pairs had to be converted. This is done by creating boundaries between -0.5 and amount of possible actions - 0.5. So in the case where the actions *hit*, *stand* and *split* are viable, the boundaries are set to -0.5 to 2.5. Each agent (also named particle) gets a randomly initialized value within the boundaries for each state. These values are used for the PSO whereas they are converted back to a policy by rounding the value to the nearest integer and setting the value of that action to one in that state in order for the agents to follow a policy. Actions are chosen greedily again to ensure the set policy is followed. Each particle is evaluated with 100 000 epochs and returns a win rate. Each particle remembers its best position in the search space as well as the global best position found thus far. All particles get assigned a velocity vector. This velocity vector gets updated every iteration and is influenced by both its own best position and the global best position (see formula 3.8).

$$\begin{aligned} v_x &= v_x + 2 \cdot \text{rand} \cdot (p_{bestx} - x) + \\ &\quad 2 \cdot \text{rand} \cdot (g_{bestx} - x) \\ x &= x + v_x \end{aligned} \tag{3.8}$$

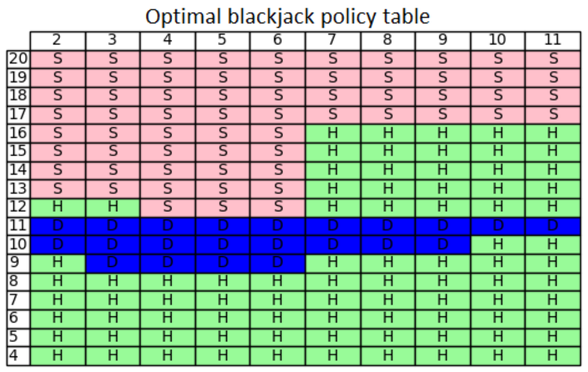
Formula 3.8 is applied to each dimension of the particle. The variable rand depicts a randomly generated value, generally between zero and one. p_{bestx} denotes the best known position in the x dimension of the particle it self, whereas g_{bestx} denotes thes best known global position in dimension x. After the velocity has been updated the position of the particle is updated by simply adding the current velocity to its current position. If the particle reached/passed the boundary in a certain dimension it is reflected in the opposite direction (reflecting wall) with 20% of the difference between the boundary and the surpassed space (dampening wall). This ensures no particle ever leaves the allowed search space. The goal of this algorithm is to find the optimal location in the search space which returns the highest win rate.

4 Results

The results are presented in two different ways. Each agent for each algorithm records its own win rate which is plotted to demonstrate learning curves as well as the optimal policy found by the algorithm. This optimal policy is represented in the form of a table showing which action is the best to take in each state the agent can find itself in. Both types of results are created for each type of algorithm.

A baseline was created for comparing results by running the Q-learning algorithm with ϵ-greedy exploration policy with an ϵ of one without decay. With ϵ at 1, the agent will always choose a random action and disregard any information presented to it. This creates a baseline for random plays. This simulation resulted in an average win rate of 31,1% (see Appendix A for raw results)

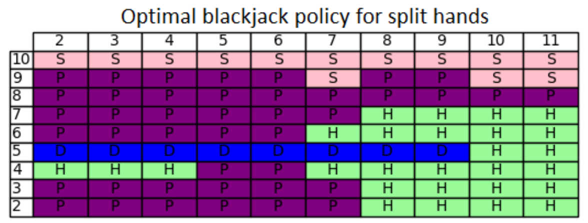
On the contrary, for a comparison for the policy tables the optimal policy table for regular hands is shown in Figure 4.1 and the optimal policy table for splittable hands is shown in Figure 4.2.



Optimal blackjack policy table

	2	3	4	5	6	7	8	9	10	11
20	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S
18	S	S	S	S	S	S	S	S	S	S
17	S	S	S	S	S	S	S	S	S	S
16	S	S	S	S	S	H	H	H	H	H
15	S	S	S	S	S	H	H	H	H	H
14	S	S	S	S	S	H	H	H	H	H
13	S	S	S	S	S	H	H	H	H	H
12	H	H	S	S	S	H	H	H	H	H
11	D	D	D	D	D	D	D	D	D	D
10	D	D	D	D	D	D	D	D	H	H
9	H	D	D	D	D	H	H	H	H	H
8	H	H	H	H	H	H	H	H	H	H
7	H	H	H	H	H	H	H	H	H	H
6	H	H	H	H	H	H	H	H	H	H
5	H	H	H	H	H	H	H	H	H	H
4	H	H	H	H	H	H	H	H	H	H

Figure 4.1: The optimal blackjack policy table.



Optimal blackjack policy for split hands

	2	3	4	5	6	7	8	9	10	11
10	S	S	S	S	S	S	S	S	S	S
9	P	P	P	P	P	S	P	P	S	S
8	P	P	P	P	P	P	P	P	P	P
7	P	P	P	P	P	P	H	H	H	H
6	P	P	P	P	P	H	H	H	H	H
5	D	D	D	D	D	D	D	D	H	H
4	H	H	H	P	P	H	H	H	H	H
3	P	P	P	P	P	P	H	H	H	H
2	P	P	P	P	P	P	H	H	H	H

Figure 4.2: The optimal blackjack policy table for hands in which you are allowed to split.

The policy tables are read as follows: The x-axis indicates the face up card of the dealer. The y-axis for non split states represent the total value of all cards in the hand of the dealer where as in the tables for the split states the value on the y-axis represents the value of one of the cards of the player. Since these hands only occur with two cards in the hand with both having the same value. The entries in the table are either H; denoting the action *hit*, S; representing the action *stand*, D; denoting *double down* and P; representing *split*. Any following policy table can be read in the same way.

4.1 Reinforcement Learning

The results for the RL algorithms are created by running 50 agents with the same parameters and same exploration policy to return an average win rate learning curve as well as an average optimal policy found by all of the 50 agents. 50 agents are used to eliminate as much randomness as possible. Running one agent can result in either a very high or very low win rate depending on chance, even after training an agent for 1 000 000 epochs.

The parameters used to create the results from the RL algorithms are displayed in Table 4.1. The rewards for each action are defined in Table 4.2

Figure 4.3 shows the results returned by the Q-learning algorithm for each different exploration policy implemented (greedy, ϵ-greedy and softmax). Each data point represents the average win rate of all 50 agents over the last 100 000 games played by those agents. This means that after an agent has played more than 100 000 games the first games are no longer counted towards its win rate. This was chosen due to the fact that the agent is learning and should not be punished for not achieving a low win rate early on. In the same way results for the QV-learning algorithm was created and is

Table 4.1: Parameters used by RL algorithms.

random seed	epochs	E	α	γ
1	1 000 000	0,05	0,4	0,9

Table 4.2: Rewards used by RL algorithms.

loss	win	hit without bust
-0,5	4	2

displayed in Figure 4.4.

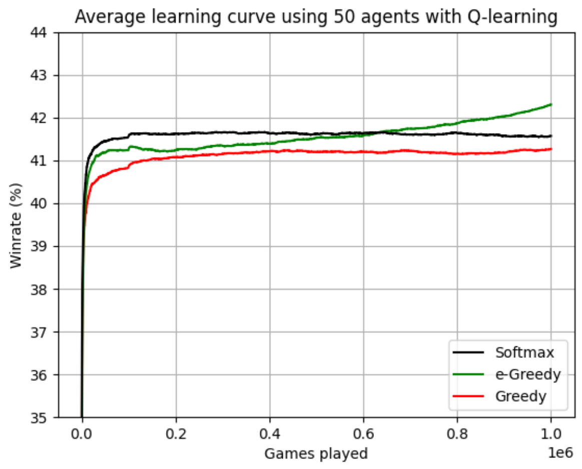


Figure 4.3: The average learning curve of 50 agents using Q-learning and 3 different exploration policies.

The learning curves (Figure 4.3 and Figure 4.4) show the ability of the algorithm to learn how to play blackjack, however not only the pure win rate is of interest. The policy with which the agent reaches its highest win rate is also important. Therefore this policy is also returned and visualized in a table. Each algorithm returns 1 optimal policy table which is the averaged table of all 50 agents at the end of the learning process. The average table is simply created by counting which action in each state belongs to the policy of the most agents. However every exploration policy of each algorithm creates its own policy table, but only the table of the best performing exploration policy is shown here, the others can be found in Appendix B. Figure 4.5 shows the optimal policy table for Q-learning with ϵ-greedy and Figure 4.6 shows the optimal policy table for QV-learning with ϵ-greedy. The split tables of these results can be found in appendix C.

4.2 Evolutionary Algorithms

The results for the evolutionary algorithms are created by running each algorithm for 200 generation (GA) or iterations (PSO) with 750 agents each. As mentioned before in section 3.4 and 3.5, each agent evaluates its policy by playing 100 000 games. To illustrate the results the win rate of each agent in each generation/iteration is being tracked in order

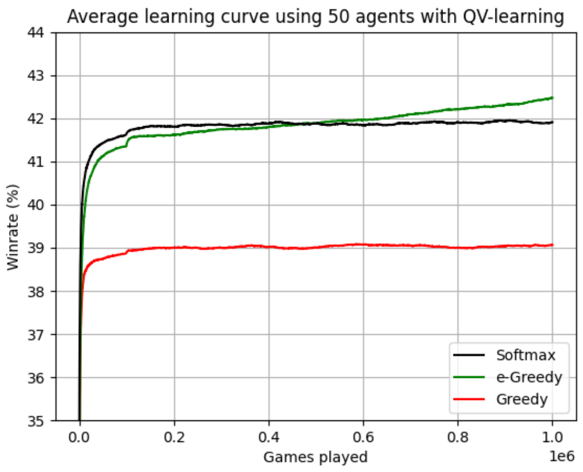
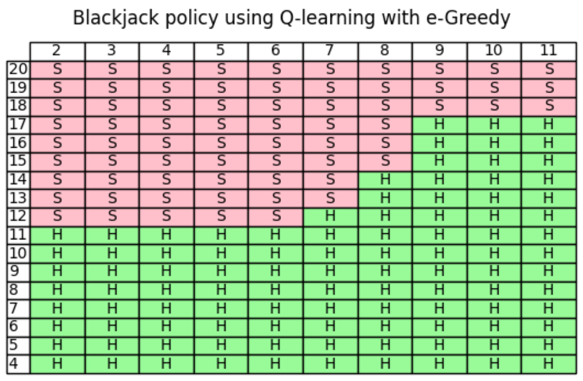


Figure 4.4: The average learning curve of 50 agents using QV-learning and 3 different exploration policies.

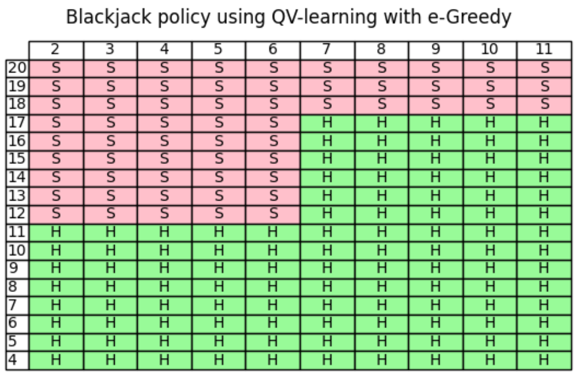

Blackjack policy using Q-learning with e-Greedy

	2	3	4	5	6	7	8	9	10	11
20	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S
18	S	S	S	S	S	S	S	S	S	S
17	S	S	S	S	S	S	S	H	H	H
16	S	S	S	S	S	S	S	H	H	H
15	S	S	S	S	S	S	S	H	H	H
14	S	S	S	S	S	S	H	H	H	H
13	S	S	S	S	S	S	H	H	H	H
12	S	S	S	S	S	H	H	H	H	H
11	H	H	H	H	H	H	H	H	H	H
10	H	H	H	H	H	H	H	H	H	H
9	H	H	H	H	H	H	H	H	H	H
8	H	H	H	H	H	H	H	H	H	H
7	H	H	H	H	H	H	H	H	H	H
6	H	H	H	H	H	H	H	H	H	H
5	H	H	H	H	H	H	H	H	H	H
4	H	H	H	H	H	H	H	H	H	H

Figure 4.5: The average policy of 50 agents using Q-learning with ϵ-greedy exploration policy for regular hands.

to graph the average win rate along with the min and max for each generation/iteration. Again these learning curves graphs (as seen in Figure 4.7 and Figure 4.8) show the learning ability of the evolutionary algorithms for the blackjack problem. The same random seed is used for these algorithms as well as a velocity constant for PSO of 0.5. The GA uses 2 different parent selection methods as well as two different crossover ratios as explained earlier, the results are combined in Figure 4.7.

For the evolutionary algorithm the policies of the agents are also being tracked. This is done by creating a policy table of the best agent in each generation for the GA and the best particle of each



Blackjack policy using QV-learning with e-Greedy

	2	3	4	5	6	7	8	9	10	11
20	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S
18	S	S	S	S	S	S	S	S	S	S
17	S	S	S	S	S	H	H	H	H	H
16	S	S	S	S	S	H	H	H	H	H
15	S	S	S	S	S	H	H	H	H	H
14	S	S	S	S	S	H	H	H	H	H
13	S	S	S	S	S	H	H	H	H	H
12	S	S	S	S	S	H	H	H	H	H
11	H	H	H	H	H	H	H	H	H	H
10	H	H	H	H	H	H	H	H	H	H
9	H	H	H	H	H	H	H	H	H	H
8	H	H	H	H	H	H	H	H	H	H
7	H	H	H	H	H	H	H	H	H	H
6	H	H	H	H	H	H	H	H	H	H
5	H	H	H	H	H	H	H	H	H	H
4	H	H	H	H	H	H	H	H	H	H

Figure 4.6: The average policy of 50 agents using QV-learning with ϵ-greedy exploration policy for regular hands.

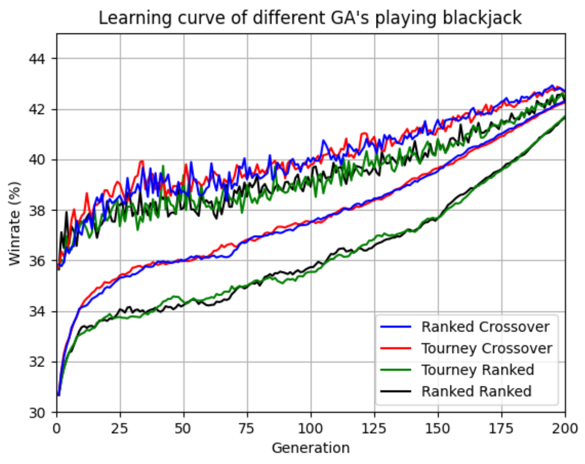


Figure 4.7: The learning curve of each combination of GA. The legend first indicates the parent selection method and secondly indicates the crossover ratio. The middle lines indicate the average and the top and bottom lines indicate the max and min respectively in that generation.

iteration for the PSO. Figure 4.9 shows the policies of the best agent after generation 200 for the GA (see appendix C for the split table). Only the policy tables of the GA with the best performance is shown here which is the GA with ranked parent selection and single point crossover ratio. see Appendix D for the remaining policy tables for the other combinations of the GA. Figure 4.10 shows the policy tables for the PSO algorithm.

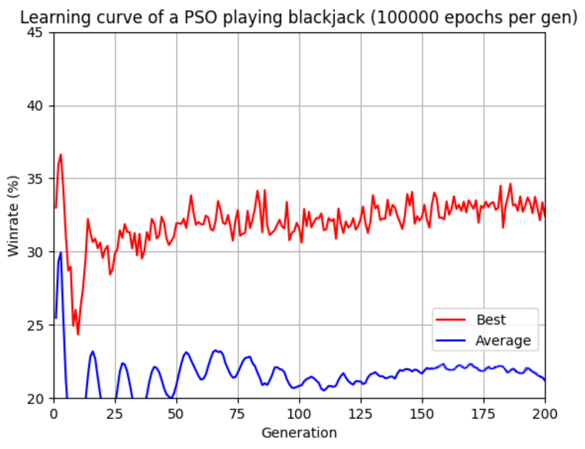
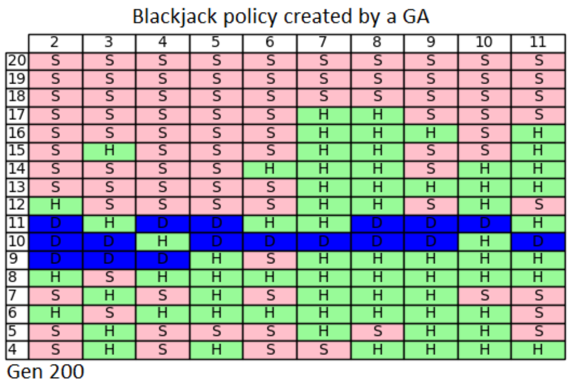


Figure 4.8: The learning curve of the PSO algorithm.

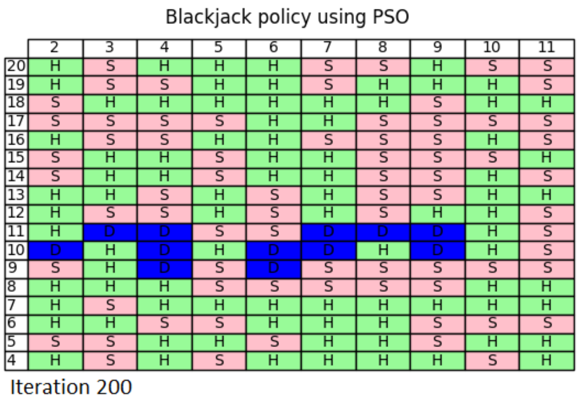


Blackjack policy created by a GA

	2	3	4	5	6	7	8	9	10	11
20	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S
18	S	S	S	S	S	S	S	S	S	S
17	S	S	S	S	S	H	H	S	S	S
16	S	S	S	S	S	H	H	H	S	H
15	S	H	S	S	S	H	H	S	S	H
14	S	S	S	S	H	H	H	S	H	H
13	S	S	S	S	S	H	H	H	H	H
12	H	S	S	S	S	H	H	S	H	S
11	D	H	D	D	H	H	D	D	D	H
10	D	D	H	D	D	D	D	D	H	D
9	D	D	D	H	S	H	H	H	H	H
8	H	S	H	H	H	H	H	H	H	H
7	S	H	S	H	S	H	H	H	S	S
6	H	S	H	H	H	H	H	H	H	S
5	S	H	S	S	S	H	S	H	H	S
4	S	H	S	H	S	S	H	H	H	H

Gen 200

Figure 4.9: Policy table for regular states created by a genetic algorithm with ranked parent selection and single point crossover.



Blackjack policy using PSO

	2	3	4	5	6	7	8	9	10	11
20	H	S	H	H	H	S	S	H	S	S
19	H	S	S	H	H	S	H	H	H	S
18	S	H	H	H	H	H	H	S	H	H
17	S	S	S	S	H	H	S	S	S	S
16	H	S	S	H	H	S	S	S	H	S
15	S	H	H	S	H	H	S	S	S	H
14	S	H	H	S	H	H	S	S	H	S
13	H	H	S	H	S	H	S	S	H	H
12	H	S	S	H	S	H	S	H	H	S
11	H	D	D	S	S	D	D	D	H	S
10	D	H	D	H	D	D	H	D	H	S
9	S	H	D	S	D	S	S	S	S	S
8	H	H	H	S	S	S	S	S	H	H
7	H	S	H	H	H	H	H	H	H	H
6	H	H	S	S	H	H	H	S	S	S
5	S	S	H	H	S	H	H	S	H	H
4	H	S	H	S	H	H	H	H	S	H

Iteration 200

Figure 4.10: Policy table for regular states created by a PSO algorithm.

Optimal	RL				Evolutionary			
	Q-learning		QV-learning		GA			PSO
	Greedy	41,26%	Greedy	39,06%		Single Point	Ranked	
∼42,5%	ϵ-greedy	42,30%	ϵ-greedy	42,47%	Ranked	42,30%	41,66%	21,21%
	Softmax	41,58%	Softmax	41,94%	Tourney	42,24%	41,69%	

Table 4.3: The average maximum win rate achieved by each different algorithm, including exploration policies for the RL algorithms. For the GA the horizontal entries represent the crossover ratio and the vertical entries represent the parent selection method.

4.3 Statistical tests

Table 4.3 shows an overview of the performance of all algorithms. The optimal known policy which is around 42,5% is included. As mentioned before in section 1, this optimal max win rate has no precise percentage and differs per source. For the RL algorithms each of the 50 agents has a win rate based on the last 100 000 played games, the average of these 50 win rates are displayed in the table. For the evolutionary algorithms the average of all its 750 agents are taken from the last generation.

An one way analysis of variance showed that the different exploration policies within the Q-learning algorithm have a significant difference, $F(2, 147) = 34, 94, p = 3, 86 \cdot 10^{-13}$. Post hoc analysis using the Nemenyi post hoc test showed that the average win rate of the ϵ-greedy exploration policy (M=42,30, SD=0,18) is significantly higher than the other exploration policies (greedy (M=41,26, SD=1,00) and softmax(M=41,58, SD=0,44)). Performing the same one way analysis of variance on the different exploration policies within QV-learning algorithm also showed significant differences, $F(2, 147) = 162, 56, p = 5.69 \cdot 10^{-38}$. Post hoc analysis using the Nemeny post hoc test showed that there is a significant difference between each exploration policy. Reporting from best to worst; ϵ-greedy (M=42,47, SD=0,19), softmax (M=41,94, SD=0,59) and greedy (M=39,06, SD=1,65). These statistical tests show that the ϵ-greedy exploration policy is significantly better than the other exploration policies.

An one way analysis of variance was also used for the different GA methods to show statistical difference, $F(3, 2996) = 1680, 7, p = 0, 0$. The statiscal test was done using the win rate of all agents (750) in the last generation of each variation of the algorithm. Using the Nemenyi post hoc test showed that each different GA setup was significantly different from each other, and thus the GA with ranked parent selection and single point crossover has the highest average win rate (M=42,30, SD=0,14). The PSO algorithm was excluded from any statistical test because it is clear to see that it is worse than any other algorithm. Another one way analysis of variance was used to show if there is a significant difference between the best variation of each algorithm, using Q-learning with ϵ-greedy, QV-learning with ϵ-greedy and GA with ranked parent selection and single point crossover, $F(2, 847) = 34, 46, p = 4, 08 \cdot 10^{-15}$. Using the Tukey-Kramer post hoc test showed that QV-learning (M=42,47, SD=0,19) is significantly better than the other algorithms (Q-learning (M=42,30, SD=0,18), GA (M=42,30, SD=0,14).

5 Discussion

As mentioned at the start of section 4, the average win rate from playing blackjack with random actions is 31,1%. This is the baseline which every algorithm will be judged by. As well as comparing the policy table created by each algorithm to the optimal policy table.

5.1 Reinforcement Learning

As can be seen in Table 4.3, both RL algorithms performed well. With the greedy exploration policy performing the worst for both algorithms. This was expected since it is known that greedy can get stuck in sub optimal policies since it does not encourage exploration, as soon as an action seems to do alright it will not try any of the others in that state even if it might be better in the long run. The softmax policy excels at learning quickly, as can be seen in Figure 4.3 and Figure 4.4 the win rate of the softmax policy achieves a high win rate very quickly for both algorithms after which it stabilizes and retains the high win rate until the end. However the best resulting exploration policy is ϵ-greedy, this policy takes longer to learn but in the end achieves better results for both algorithms with both going well over the 42%, supported by the statistical test in section 4.3. As can be concluded from the statistical test done in section 4.3, QV-learning is slightly better that Q-learning, with 42,47% and 42,30% respectively. The 42,47% win rate of QV-learning comes very close to the maximum possible win rate of around 42,5%. Comparing it to the baseline of playing blackjack with random actions, which has a 31,1% win rate, each exploration policy for both algorithms performed significantly better. Comparing the results of the policy tables of the RL algorithms (see Figure 4.5 and 4.6) to the optimal policy table (see Figure 4.1 and Figure 4.2) there are some significant differences. Most of the entries that are supposed to be *stand* according to the optimal table are also *stand* for both RL algorithms. Only a few entries differ for the 'edge' cases, which are those were it is less obvious whether hitting or standing is better and are located at the edge between the *hit* and *stand* entries. However more notably are the lack of *double down* and *split* (see appendix C for the split tables), this is possibly due to the fact that doubling down does not affect the win rate directly since it can be seen as the same actions as hitting and standing, in that order. It is only monetary rewards that are doubled when a game is won with doubling down. The lack of splits are possibly due to the fact that splits do not directly influence the win rate, but only the games resulting from the split do, which are treated separately. However this shows that without the perfect strategy a player can still achieve a near perfect win rate.

5.2 Evolutionary Algorithms

For the evolutionary algorithms the results are quite different from each other. The genetic algorithm achieved good results with the best version achieving a 42,30% win rate, supported by statistical test done in section 4.3. This is only slightly below the optimal win rate. Each version of the GA performed significantly better than the baseline of 31,1%. The policy tables for the GA as shown in Figure 4.9 are significantly different from the optimal policy table. This once again shows that playing a sub optimal strategy can still achieve a good win rate. Notably the *double down* and *split* (see appendix C for split table) actions are more common for the GA compared to the RL algorithms. The particle swarm optimization algorithm however did not find good results. With only a maximum of 32,7% win rate it barely performed better than the random baseline of 31,1%. This suggests that PSO was not a proper algorithm choice for this particular problem. The resulting 32,7% is only that of the best particle from PSO, whereas the average is much lower at 21,2% suggesting that it actively attempts to perform worse than random guessing. The learning curve of PSO is also very stagnant, there seems to be no learning at all, just randomly performing slightly better or worse than the previous iteration.

5.3 Comparison

Comparing the RL algorithms to the evolutionary algorithms (see table 4.3) as well as the statistical tests done in section 4.3 it can be concluded that that the RL algorithms in general are a better fit for the specific problem of a stochastic environment of blackjack. In particular the QV-learning algorithm performed the best out of all the algorithms, while only slightly better than Q-learning, it did achieve higher win rates with both ϵ-greedy and softmax policies. Another aspect which has not been mentioned thus far, is the run time of the algorithms. Due to the high number of agents and generations and the fact that each agent has to run for 100 000 games within each generation for the evolutionary algorithms, the run time was rather long. One run of a GA or PSO takes about two days on a medium performance PC. This is with multiprocessing included, without which it would take an additional few days. Whereas the reinforcement learning algorithms took only around one hour to finish all 50 agents with 1 000 000 games each. This is another reason to conclude the RL algorithms as better suited for learning blackjack.

6 Concluding remarks

The PSO algorithm performed rather badly. Considering the dimensionality of this problem is quite significant (>400) the search space for the algorithm becomes quite large. According to Piotrowski et al. (2020) problems with high dimensionallity might need a bigger particle swarm. This could mean 2000-4000 particles or maybe even more, with each running for 100 000 games per iteration this would make it extremely slow. Consider it already took two days to run with 750 particles, to run it with 4000 particles would take 10 days or more. One way to speed it up would be to play less games per particle per iteration, but this would create too much randomness in evaluation. Playing only a few games (around 10 000) could skew the results by a few percentages due to random luck or bad luck. Another reason that PSO might have performed badly is due to that it requires a continuous search space, whereas the state-action pairs are not continuous and were adapted to be so. This adaptation might have not resulted in better results. Continuing on this problem, the range within each dimension is rather small, only between -0,5 and a either 1,5, 2,5 or 3,5. This might give the algorithm a hard time to find the optimal location.

The policy tables for each algorithm had rather significant differences from the optimal table. However without the optimal strategy most algorithms still found very good results. Most notably the GA which had a significant difference in the policy table but still managed to achieve a 42,30% win rate. Considering the baseline of 31,1% there is only around 11,4% difference between playing completely randomly and playing fully optimal. So by even following a very simple strategy of hitting below 16 and standing at and above 16 would already increase the win rate significantly. In fact it will already increase the win rate to around 41,5%. Going from this simple strategy to the perfect strategy, only one percent more wins can be achieved. This shows that changing only one state-action pair, only reduces/increases the win rate by a very slight margin.

The fact that the optimal policy is hard to learn can be conceptualized by realizing that standing with a very low player hand of five or a higher hand of 16 results in the exact same win rate. This is due to the fact that the dealer will never have less than 17 points, since it will always hit below 17. The only way to win a hand of either five or 16 is if the dealer busts. This chance is equal no matter what the value of your hand is.

References

Garvie, D. (2017). *Blackjack.* Retrieved from `https://www.pagat.com/banking/blackjack.html`

Hassanat, A., Almohammadi, K., Alkafaween, E., Abunawas, E., Hammouri, A., & Prasath, S. (2019, December). Choosing mutation and crossover ratios for genetic algorithms-a review with a new dynamic approach.

Kakvi, S. (2009, September). Reinforcement learning for blackjack. In (p. 300-301).

Man, K.-F., Tang, K.-S., & Kwong, S. (2001). *Genetic algorithms: Concepts and designs.* Springer London, Limited.

Pathak, K., & Kapila, J. (2018, October). Reinforcement evolutionary learning method for self-learning..

Piotrowski, A., Napiórkowski, J., & Piotrowska, A. (2020, May). Population size in particle swarm optimization. *Swarm and Evolutionary Computation*, *58*.

Summerville, G. (2019, February). Winning blackjack using machine learning. *Towards Data Science*.

Suran, A. (2020, July). *On-policy v/s off-policy learning.* Retrieved from `https://towardsdatascience.com/on-policy-v-s-off-policy-learning-75089916bc2f`

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press, Cambridge.

Thorp, E. O. (1966). *Beat the dealer : a winning strategy for the game of twenty-one.* Random House.

Tijjani, S., & Ozkaya, A. (2014, November). A comparison of reinforcement learning and evolutionary algorithms for container loading problem..

Wang, D., Tan, D., & Liu, L. (2018, January). Particle swarm optimization algorithm: an overview. *Soft Computing*, *22*.

Watkins, C. (1989, January). Learning from delayed rewards.

Wiering, M. (2005, January). Qv(lambda)-learning: A new on-policy reinforcement learning algrithm.

Yanes Luis, S., Gutiérrez, D., & Toral, S. (2021, April). A dimensional comparison between evolutionary algorithm and deep reinforcement learning methodologies for autonomous surface vehicles with water quality sensors. *Sensors*, *2021*.

Zhang, S., & Zaïane, O. (2017, 11). Comparing deep reinforcement learning and evolutionary methods in continuous control.

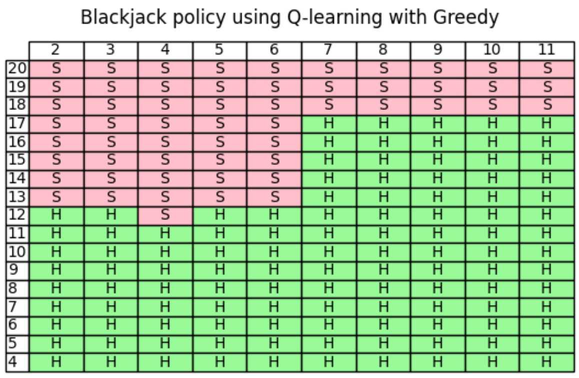
A Random chance baseline

To create a random play baseline the simulation was run 20 times with 100 000 epochs each. The results are generated with random seed 1 through 20 respectively. This gave the results shown in Table A.1. The resulting average win rate is 31,1%. Multiple simulations where done to minimize create an accurate baseline with minimal randomness.

Table A.1: win rates returned by randomly picking actions.

31,07%	30,90%	30,89%	31,21%	30,64%
30,83%	31,26%	30,93%	31,34%	31,58%
30,68%	31,29%	31,43%	31,11%	30,96%
31,23%	31,34%	31,20%	31,36%	31,15%

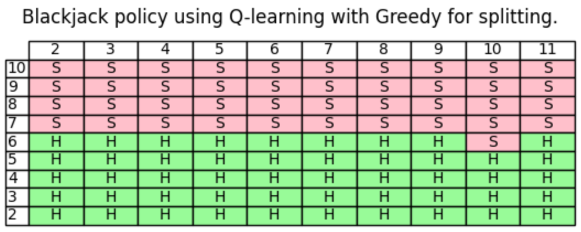
B Policy tables for RL



Blackjack policy using Q-learning with Greedy

	2	3	4	5	6	7	8	9	10	11
20	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S
18	S	S	S	S	S	S	S	S	S	S
17	S	S	S	S	S	H	H	H	H	H
16	S	S	S	S	S	H	H	H	H	H
15	S	S	S	S	S	H	H	H	H	H
14	S	S	S	S	S	H	H	H	H	H
13	S	S	S	S	S	H	H	H	H	H
12	H	H	S	H	H	H	H	H	H	H
11	H	H	H	H	H	H	H	H	H	H
10	H	H	H	H	H	H	H	H	H	H
9	H	H	H	H	H	H	H	H	H	H
8	H	H	H	H	H	H	H	H	H	H
7	H	H	H	H	H	H	H	H	H	H
6	H	H	H	H	H	H	H	H	H	H
5	H	H	H	H	H	H	H	H	H	H
4	H	H	H	H	H	H	H	H	H	H

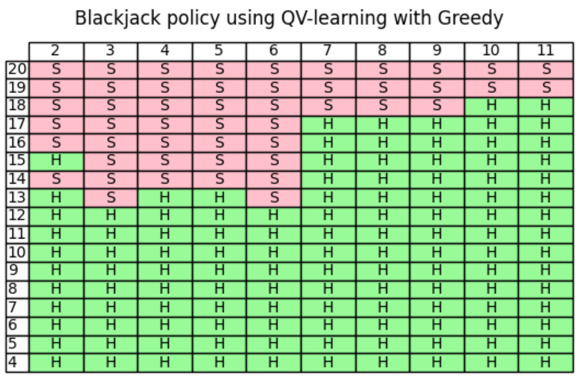
Figure B.1: The average policy of 50 agents using Q-learning with greedy exploration policy for regular hands.



Blackjack policy using Q-learning with Greedy for splitting.

	2	3	4	5	6	7	8	9	10	11
10	S	S	S	S	S	S	S	S	S	S
9	S	S	S	S	S	S	S	S	S	S
8	S	S	S	S	S	S	S	S	S	S
7	S	S	S	S	S	S	S	S	S	S
6	H	H	H	H	H	H	H	H	S	H
5	H	H	H	H	H	H	H	H	H	H
4	H	H	H	H	H	H	H	H	H	H
3	H	H	H	H	H	H	H	H	H	H
2	H	H	H	H	H	H	H	H	H	H

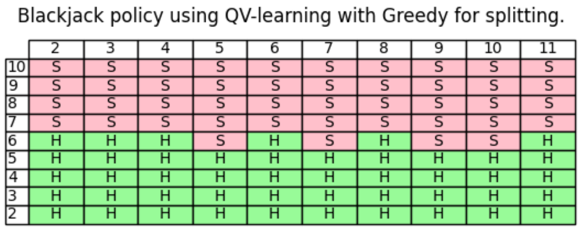
Figure B.2: The average policy of 50 agents using Q-learning with greedy exploration policy for split hands.



Blackjack policy using QV-learning with Greedy

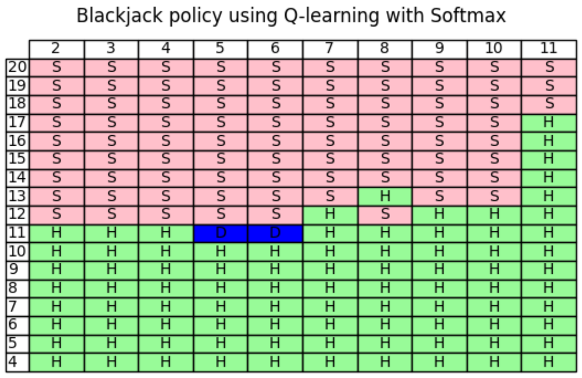
	2	3	4	5	6	7	8	9	10	11
20	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S
18	S	S	S	S	S	S	S	S	H	H
17	S	S	S	S	S	H	H	H	H	H
16	S	S	S	S	S	H	H	H	H	H
15	H	S	S	S	S	H	H	H	H	H
14	S	S	S	S	S	H	H	H	H	H
13	H	S	H	H	S	H	H	H	H	H
12	H	H	H	H	H	H	H	H	H	H
11	H	H	H	H	H	H	H	H	H	H
10	H	H	H	H	H	H	H	H	H	H
9	H	H	H	H	H	H	H	H	H	H
8	H	H	H	H	H	H	H	H	H	H
7	H	H	H	H	H	H	H	H	H	H
6	H	H	H	H	H	H	H	H	H	H
5	H	H	H	H	H	H	H	H	H	H
4	H	H	H	H	H	H	H	H	H	H

Figure B.3: The average policy of 50 agents using QV-learning with greedy exploration policy for regular hands.


Blackjack policy using QV-learning with Greedy for splitting.

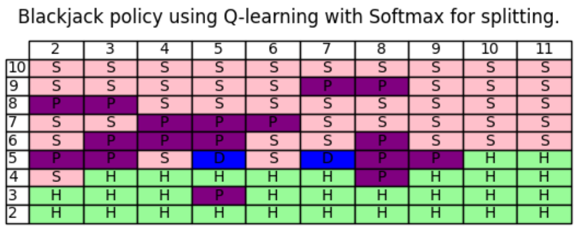
	2	3	4	5	6	7	8	9	10	11
10	S	S	S	S	S	S	S	S	S	S
9	S	S	S	S	S	S	S	S	S	S
8	S	S	S	S	S	S	S	S	S	S
7	S	S	S	S	S	S	S	S	S	S
6	H	H	H	S	H	S	H	S	S	H
5	H	H	H	H	H	H	H	H	H	H
4	H	H	H	H	H	H	H	H	H	H
3	H	H	H	H	H	H	H	H	H	H
2	H	H	H	H	H	H	H	H	H	H

Figure B.4: The average policy of 50 agents using QV-learning with greedy exploration policy for split hands.


Blackjack policy using Q-learning with Softmax

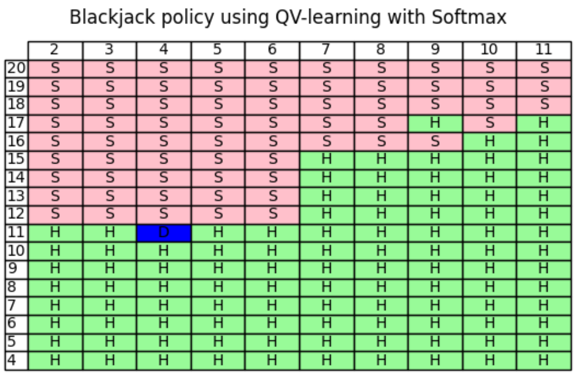
	2	3	4	5	6	7	8	9	10	11
20	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S
18	S	S	S	S	S	S	S	S	S	S
17	S	S	S	S	S	S	S	S	S	H
16	S	S	S	S	S	S	S	S	S	H
15	S	S	S	S	S	S	S	S	S	H
14	S	S	S	S	S	S	S	S	S	H
13	S	S	S	S	S	S	H	S	S	H
12	S	S	S	S	S	H	S	H	H	H
11	H	H	H	D	D	H	H	H	H	H
10	H	H	H	H	H	H	H	H	H	H
9	H	H	H	H	H	H	H	H	H	H
8	H	H	H	H	H	H	H	H	H	H
7	H	H	H	H	H	H	H	H	H	H
6	H	H	H	H	H	H	H	H	H	H
5	H	H	H	H	H	H	H	H	H	H
4	H	H	H	H	H	H	H	H	H	H

Figure B.5: The average policy of 50 agents using Q-learning with softmax exploration policy for regular hands.


Blackjack policy using Q-learning with Softmax for splitting.

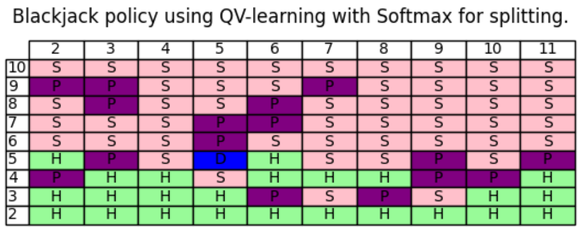
	2	3	4	5	6	7	8	9	10	11
10	S	S	S	S	S	S	S	S	S	S
9	S	S	S	S	S	P	P	S	S	S
8	P	P	S	S	S	S	S	S	S	S
7	S	S	P	P	P	S	S	S	S	S
6	S	P	P	P	S	S	P	S	S	S
5	P	P	S	D	S	D	P	P	H	H
4	S	H	H	H	H	H	P	H	H	H
3	H	H	H	P	H	H	H	H	H	H
2	H	H	H	H	H	H	H	H	H	H

Figure B.6: The average policy of 50 agents using Q-learning with softmax exploration policy for split hands.


Blackjack policy using QV-learning with Softmax

	2	3	4	5	6	7	8	9	10	11
20	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S
18	S	S	S	S	S	S	S	S	S	S
17	S	S	S	S	S	S	S	H	S	H
16	S	S	S	S	S	S	S	S	H	H
15	S	S	S	S	S	H	H	H	H	H
14	S	S	S	S	S	H	H	H	H	H
13	S	S	S	S	S	H	H	H	H	H
12	S	S	S	S	S	H	H	H	H	H
11	H	H	D	H	H	H	H	H	H	H
10	H	H	H	H	H	H	H	H	H	H
9	H	H	H	H	H	H	H	H	H	H
8	H	H	H	H	H	H	H	H	H	H
7	H	H	H	H	H	H	H	H	H	H
6	H	H	H	H	H	H	H	H	H	H
5	H	H	H	H	H	H	H	H	H	H
4	H	H	H	H	H	H	H	H	H	H

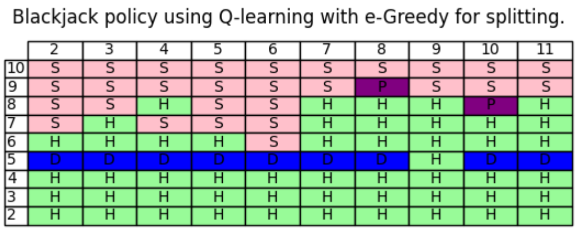
Figure B.7: The average policy of 50 agents using QV-learning with softmax exploration policy for regular hands.


Blackjack policy using QV-learning with Softmax for splitting.

	2	3	4	5	6	7	8	9	10	11
10	S	S	S	S	S	S	S	S	S	S
9	P	P	S	S	S	P	S	S	S	S
8	S	P	S	S	P	S	S	S	S	S
7	S	S	S	P	P	S	S	S	S	S
6	S	S	S	P	S	S	S	S	S	S
5	H	P	S	D	H	S	S	P	S	P
4	P	H	H	S	H	H	H	P	P	H
3	H	H	H	H	P	S	P	S	H	H
2	H	H	H	H	H	H	H	H	H	H

Figure B.8: The average policy of 50 agents using QV-learning with softmax exploration policy for split hands.

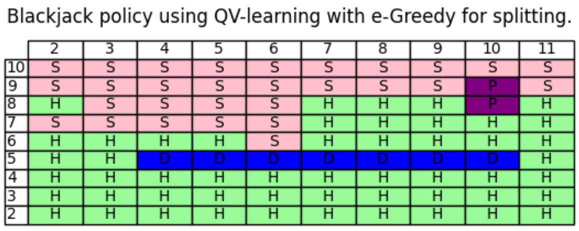
C Policy split tables for Q- and QV-learning with ϵ-greedy and GA



Blackjack policy using Q-learning with e-Greedy for splitting.

	2	3	4	5	6	7	8	9	10	11
10	S	S	S	S	S	S	S	S	S	S
9	S	S	S	S	S	S	P	S	S	S
8	S	S	H	S	S	H	H	H	P	H
7	S	H	S	S	S	H	H	H	H	H
6	H	H	H	H	S	H	H	H	H	H
5	D	D	D	D	D	D	D	H	D	D
4	H	H	H	H	H	H	H	H	H	H
3	H	H	H	H	H	H	H	H	H	H
2	H	H	H	H	H	H	H	H	H	H

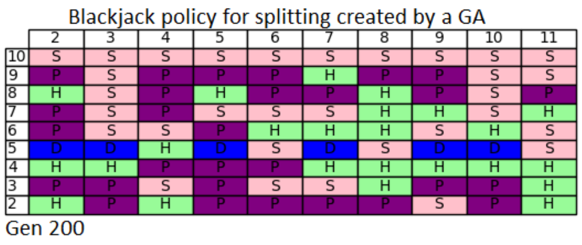
Figure C.1: The average policy of 50 agents using Q-learning with ϵ-greedy exploration policy for split hands.



Blackjack policy using QV-learning with e-Greedy for splitting.

	2	3	4	5	6	7	8	9	10	11
10	S	S	S	S	S	S	S	S	S	S
9	S	S	S	S	S	S	S	S	P	S
8	H	S	S	S	S	H	H	H	P	H
7	S	S	S	S	S	H	H	H	H	H
6	H	H	H	H	S	H	H	H	H	H
5	H	H	D	D	D	D	D	D	D	H
4	H	H	H	H	H	H	H	H	H	H
3	H	H	H	H	H	H	H	H	H	H
2	H	H	H	H	H	H	H	H	H	H

Figure C.2: The average policy of 50 agents using QV-learning with ϵ-greedy exploration policy for split hands.



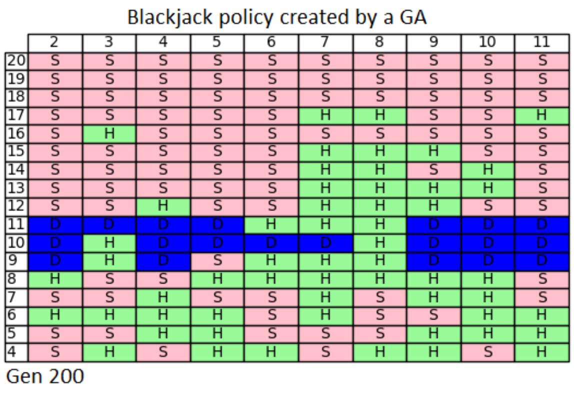
Blackjack policy for splitting created by a GA

	2	3	4	5	6	7	8	9	10	11
10	S	S	S	S	S	S	S	S	S	S
9	P	S	P	P	P	H	P	P	S	S
8	H	S	P	H	P	P	H	P	S	P
7	P	S	P	S	S	S	H	H	S	H
6	P	S	S	P	H	H	H	S	H	S
5	D	D	H	D	S	D	S	D	D	S
4	H	H	P	P	P	H	H	H	H	H
3	P	P	S	P	S	S	H	P	P	H
2	H	P	H	P	P	P	P	S	P	H

Gen 200

Figure C.3: Policy table for split states created by a genetic algorithm with ranked parent selection and single point crossover.

D Policy tables for GA

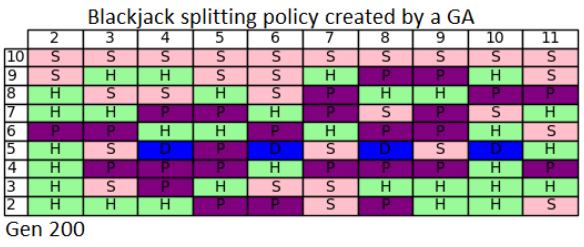


Blackjack policy created by a GA

	2	3	4	5	6	7	8	9	10	11
20	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S
18	S	S	S	S	S	S	S	S	S	S
17	S	S	S	S	S	H	H	S	S	H
16	S	H	S	S	S	S	S	S	S	S
15	S	S	S	S	S	H	H	H	S	S
14	S	S	S	S	S	H	H	S	H	S
13	S	S	S	S	S	H	H	H	H	S
12	S	S	H	S	S	H	H	H	S	S
11	D	D	D	D	H	H	H	D	D	D
10	D	H	D	D	D	D	H	D	D	D
9	D	H	D	S	H	H	H	D	D	D
8	H	S	S	H	H	H	H	H	H	S
7	S	S	H	S	S	H	S	H	H	S
6	H	H	H	H	S	H	S	S	H	H
5	S	S	H	H	S	S	S	H	H	H
4	S	H	S	H	H	S	H	H	S	H

Gen 200

Figure D.1: The blackjack policy of the best agent in the last generation created by a GA with ranked parent selection and ranked crossover.

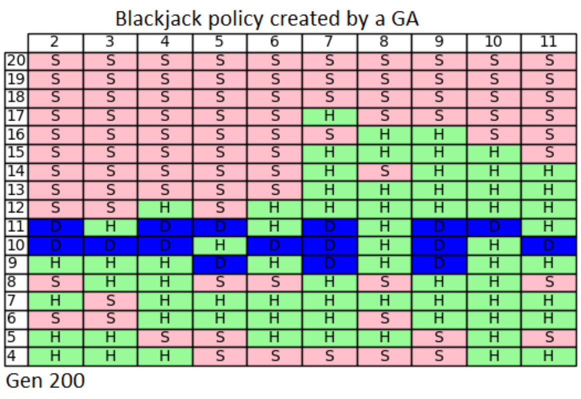


Blackjack splitting policy created by a GA

	2	3	4	5	6	7	8	9	10	11
10	S	S	S	S	S	S	S	S	S	S
9	S	H	H	S	S	H	P	P	H	S
8	H	S	S	H	S	P	H	H	P	P
7	H	H	P	P	H	P	S	P	S	H
6	P	P	H	H	P	H	P	P	H	S
5	H	S	D	P	D	S	D	S	D	H
4	H	P	P	P	H	P	P	P	H	P
3	H	S	P	H	S	S	H	H	H	H
2	H	H	H	P	P	S	P	H	H	S

Gen 200

Figure D.2: The blackjack policy for splitting of the best agent in the last generation created by a GA with ranked parent selection and ranked crossover.

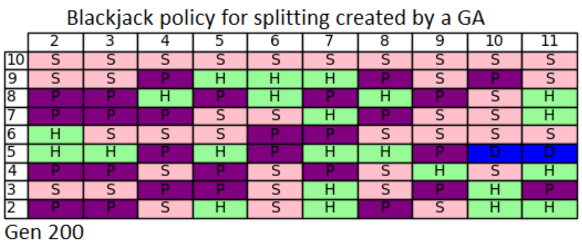


Blackjack policy created by a GA

	2	3	4	5	6	7	8	9	10	11
20	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S
18	S	S	S	S	S	S	S	S	S	S
17	S	S	S	S	S	H	S	S	S	S
16	S	S	S	S	S	S	H	H	S	S
15	S	S	S	S	S	H	H	H	H	S
14	S	S	S	S	S	H	S	H	H	H
13	S	S	S	S	S	H	H	H	H	H
12	S	S	H	S	H	H	H	H	H	H
11	D	H	D	D	H	D	H	D	D	H
10	D	D	D	H	D	D	H	D	H	D
9	H	H	H	D	H	D	H	D	H	H
8	S	H	H	S	S	H	S	H	H	S
7	H	S	H	H	H	H	H	H	H	H
6	S	S	H	H	H	H	S	H	H	H
5	H	H	S	S	H	H	H	S	H	S
4	H	H	H	S	S	S	S	S	H	H

Gen 200

Figure D.3: The blackjack policy of the best agent in the last generation created by a GA with tourney parent selection and single point crossover.

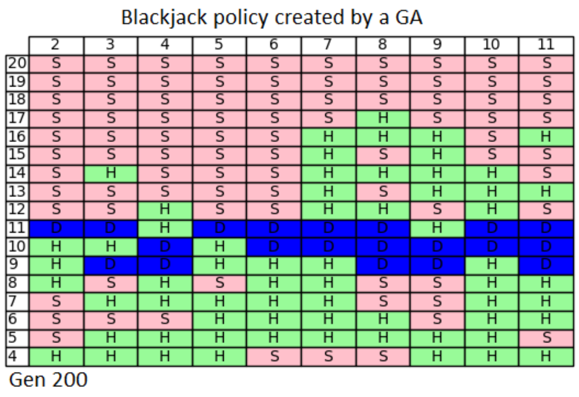


Blackjack policy for splitting created by a GA

	2	3	4	5	6	7	8	9	10	11
10	S	S	S	S	S	S	S	S	S	S
9	S	S	P	H	H	H	P	S	P	S
8	P	P	H	P	H	P	H	P	S	H
7	P	P	P	S	S	H	P	S	S	H
6	H	S	S	S	P	P	S	S	S	S
5	H	H	P	H	P	H	H	P	D	D
4	P	P	S	P	S	P	S	H	S	H
3	S	S	P	P	S	H	S	P	H	P
2	P	P	S	H	S	H	P	S	H	H

Gen 200

Figure D.4: The blackjack policy for splitting of the best agent in the last generation created by a GA with tourney parent selection and single point crossover.

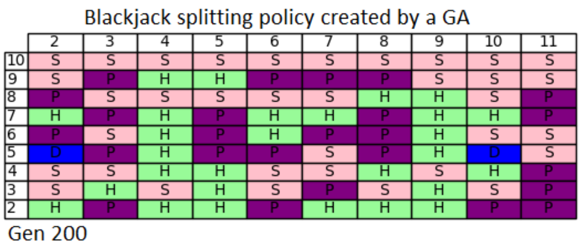


Blackjack policy created by a GA

	2	3	4	5	6	7	8	9	10	11
20	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S
18	S	S	S	S	S	S	S	S	S	S
17	S	S	S	S	S	S	H	S	S	S
16	S	S	S	S	S	H	H	H	S	H
15	S	S	S	S	S	H	S	H	S	S
14	S	H	S	S	S	H	H	H	H	S
13	S	S	S	S	S	H	S	H	H	H
12	S	S	H	S	S	H	H	S	H	S
11	D	D	H	D	D	D	D	H	D	D
10	H	H	D	H	D	D	D	D	D	D
9	H	D	D	H	H	H	D	D	H	D
8	H	S	H	S	H	H	S	S	H	H
7	S	H	H	H	H	H	S	S	H	H
6	S	S	S	H	H	H	H	S	H	H
5	S	H	H	H	H	H	H	H	H	S
4	H	H	H	H	S	S	S	H	H	H

Gen 200

Figure D.5: The blackjack policy of the best agent in the last generation created by a GA with tourney parent selection and ranked crossover.



Blackjack splitting policy created by a GA

	2	3	4	5	6	7	8	9	10	11
10	S	S	S	S	S	S	S	S	S	S
9	S	P	H	H	P	P	P	S	S	S
8	P	S	S	S	S	S	H	H	S	P
7	H	P	H	P	H	H	P	H	H	P
6	P	S	H	P	H	P	P	H	S	S
5	D	P	H	P	P	S	P	H	D	S
4	S	S	H	H	S	S	H	S	H	P
3	S	H	S	H	S	P	S	H	S	P
2	H	P	H	H	P	H	H	H	P	P

Gen 200

Figure D.6: The blackjack policy for splitting of the best agent in the last generation created by a GA with tourney parent selection and ranked crossover.