A Self-Learning Reinforcement Learning-Based Blackjack

Yashwanth Goud Chithaloori
Masters Artificial Intelligence
Technical University of Applied Sciences Wurzburg-Schweinfurt
yashwanthgoud.chithaloori@study.thws.de

***Abstract*—Many casinos have a popular card game known as blackjack. The goal of the game is to win money by having a higher point total than the dealer, but not going over 21. This paper presents the implementation and training of Blackjack using Q-learning. The agent will learn from its experience, and this reinforcement learning helps the agent to deal with decision-making challenges like Blackjack. We compare two strategies: a basic strategy with a Q-learning agent that learns Blackjack's optimal policy and an enhanced version that integrates the Hi-Lo Complete point count. Basic strategy follows a standard rule-book for blackjack, and card counting tracks the game by the good and bad cards that are left in the deck. The agent was evaluated under standard Blackjack rules and two rule variations: "Dealer Hits Soft 17" and "Double After Split Allowed." We explore the impact of these variations on strategy effectiveness and analyze learning outcomes through expected value and win-rate metrics. A Double Q-learning variant is also implemented to improve stability. Despite being limited to tabular Q-learning, the system demonstrates how reinforcement learning adapts to probabilistic environments like Blackjack.**

***Index Terms*— Blackjack, Reinforcement Learning, Q-learning, Basic Strategy, Card Counting, Hi-lo Count, Rule Variations, Decision Making, Expected Value, Win Rate.**

I. INTRODUCTION

Blackjack is a classic casino game that involves uncertain probability and strategic decision-making. Players use heuristic strategies like Basic Strategy or card count systems such as Hi-Lo point count to optimize their decisions. Especially Reinforcement Learning(RL) techniques like Q-learning and its Double Q-learning variant to develop and train agents that play Blackjack effectively.

A. What is Blackjack ?

Blackjack, a game played between one or more players and the dealer. The objective is to get a hand total near to 21 than the dealer's hand, without exceeding 21. It works by assigning every card a point value. Cards from 2 to 10 are numbered. Cards are worth their face value, while face cards, Jack, Queen, and King, are worth 10 points [2]. An ace counted as either 1 or 11 points, depending on which value is high favorable to the hand's busting. Before starting the game, each player at the table should place a bet. after the initial bet, each player gets two face-up cards, while the dealer receives one face-up card and one face-down card. The major challenge for the player is to select the best action based on their own cards and the dealer's visible card. Here, the available actions are: hit(take another card), stand(keep the current hand), split(if holding cards of the same value), or double(double the bet and receive one final card).

Players can request an additional card from the dealer, an action known as "hitting," and may continue to do so as long as their hand total does not exceed 21. Should a player wish to stop receiving cards, they can choose to "stand." Furthermore, if a player's initial two cards are of identical value, they have the option to "split" them, effectively creating two separate hands to play independently [1]. For example, if two 6's are present, one may split and play each 6 individually. The action, however, requires an additional bet equal to the original wager. Alternatively, at any point, a player may opt to double down, which involves doubling their current bet in exchange for receiving just one more card from the dealer.

B. What is Reinforcement Learning(RL)?

Reinforcement learning(RL) fundamentally operates on a principle of trial and error, where an agent interacts with an environment and receives feedback. This feedback is expressed as a numerical reward signal, which serves as the primary mechanism for the agent's learning process [1]. With rewards, the agent refines its policy. This policy dictates the actions it chooses to take in various states, with the goal of maximizing its cumulative reward over time. In RL, techniques such as Q-learning and Double Q-learning work by providing feedback signals to improve the decision-making process of the agent.

C. What are Basic strategy and complete point count?

The "basic strategy" provides a set of optimal guidelines for players, dictating when to hit, stand, double down, or split based solely on their hand and the dealer's visible card, thereby maximizing their winning probabilities. Complete point count System(Hi-lo system) assigns numerical values to individual cards - positive(+1) for low cards(2-6), negative(-1) doe high cards(10's,face cards,Aces),and zero(0) for neutral cards(7-9) [2].By tracking these totals, players gain a statistical edge, allowing them to strategically increase their bets when the count is favorable(high) and decrease them when it's unfavorable (low).

D. What is Q learning?

Q-learning is an off-policy Reinforcement Learning(RL) technique. The Q-learning algorithm is a temporal difference(TD) method that directly approximates the optimal action-value function, denoted as Q*(s, a).Q value estimates how good it is to take a particular action in a particular state [1]. The Q-learning algorithm is outlined below.

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big] \quad (1)$$

The parameter α, known as the learning rate, controls the magnitude of the adjustment made to the Q-values during each update step. The parameter γ, referred to as the discount rate, determines the importance of future rewards. Rewards are referred to as r.

II. Methodology

This part will cover how the blackjack environment is implemented, how the Q-learning techniques are implemented for basic strategy, complete point count system and double Q-learning for updated point count.

A. Environment Setup

Full-featured Blackjack environment is developed with Python, which supports multiple decks, each deck with a standard 52-card deck, including hearts, diamonds, clubs, and spades, card dealing, hand evaluation, and advanced actions including splitting and doubling down. The environment tracks Hi-Lo running and true counts [2].

- Deck Management
  The environment initializes a shoe that contains standard 52-card decks(4 decks). Card values are assigned as follows: 2-9 retain their face value, 10s and face cards (King, Queen, Jack) are valued at 10, and Aces are valued at 11.
- Hand Calculation
  Hand value is responsible for accurately determining the total score of any given hand. A critical method is its handling of Aces. Initially, an ace contains 11 points in the hand if the hand value exceeds 21(bust), the hand value is re-evaluated to 1.
- Player Actions
  Hit- the player draws an additional card. If a player's hand exceeds 21, they immediately bust. Stand-The player decides not to take any more cards Double Down- The player doubles their initial bet, receives exactly one more card, and then automatically stands. This action is typically restricted to the player's initial two cards and is not allowed if a hand has already been doubled or split. Split-If a player had two similar cards of the same rank, they can split them into two separate hands, and an additional bet equal to the original is placed for the second hand. Each hand played independently
- Dealer play
  Dealer continues to draw cards until their hand value is 17 or greater.
- Reward
  Rewards are crucial for guiding the Q-learning process. For Win: +1 units(or +1.5 for player blackjack if dealer does not blackjack, loss:-1 unit, draw: 0 units, double down: final reward is multiplied by 2. this encourage the agent to learn optimal doubling down opportunities, bust: an immediate loss:-1 unit for regular bust, -2 for a double down bust.
- Point count(Hi-Lo)
  The environment includes point count system, where 2-6 low cards count +1, 10-Ace: cards count -1, and 7-9: neutral cards count of 0.

B. Implementation of Basic Strategy with Q-learning

Q-learning is an off-policy reinforcement learning approach, an agent learns to play Blackjack by using a Q-table. This table acts as a lookup for every possible game situation, indicating the estimated reward associated with each available action in that specific state. The agent employs an epsilon-greedy policy. Initially, the agent explores more by making random decisions, but gradually the agent exploits after a few episodes, by using different variations (hit, stand, double down, and split). Action selection follows an ε-greedy strategy: with probability ε, the agent explores by selecting a random valid action, and with probability $1-\varepsilon$, it exploits the best Q-values. The ε value decays over time to encourage more exploitation in later stages of training, starting at 1.0 and decreased gradually to a final minimum of 0.01 As shown in Figure(1).

In basic strategy, the agent will consider actions while determining:the total value of the player's current hand, dealer's visible card, whether the hand is soft, and whether the player is allowed to split or double down.Along with that it will track the hand which is currently being played in multiple-hand scenarios resulting from splitting.These representations allow the agent to learn strategies that consider both basic game dynamics and advanced option like doubling and splitting .

The Q-values are updated using the standard temporal-difference (TD) learning rule. After each action, the agent gets a reward from the environment and it updates the Q-value of the previous state-action pair accordingly. The update rule used is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

Here, $Q(s,a)$ represents the current Q-value for taking action a in state s, while α denotes the learning rate, and r is the reward received after performing the action a in state s. The term γ is the discount factor, and $\max_{a'} Q(s',a')$ represents the maximum estimated future reward attainable from the next state s'. The entire expression $[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$ is known as the temporal difference (TD) error. The agent is trained over multiple episodes using this update rule, allowing it to iteratively adjust its Q-values depending on the observed rewards and transitions. This process enables the agent to progressively improve its policy by balancing exploration and exploitation.

Throughout training, performance statistics such as wins, losses, draws, and total profit are logged, and the agent's policy is refined based on repeated play across over one million simulated games. The final learned policy reflects

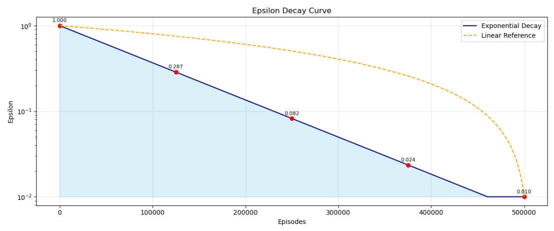

Fig. 1. Epsilon decay curve over episodes.

an approximation of the traditional basic strategy, adapted through reinforcement learning under the specific game rules and constraints defined in the simulation

C. *Complete point count with Q-learning*

To further enhance the agent's performance beyond basic strategy, the Q-learning approach was implemented, incorporating a card-counting mechanism. This `QLearningPointCountAgent` leverages the true count, a dynamic indicator of the remaining deck's favorability, to inform both strategic decisions and betting adjustments. The state representation for this agent is augmented state representation for this agent is augmented to $(player\_value$, $dealer\_upcard$, is_soft, can_split, can_double, $hand\_index$, $tc\_bucket)$, where tc_bucket discreteness the true count into predefined bins. quantized the true count into discrete bins. This expansion allows the Q-table to learn optimal actions contingent on the current card advantage. While the core Q-learning update rule, utilizing a learning rate $\alpha = 0.15$ [4] and a discount factor $\gamma = 0.95$, remains consistent, rewards are now scaled by a dynamically determined bet_amount. This bet_amount is predefined by the betting strategy (`self.betting_strategy`) that increases the wager as the true count becomes more favorable (higher profit expectancy). The training process follows an ϵ-greedy exploration schedule, decaying from $initial\_epsilon = 1.0$ to $final\_epsilon = 0.01$ over $num\_episodes = 500,000$ via an $epsilon\_decay = 0.99999$ factor as shown in Figure(1), ensuring sufficient exploration before converging on the optimal policy tailored to various true count scenarios. It is important to note that the detailed implementation of the card counting rules and true count calculation (`self.env.get_true_count()`) resides within the Blackjack environment, providing the necessary input to the agent's state representation. This integrated approach enables the agent to learn strategy deviations and betting adjustments that exploit fluctuating odds, a hallmark of professional Blackjack play.

D. *Rule variations with Basic Strategy and Complete point count*

Beyond the core Blackjack rules, our implementation specifically investigates the impact of two common rule variations on optimal strategies: "Dealer Hits Soft 17" (H17) and "Double After Split Allowed" (DAS). These variations are integrated into a specialized `BlackjackEnvironmentWithRuleVariations` class, which inherits from the standard `Blackjack Environment` and overrides relevant methods to implement these specific rule changes.

Dealer Hits Soft 17 (H17): The "Dealer Hits Soft 17" rule is handled within the `play_dealer_hand` method of the `BlackjackEnvironmentWithRuleVariations` class. While the standard environment (and many casinos) requires the dealer to stand on all 17s (S17), the H17 variation modifies this. Under this rule, if the dealer's hand total is 17 and includes an Ace valued as 11("soft 17"), the dealer is compelled to take an additional card. This seemingly minor change typically increases the house edge, subtly shifting optimal player strategies, particularly regarding decisions to hit or stand against a dealer's Ace.

Double After Split Allowed (DAS): The "Double After Split Allowed" rule is managed within the `execute_action` method. In standard Blackjack, players are generally prohibited from doubling down on a hand that originated from a split. This variation lifts that restriction, allowing players to double their bet on a hand that has been created by a split (provided it's their first move on that specific split hand). This offers an additional strategic opportunity for the player to capitalize on favorable situations where doubling is beneficial.

For each rule variation, both the Basic Strategy Q-learning agent and the Point Count Q-learning agent are trained and evaluated independently over 1,500,000 episodes. This systematic evaluation allows for a direct comparison of their performance (measured by win percentage, loss percentage, draw percentage, and expected value) under varying game conditions. This comparative analysis provides critical insights into how these specific rule changes influence the learned optimal policies and the overall profitability of each strategy.

E. *Enhanced Complete Point Counting with Double Q-learning*

To address the known overestimation bias inherent in standard Q-learning, a **Double Q-Learning** [3] approach was implemented for the card-counting agent. This method employs two independent Q-tables, Q_1 and Q_2, to decouple action selection from action evaluation, thereby mitigating the tendency to overestimate action values.

The state representation remains consistent with the single Q-learning point count agent: $(player\_value$,$dealer\_upcard$, is_soft, can_split, can_double, $hand\_index$, $tc\_bucket)$, incorporating the binned true count (tc_bucket). For action selection, the agent utilizes the sum of the Q-values from both tables, $Q_1(s,a) + Q_2(s,a)$, to determine the most promising action under the ϵ-greedy policy. This ensures that the chosen action reflects a consensus from both estimators.

The key distinction lies in the update rule. After taking an action a in state s and observing reward r and next state s', one of the Q-tables (Q_1 or Q_2) is randomly selected for the update. If Q_1 is chosen for the update, the target value is calculated using Q_1 to select the best next action, but Q_2 is used to evaluate its value, and vice-versa. Specifically, the update equations are:

If a random value is less than 0.5 (update Q_1):

$$Q_1(s,a) \leftarrow Q_1(s,a) + \alpha \left[r + \gamma Q_2 \left(s', \arg\max_a Q_1(s',a) \right) - Q_1(s,a) \right] \tag{2}$$

Otherwise (update Q_2):

$$Q_2(s,a) \leftarrow Q_2(s,a) + \alpha \left[r + \gamma Q_1(s', \arg\max_a Q_2(s',a)) - Q_2(s,a) \right] \tag{3}$$

Here, $\alpha = 0.015$ serves as the learning rate and $\gamma = 0.95$ as the discount factor. Rewards (r) are scaled by the dynamically determined bet_amount, which is derived from the environment's $true\_count$ using the same predefined betting strategy as the single Q-learning agent. The training process, spanning $num\_episodes = 1,500,000$, continues to employ an ϵ-greedy strategy with ϵ decaying exponentially from $initial\_epsilon = 1.0$ to $final\_epsilon = 0.01$ with an $epsilon\_decay = 0.99999$ factor. This dual-table approach aims to achieve more stable and accurate value estimates, potentially leading to a more robust and truly optimal playing and betting strategy in the presence of stochastic rewards and delayed outcomes inherent in Blackjack.

III. Experiments and Evaluation

This section shows the experiments done with various techniques for playing blackjack. This includes basic strategy with Q-learning and point count with Q-learning.

Initially, the basic strategy and complete point were trained using Q-learning with 10,000 and 100,000 episodes. The winning percentage of both was around 29 and 33 percent. After increasing the number of episodes and training the agent, the winning percentage has increased gradually. Changing the alpha value for both the basic strategy and the complete point count, initially with 0.1, the winning rate is less than the alpha value of 0.15.The next section will discuss about results of our experiments.

IV. Results and Discussion

In this section, the results of all strategies which implemented in blackjack are discussed. subsection Outcome of Basic Strategy with Q-learning The agent was trained to learn the optimal basic strategy without card counting, underwent 1,500,000 episodes of simulation. The training progress, specifically the decay of the ϵ-greedy exploration rate, is presented in Table I. Epsilon, initialized at 1.0, decayed exponentially with a factor of 0.99999 until it reached its final_epsilon of 0.01 around 500,000 episodes, after which it remained constant, signifying a shift from exploration to exploitation.The performance of the Q-learning Basic Strategy Agent over the 1,500,000 games is summarized in Table I.

The performance of the Q-learning Basic Strategy Agent over the 1,500,000 games is summarized in Table I.

The total profit across all simulations was -137,341.00 units, resulting in an Expected Value (EV) per hand of -0.0916 as

TABLE I
Epsilon Decay during Q-Learning Basic Strategy Training

Episode	Epsilon
100,000	0.3679
300,000	0.0498
500,000	0.0100
100,000	0.0100
⋮	⋮
1,500,000	0.0100

TABLE II
Basic Strategy with Q-Learning reults

Strategy	Episodes	Win Rate	Loss Rate	Draw Rate
Basic	1,00,000	32.72%	60.97%	6.31%
Basic	15,00,000	40.10%	50.38%	9.52%

shown in Figure(2). This negative Expected Value is consistent with theoretical expectations for Blackjack played using perfect basic strategy, where the house typically maintains a slight edge (often cited around 0.5% to 1.0% depending on rules, which translates to an EV of -0.005 to -0.01 per unit bet). Our observed EV of -0.0916 per hand (or -9.16% of the initial bet) is notably higher than the theoretical house edge for a typical basic strategy.

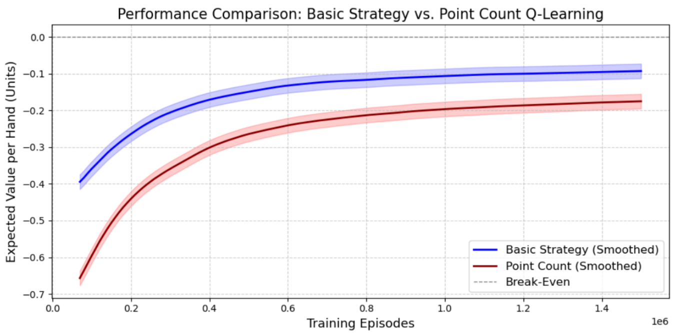

Fig. 2. Performance Comparison Q learning and Complete point count.

A. Q-Learning with Point Counting

As like Q-learning Basic strategy Point count agent is trained to learn the optimal policy. The performance of the Q-learning Point Count Agent across the 1,500,000 simulated games is detailed in Table III.

Expected Value per hand for the Point Count agent is significantly more negative (-0.1739 vs. -0.0916) As shown in Figure(2). This outcome is counterintuitive to the established benefits of card counting in Blackjack, which is designed to provide a positive player edge or, at minimum, significantly reduce the house edge.

As summarized in Table III, the Point Count Q-learning agent achieved a win rate of 40.14%, a loss rate of 50.89%, and a draw rate of 8.97%. Over the 1.5 million games, the total profit recorded was -260,845.00 units, resulting in an Expected Value (EV) per hand of -0.1739. Comparing these results to the Basic Strategy Q-learning agent, a notable shift

TABLE III
COMPLETE POINT COUNT WITH Q-LEARNING RESULTS

Strategy	Episodes	Win Rate	Loss Rate	Draw Rate
Point count	1,00,000	32.42%	61.44%	6.14%
Point count	15,00,000	40.14%	50.89%	8.97%

in performance is observed. While the win rate remained relatively similar (40.14% vs. 40.10%), the overall.

B. Results and Discussion

The performance of both Q-learning agents was evaluated across three Blackjack rule variations: Standard Rules, Dealer Hits Soft 17, and Double After Split Allowed.

The Basic Strategy Agent consistently yielded a negative Expected Value per hand across all rule sets (Table IV). The EV ranged from -0.0908 (Standard Rules) to -0.0969 (H17), indicating a house edge consistent with theoretical expectations for basic strategy, albeit slightly higher than the commonly cited minimal edge. Rule variations had a marginal impact on profitability, with H17 leading to a slightly more negative EV.

TABLE IV
RULE VARIATIONS RESULTS WITH BASIC STRATEGY

Rule Variation	Win %	Loss %	Draw %	EV
Standard Rules	40.18	50.41	9.41	-0.0908
Dealer Hits Soft 17	40.18	50.86	8.95	-0.0969
DAS	40.04	50.52	9.44	-0.0944

Conversely, the Point Count Agent consistently exhibited a more significantly negative EV (ranging from -0.1632 under H17 to -0.1745 under DAS rules) across all variations compared to the Basic Strategy Agent (Table V). This outcome, counterintuitive in its benefits of card counting, reiterates that the direct reward scaling by bet amount within the Q-learning update likely misguided the agent to minimize weighted losses rather than optimize true unit expected value. While rule variations subtly altered the specific EV, the fundamental issue of the agent's learning signal in the presence of dynamic betting remained the dominant factor, leading to sub-optimal performance for the Point Count agent in all tested environments.

TABLE V
RULE VARIATIONS RESULTS WITH COMPLETE POINT COUNT

Rule Variation	Win %	Loss %	Draw %	EV
Standard Rules	40.20	50.83	8.97	-0.1709
Dealer Hits Soft 17	40.29	50.99	8.72	-0.1632
DAS	40.13	50.97	8.90	-0.1745

C. Double Q-Learning with Point Counting

The Double Q-Learning Point Count Agent was trained for 1,500,000 episodes, demonstrating a more stable convergence of Expected Value (EV) compared to the single Q-Learning variant (Table VII). Initial high negative EV values rapidly decreased, reaching -0.1159 by the end of training, despite the ϵ value reaching its minimum of 0.01 at around 600,000 episodes (Table VI). This improvement in EV (-0.1159 for Double Q-Learning vs. -0.1739 for single Q-Learning on a comparable scale) suggests that decoupling action selection and evaluation effectively mitigates the overestimation bias inherent in standard Q-learning, leading to more accurate value estimates. While still yielding a negative EV, consistent with the house edge and any remaining learning inefficiencies or environmental specifics, Double Q-learning proved more robust in learning a sophisticated strategy integrating card counting and variable betting.

TABLE VI
EPSILON DECAY AND CURRENT EV DURING DOUBLE Q-LEARNING POINT COUNT TRAINING

Episode	Epsilon	Current EV
100,000	0.4040	-0.5335
1,000,000	0.0100	-0.1482
1,500,000	0.0100	-0.1159

TABLE VII
DOUBLE Q-LEARNING RESULTS

Strategy	Episodes	Win Rate	Loss Rate	Draw Rate
Double Q-learning	1,00,000	33.20%	60.61%	6.19%
Double Q-learning	15,00,000	41.67%	49.93%	8.39%

D. Winning Percentage Comparison

A direct comparison of the winning percentages among the three implemented Q-learning agents reveals distinct outcomes, as summarized in Table VIII.

TABLE VIII
WINNING PERCENTAGE COMPARISON OF Q-LEARNING STRATEGIES

Strategy	Win %
Basic Q-Learning	40.10
Point Count Q-Learning	40.14
Double Q Point Count	41.67

The Basic Strategy Q-Learning agent achieved a win rate of 40.10%. The Point Count Q-Learning agent showed a marginally higher win rate at 40.14%. Notably, the Double Q-Learning Point Count agent demonstrated the highest winning percentage among the three, reaching 41.67%.

The Double Q-Learning Point Count agent's higher win rate suggests that its ability to mitigate overestimation bias, with a card-counting strategy, led to more effective decision-making and a higher frequency of winning hands compared to both the basic strategy and the single Q-learning point count approach. While the Point Count Q-Learning agent's win rate was only marginally better than the Basic Strategy agent's, its overall profitability (Expected Value) was significantly worse. This indicates that win rate alone does not fully capture the strategic effectiveness, especially when dynamic betting is involved. The Double Q-Learning approach, by addressing the overestimation, allowed for a more substantial improvement in win rate, aligning more closely with the theoretical benefits expected from refined strategies in Blackjack.

V. Conclusion

This paper explored the application of reinforcement learning, especially the Q-learning algorithm, to determine an optimal strategy for playing Blackjack with basic strategy, Complete point count, and enhanced complete point count systems. The Complete point count outperformed the basic strategy by 0.2 winning rate, which shows a little advantage, including the point count. The best-performing agent was the Double Q-learning Point Count agent. While the expected value remains negative due to the house advantage, this approach shows how reinforcement learning can approach real-world decision-making problems under uncertainty. Future work includes function approximation, Monte Carlo learning, and policy gradient methods.

References

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018.

[2] Edward O. Thorp. *Beat the Dealer*. Vintage, New York, 1966.

[3] Hado. Van Hasselt, “Double Q-learning,” in *Advances in Neural Information Processing Systems*, vol. 23, pp. 2654–2662, 2010.

[4] Charles de Granville, *Applying Reinforcement Learning to Blackjack Using Q-Learning* , University of Oklahoma

Appendix A
Acknowledgment

”My sincere thanks go to Prof. Dr. Frank Deinzer. His lessons on ’Reasoning and Decision Making under Uncertainty’ were key to me successfully completing this work and writing this paper.” ”I’m also very thankful to my university, THWS. They provided free access to all the research tools, which allowed me to easily read and cite many important articles and papers