UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

ARTIFICIAL NEURAL NETWORK-ASSISTED HEURISTIC TREE SEARCH IN COMPUTER DOMINION

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

By

CHRIS FENNER
Norman, Oklahoma
2014

ARTIFICIAL NEURAL NETWORK-ASSISTED HEURISTIC TREE SEARCH IN COMPUTER DOMINION

A THESIS APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

BY

To Mom and Dad

Acknowledgements

Though this is one of the first pages in this document, it is, unfortunately, one of the last to be written. The truly unfortunate part about writing an Acknowledgements page after writing a Master's Thesis is that most people who have written a Master's Thesis would rather do almost anything in the world besides write a single additional word. Consequently, the "Ack Page," as the kids on the streets these days call it, is usually a far cry from a proper expression of appreciation for everyone who made a difference in the writings to follow. One can write only so many sentences about "heuristics" and "agents" and "high-dimensional vector spaces" before completely losing the ability to state things in everyday language so people can actually understand them, like "guesses" and "players" and "high-dimensional vector spaces."

I would like to thank my fellow classmates, coworkers, friends, and peers for being so understanding during my work on this thesis, for not losing their patience with me after hearing about "high-dimensional vector spaces" for the 355th time or being offended by my extremely reclusive behavior for the past few months. I appreciate you so much for being so flexible with your schedules and not giving up on me even after my weeklong stints working alone in my dorm room with naught but a bag of jelly beans for conversation. After I turn in the final copy of this paper, when it's too late to make any last-minute changes in my Master's Thesis, I hope it's not too late to make last-minute memories in our friendships before graduation hits us and we move away to places near and far from Norman.

I would also like to express my sincere appreciation for the professors with whom it has been my great pleasure to work, learn, and discover these past five years. Each of you have such unique teaching styles with which you have marked my mind for the foreseeable future. Each of you have given me so many “aha’s” about so many topics. I will always be a learner, but it will be because of you if I ever achieve success as a teacher someday. My thanks go out to all the Computer Science and Electrical & Computer Engineering faculty, but I would like to expressly thank Drs. Hougen, McGovern, and Lakshmivarahan for their contribution to this thesis as my committee members during what is always (even when you’re on sabbatical, Dr. McGovern!) a hectic time of the semester.

This paper and the work it represents could absolutely not exist without the gracious sponsorship of my parents. Thank you so much for supporting me throughout my education, both as homeschooling parents and longsuffering financiers of my university education. I would like to thank my whole family for being so understanding of my extremely busy schedule and lack of availability to come home on many weekends. I have missed you too, and I hope to be much less aloof to you in the years to come.

Finally, I must acknowledge God, who fashioned my own neural networks and gave me agency to do satisfying work during the brief time I have to make a difference to this planet in the years to come. Every day I see in my studies traces of the power and wisdom of the Creator who marked me with the desire to create beautiful, intelligent, and magical things out of glorified sand, just as He does.

Contents

List of Tables

List of Figures

Abstract

A recent trend among strategy games is the “deckbuilding” game, in which players play cards from a personal deck in order to strategically modify the deck itself in some way. The end goal is to achieve the greatest score at the end of the game. Scoring in these games is typically represented by special cards within the decks themselves or tokens held separately from the deck. Dominion is a popular deckbuilding game published by Rio Grande games. Similar games include Thunderstone from Alderac Entertainment Group and Ascension from Stone Blade Entertainment.

We study games in order to better understand similar real-life problems. The victory condition of deckbuilding games is typically not a single goal state but a whole set of states in which the player’s score is greater than all of the opponents’. Because of this, deckbuilding games represent a type of optimization decision problem. Because deckbuilding games are also stochastic decision problems in which changes to the deck affect the efficacy of future hands, they serve as a good example of the type of optimization decision problem whose decision tree is too complex to explore exhaustively.

In searching the decision tree for moves in Dominion, we employ branch-and-bound heuristic tree search to prune away portions of the tree which are wasteful to explore and instead focus on more promising moves. We train a feed-forward artificial neural network on several heuristic functions of the player’s deck and the game state. This network can then inform the heuristic search algorithm to find competitive moves and win more than 60% of 4-player games against simple Dominion strategies and maintain a plurality of victories even when playing expanded versions of the game with new cards which the agent has never seen.

CHAPTER 1

Introduction

A number of problems studied by computer scientists today can be described as "mixture problems," multivariate problems in which the objective is to find an optimal mixture of elements. A simple example of a mixture problem is developing a recipe for baking bread. Different proportions of flour, water, salt, and yeast yield various types and qualities of bread. In finance, a portfolio consisting of a mixture of instruments in certain amounts will vary from from another portfolio with different proportions.

Decision problems can be mixture problems as well. Consider the problem of deciding how to reinvest dividends from a stock portfolio. Starting from state S, which can be represented as a particular formula in the mixture space, each action A (an example of an action in this space would be purchasing another type of stock) will yield a state S', another formula in the mixture space.

In this thesis, we explore applications of computation intelligence (in particular, the marriage of simple artificial neural network function approximation and branch-and-bound tree search) to the particular mixture-based decision problem of Dominion. For a detailed discussion of the rules of Dominion, proceed to Chapter 2 on page 4. It is our hope that our results in this project translate well to other mixture decision domains, such as stock portfolio selection.

1.1 Deckbuilding Strategy Games

A deckbuilding strategy game is a prime example of a mixture decision problem. In a deckbuilding game, each player is dealt hands from a deck of cards that belong to him or her. In playing these hands, each player may be presented with opportunities to add cards to (or remove cards from) his or her deck to be used (or not used, as the case may be) in future hands.

Deckbuilding games are mixture decision problems because they deal with decks of cards. For example, a player considering adding a certain card to their deck must consider how that card will interact with their deck as a whole, not just how good that card is on its own. Additionally, the randomness introduced by shuffling further increases the difficulty of making decisions in deckbuilding strategy games.

1.2 Objectives

Developing a competent AI for a complex game is an undertaking of great compromise. Many factors must be weighed when considering which AI techniques to apply to a given problem. These are the objectives which we seek to meet with this particular project:

1. Feasibility

   Practical limits on time and space ought to be observed when designing an agent to play a game enjoyed by humans. Ideally, the finished agent could be used by humans for solo play and practice. Thus, the agent designed here should be as efficient as possible with its time and storage resources. Provincial [9] is described as taking minutes of training time before games can begin. After training, it is very fast. Fynbo and Nellemann [10] do not describe the runtime of their neural network. One of the goals for this agent is for it to be both time- and space-efficient.

   The agent should make decisions quickly enough so that a human could play against it without getting bored. Ideally, the agent should make decisions in less than ten seconds on average on a standard desktop computer.

2. Performance

   The agent should be able to perform competitively with the simple strategy players (described in Chapter 2.4 on page 11).

3. Performance Under Constraint

   The agent should be able to apply partial knowledge of Dominion cards it has seen and continue to perform with the simple strategy players.

4. Extensibility

   One of the distinguishing features of Dominion is how much it has grown since it was first published. Since the original release of Dominion in 2008, eight additional Dominion sets have been added: Intrigue and Seaside in 2009, Alchemy and Prosperity in 2010, Cornucopia and Hinterlands in 2011, Dark Ages in 2012, and Guilds in 2013. The original set had only 25 kinds of Kingdom cards, but with all the sets combined there are now 205. Additionally, each expansion added new mechanics which added even more variety to the game. Thus, when implementing an AI for Dominion, it is worthwhile to consider extensibility. Fischer's agent [9] appears to be arbitrarily extensible, given training time to learn a buy strategy for the new cards. Fynbo and Nellemann's agent [10], however, is not extensible, as they even noted in their paper. The agent discussed here, by contrast, is designed to recognize familiar components in unfamiliar cards and work "out-of-the-box" to play with new cards as soon as it is introduced to them.

   The agent should be able to apply its knowledge of Dominion to play games using Kingdom cards it has never seen and still perform competitively with the heuristics.

CHAPTER 2

Dominion

2.1 Overview

Dominion [20] is a deckbuilding strategy game published by Rio Grande games in 2008, winning the prestigious 2009 Spiel des Jahres and Deutscher Spiele Preis awards [1]. In Dominion, all players start with the same initial deck of ten cards: three Estates and seven Coppers. Estates are Victory cards, which provide no functional purpose during the game but award points at the end of the game. Coppers are Treasure cards, which can be played during a turn for Coins, that can be spent purchasing additional cards. By means of these cards, more cards may be purchased which may be played for different (often better) effects. At the end of the game, all players total up their points from Victory cards and the player with the most points wins. The game ends when the Province pile (Province is the best Victory card in the original game) runs out, or when three other piles run out.

For brevity's sake, the above descriptions of these rules have been generalized and simplified. Under certain circumstances, almost all of the rules discussed above can change. These generalizations are employed to provide a beginning understanding of the game in order to follow this thesis. A more thorough discussion of the game is provided below.

One of the interesting features of Dominion is that the available cards for purchase are determined randomly at the start of the game. As of March 2014, there are 202 total different "Kingdom" cards available among all nine boxed game sets. In addition, there are three special promotional cards which were released independently, for a total of 205 Kingdom cards. When setting up a game of Dominion, only ten cards are chosen (usually at random) as the "Kingdom" for that particular game. Ten copies of each Kingdom card make up the Supply. Only cards in the Supply may be purchased during the game. Because of this setup dynamic, virtually every game of Dominion will be unique, since the cards available determine the decisions each player will be able to make. The ten-card Kingdom feature makes Dominion an interesting problem in computational intelligence. If each game had the same available cards, it could be feasible to "solve" Dominion and declare a certain buy strategy to be "the best." Since there are $\binom{205}{10} = 2.89 * 10^{16}$ different possible game setups, this is a fairly impractical approach to developing a capable AI agent. Every random setup in Dominion is practically unique, so a good AI for Dominion must be flexible and able to apply its knowledge effectively to unfamiliar situations.

Once the ten Kingdom cards are chosen and each player is dealt his or her starting deck (7 Coppers, 3 Estates except in some games with the Dark Ages expansion), each player shuffles his or her deck and takes five cards in hand. After playing these cards (which themselves may cause modifications to the deck), the player has the option of buying one (or more, if they played a card that grants additional Buys) additional card(s) from the Kingdom. The played cards, unplayed hand cards (recall that Victory cards cannot be played – except for certain types), and gained card(s) are all placed into the player's discard pile, to be shuffled and made into a new deck after the deck has run out.

It is important to clarify that Treasure cards, when played, are not "spent" as money, but *played* for money. Played cards are recycled into the new deck to be used again in future turns.

Every player begins their turn with one Action and one Buy. Playing Action cards (see below) uses up an Action, but some Action cards give additional Actions when played. If no more Actions are available, no more Action cards can be played.

Additional Buys may also be granted by certain Action cards. If a player has multiple Buys, they may purchase multiple cards totaling up to the amount of money available in the Buy Phase at the end of the turn.

Dominion may be briefly analyzed from a game-theoretic point of view. It is a non-zero-sum game: each player's point gains are not necessarily correlated to point losses for the other players. It is a non-cooperative game: players may affect other players either adversely or beneficially, but there are no teams in Dominion and players either win or lose alone. Assuming a great deal of collusions between the players, and taking advantage of the fact that in Dominion, all players "rejoice in their shared victory," players could conceivably choose to try to end the game without gaining or losing any points, thus causing victory for all players. Indeed, in a learning system only learning by self-play, an equilibrium at this policy could be expected.

The decision of whether or not to purchase Attack cards during the game can be likened to the classic Hawk-Dove game [6]. Certain Attack cards, such as the Possession, which allows a player to play another player's next hand for them, are considered to be so powerful that their inclusion in a Dominion Kingdom can result in verbal bargaining to try to keep all players from buying any of them. Each player has a choice, then, of whether to play Hawk and buy strong Attack cards, or to play Dove and play peacefully with the rest of the players. There is an additional level of complexity to Attack Card game theory: some Attack cards (like the Possession) only affect the player to the left of the attacking player, while others (such as the Witch, which gives each other player a Curse Card) adversely affect all other players equally. Naturally, in two-player games these are equivalent, but in games with three or more players, this distinction is important.

2.2 Cards

There are seven basic cards available for purchase in every game of Dominion: [1]

Table 2.1: Dominion cards

Card Name	Price	Type	Effect
Copper	0	Treasure	+1 coin
Silver	3	Treasure	+2 coins
Gold	6	Treasure	+3 coins
Estate	2	Victory	1 point
Duchy	5	Victory	3 points
Province	8	Victory	6 points
Curse	0	Curse	-1 point

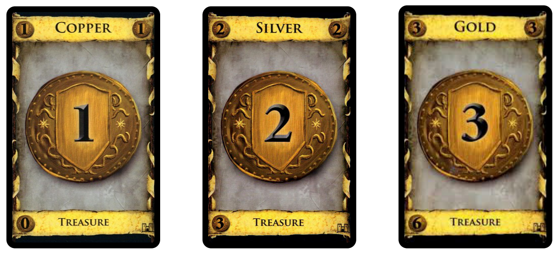


Figure 2.1: Dominion Base Treasure cards

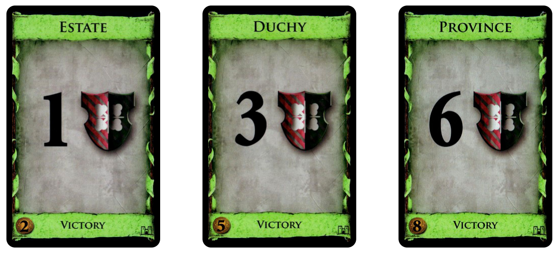


Figure 2.2: Dominion Base Victory cards

[1]Card images copyright ©Rio Grande Games. Used by permission.

Figure 2.3: Dominion Base Curse Card

For a complete list and strategic discussion of Dominion cards used in this paper, see Appendix A on page 53.

2.2.1 Card Types

The three types of cards we have shown so far are the most basic cards in Dominion and are available in every game.

Treasure cards can be played for coins, with which a player may buy other cards. Treasure cards are discarded into the player's personal discard pile when played, to be re-shuffled into the player's deck when the player's deck runs out.

Victory cards are worth points at the end of the game, and typically do not provide any other benefit. Some special cards in later expansions are both Victory cards and another type of card, so they may be played for immediate benefits during the game but also give points at the end of the game.

Curse cards are worth -1 point each, and are given to players by certain attack cards such as the Witch.

There are several other types of cards available, depending on the particular Kingdom being played:

Action cards can be played using an available Action and give a variety of effects (see below).

Attack cards are Action cards which harm other players.

Reaction cards may be played in reaction to certain game events (like another player playing an Attack card).

Duration cards (Seaside expansion) are Action cards which give additional effects on the turn after they are played. They are not discarded from play until the next turn.

Hybrid cards have multiple types. For example, there is the Nobles card which is both a Victory card (worth 2 points) and an Action card, playable for either +3 cards or +2 actions.

2.2.2 Card Effects

Action and treasure cards produce a variety of effects when played. Here are a few of the most common ones:

+Coins effects give the player Coins to spend in their Buy phase. These are not physical coins, but credits to be used during the turn. These coins do not roll over into the next turn.

+Actions effects give the player additional Actions to use by playing more Action cards, if they have any more.

+Buys effects give the player additional Buys with which to purchase multiple cards. The total price of the cards purchased may not exceed the total amount of money available.

+cards effects allow the player to draw additional cards from their own deck.

Gain a card effects let the player gain a card from the Supply, costing up to some specified amount.

Discard a card effects cause players to discard cards from their hands into their discard piles.

Trash a card effects allow players to quasi-permanently remove a card from the game. This effect is quasi-permanent because some cards allow players to interact with the Trash Pile, where Trashed cards go.

2.3 A Sample Turn in Dominion

Suppose Alice begins her turn with these cards in hand:

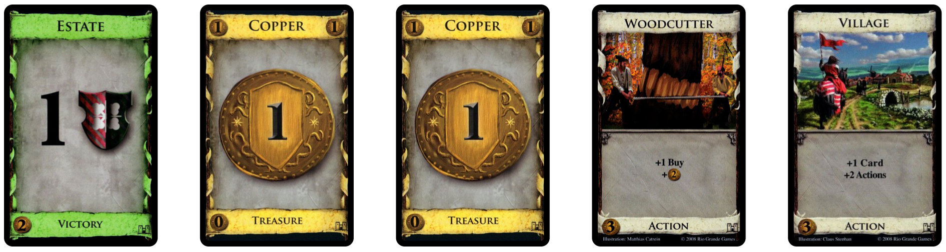


Figure 2.4: Alice's hand at the start of her turn

Alice has two Action cards in her hand, but only one Action. If she plays the Woodcutter right now, she will be out of Actions and unable to play the Village. If she plays the Village, however, she will receive additional Actions with which to play her Woodcutter. She plays the Village for +1 Card, +2 Actions, drawing a Smithy:

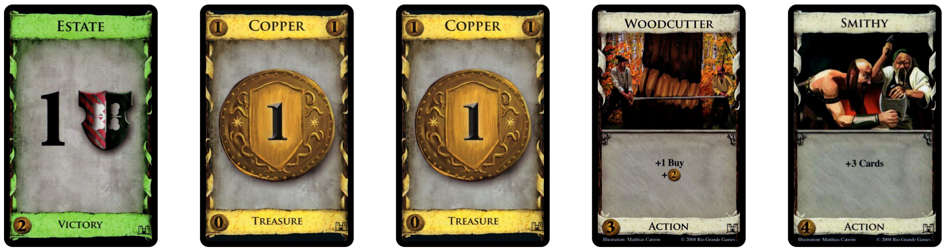


Figure 2.5: Alice's hand after playing the Village and drawing the Smithy

Now Alice has two Actions and plays the Woodcutter and the Smithy, gaining +2

Coins, +1 Buy, and drawing three more Coppers:

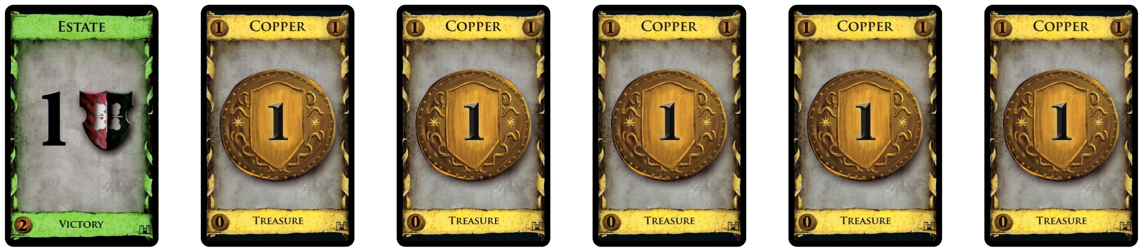


Figure 2.6: Alice's hand after playing the Woodcutter and the Smithy

She plays her five Coppers for +5 more Coins, a total of 7 Coins. Next, she buys a Gold for 6 Coins, ending her turn. All the cards shown above are then discarded into the discard pile.

2.4 Simple Dominion Strategies

2.4.1 Short-Term Strategy

Optimizing Hand Play

Though the focus of this thesis is on the long-term, high-level (inter-turn) strategy of choosing cards to gain and trash, good short-term (intra-turn) strategy is necessary for adequate agent performance. Which Action cards to play and which cards to discard (not trash) during the game constitute short-term strategy. A fairly effective short-term strategy for playing Action cards is as follows:

1. If one can play a Throne Room or a King's Court, play it. Throne Room and King's Court are Action cards that allow a player to choose another Action card and play it multiple times (twice for Throne Room, thrice for King's Court).

2. If one card gives additional Actions, play it. This increases the chances that all of the Action cards in the hand may be played.

3. Play the most expensive card remaining. This will usually give priority to the best card, although in certain situations a cheaper card may be better. In most cases, though, this will not be an issue.

2.4.2 Long-Term Strategy

In order to measure the efficacy of the long-term strategies created by the agents discussed later in this paper, several heuristic agents were created that follow common simple strategies employed by beginners to the game.

Highest-Value Card

Highest-Value Card is perhaps universally used by new players in their first few games. It is straightforward: attempt to gain the most expensive cards possible and trash the least expensive cards possible. This system breaks ties between cards of the same price randomly. This strategy usually results in a very incohesive deck, filled with random Action cards that may not necessarily work well together or even all be played if drawn together. This strategy is the worst of the heuristic strategies tested.

Big Money

Big Money has the best "bang for the buck" of any of the heuristic strategies. It is extremely simple but wins with surprising frequency. Players playing the Big Money strategy *only* purchase Treasure cards, purchasing the most valuable Treasure cards they can afford. This eliminates the trouble of deciding which Action card to play and prevents situations in which not all of the Action cards can be played.

Best Fifth Card

Best Fifth Card is much more complex. When considering gaining a new card, it simulates several sample hands populated with the card being considered and four randomly-chosen cards from the deck. cards that lead to more money at the end of the hand are chosen. This leads to a more cohesive deck than Highest-Value Card. This strategy is the overall best of the three heuristic strategies employed.

CHAPTER 3

Related Work and New Contributions

3.1 AI for Board Games

Board games can be categorized into two major types: deterministic games, like Go, Chess, Checkers, and Tic-Tac-Toe, and nondeterministic games, like Poker, Backgammon, Risk, and Dominion. Elements of randomness in the latter games, like card shuffling or dice rolls, create uncertainty about which future game states will result from current moves. In deterministic two-player games in which each player has full knowledge of the game state, like Go and Chess, Minimax tree search [4] may be employed to find optimal actions from any given game state by simulating the anticipated actions of two players, “Max” and “Min,” with each playing under the assumption that the other plays optimally. In nondeterministic games, as well as deterministic ones which have game trees that are too large to fully search with Minimax, one of the most common approaches is to use Monte-Carlo-based rollout algorithms with UCT [14] to search the game tree. UCT, or Upper Confidence Bounding for Trees, orders game tree nodes for Monte-Carlo rollout expansion based on past rewards and the number of times each node has already been explored. In this way, an AI agent using UCT can efficiently navigate large game trees or game trees with high stochasticity for Monte-Carlo search.

3.2 Monte-Carlo in Games

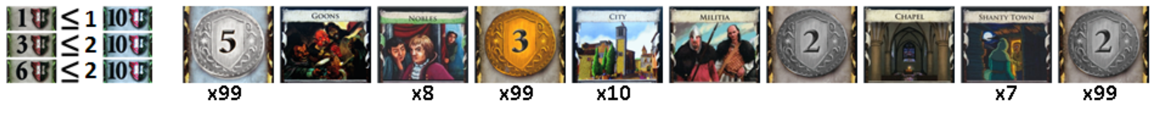
Monte-Carlo search is commonly employed to create AI agents for both deterministic and stochastic games. It has been heavily applied in the deterministic game of Go [5] [16] [8] [24] as well as in stochastic card games like Poker [2], Skat [18], and Bridge [11]. In both the large-branching-factor deterministic cases like Go, and the stochastic-game cases like the card games mentioned, Monte-Carlo search provides a means to effectively explore game trees without searching exhaustively.

3.3 Dominion AI

Veness, Lanctot, and Bowling at the University of Alberta demonstrate methods of variance reduction in UCT search in Dominion [22] to find the optimal bias value C for search in solitaire Dominion games. Though their results do not show performance data for multiplayer games, they demonstrate that their variance-reduction techniques give improvements to UCT search roughly equivalent to performing 25-40% more simulations.

Provincial [9] is another Dominion AI project by Matthew Fischer of Stanford University. Provincial uses an evolutionary algorithm to "train" on a specific Kingdom (game setup of 10 Dominion cards) to develop a comprehensive buy strategy. The buy strategies developed by Provincial are essentially lists of Dominion cards, each with a maximum number.

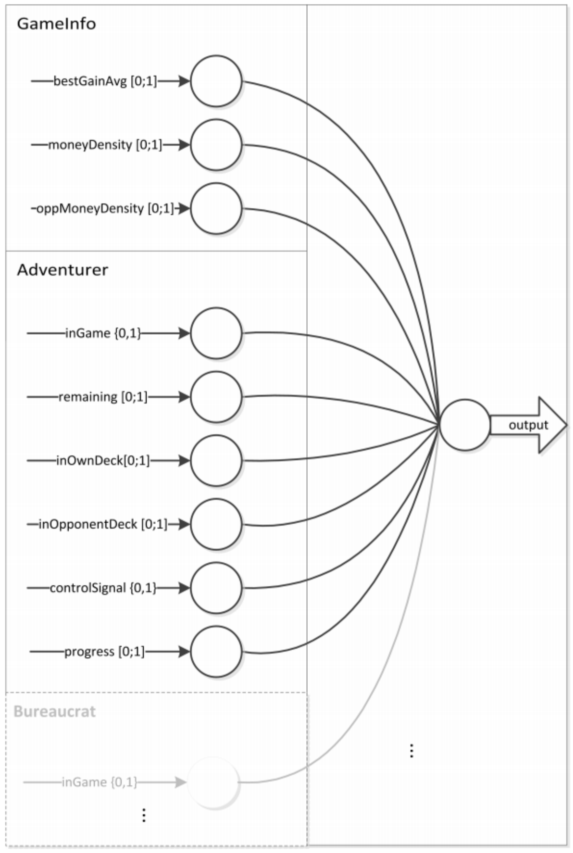
Figure 3.1: Provincial AI buy strategy [9]



Along with the priority list of cards (and maximum number to buy), these buy menus also include rules for small Victory card purchases, shown on the left-hand side of the figure. The shown buy strategy will buy Estates (the one-point Victory cards) when there are one or fewer Colonies left. (The Colony is to the Province what the Platinum is to the Gold). It will buy Duchies and Provinces (three and six points, respectively) when there are two or fewer Colonies left. Otherwise, the agent buys the highest-priority card in the list that it can afford unless it already has the given number of that card in its deck. Provincial is thus able to evolve capable buy strategies for a given Kingdom by simulation before the game begins. Provincial is implemented for the first four of the nine currently-published sets of Dominion.

Fynbo and Nellemann of Copenhagen [10] take a more theoretical-based evolutionary computational intelligence approach. They build a NEAT-based [19] network evolution environment to make all of the game decisions. This method is effective at learning very good strategies for all aspects of Dominion, but the authors conclude that it would be infeasible with current hardware to try to extend the system, which had 189 inputs for just the original set, to learn further expansions. Each additional card added to the game would require an even more complex neuron structure to be evolved.

Figure 3.2: Fynbo and Nellemann Neural Network to Evaluate Cards [10]



Fynbo and Nellemann's neural network takes several game state inputs and several additional inputs per card and produces an output indicating how desirable each card is. It uses these values when making purchase decisions throughout the game. As a network topology optimization, they use control signals for each card to "select" a card for evaluation by the network on the one output, rather than having an output for each possible card that could be bought.

3.4 Neural Networks in Markov Decision Problems

Eck and Wezel of Erasmus University Rotterdam demonstrate the application of artificial neural networks in the deterministic two-player game of Othello by training networks to estimate Q-values for actions from board states. In Q-learning, a Q-value represents the reward for executing a particular action and following a fixed policy π thereafter [23]. [21] Their networks accepted the 8-by-8 grid of board spaces (which can be each occupied by black or white, or empty) as inputs to a neural network and output Q-values for each move. They tested both single-network learners (with 64 inputs and outputs, one for each board space) and multiple-network learners (64 separate 64-input networks with one output, one network for each output value) and found that the single-network learners performed as well as the multi-network ones, but learned the Q-functions more quickly. Cai and Ferrari also applied Q-learning in the game of Clue to create a neural network to move around the board and solve the mystery [3]

3.5 Neural Networks for Heuristic Search

Greer [12] demonstrates the use of a neural network trained on chessmap (piece control relationships of board locations) representations of random game states in Chess to evaluate individual move influence when ordering moves for game tree search. The approach using the neural network reduced the overall number of nodes explored in search but did not reduce the overall computation time. It was, however, successful in simulating the type of reasoning which humans often employ when evaluating the importance of regions of the chess board.

Kocsis et al [13] similarly employ an artificial neural network to order search nodes in alpha-beta search. Like Greer, Kocsis et al. do not directly evaluate nodes using the neural network, but use the network to increase the efficiency of the existing alpha-beta search algorithm.

3.6 New Contributions

Monte-Carlo search with UCT is commonly applied to these sorts of problems. Instead of using Monte-Carlo search in Dominion, which would require many rollout simulations for each node explored, we employ artificial neural networks to evaluate node values in a manner more similar to Q-learning, though instead of using Q-values for nodes directly to deterimine actions, we use them to bound node values in branch-and-bound tree search.

The work done by Fischer, Fynbo et al., and Veness et al. is very exciting and performs well in the Dominion domain. This paper seeks to continue to broaden the area of research in the area of Dominion by pursuing the goal of creating a generic agent that is flexible to the environment of available cards but still competitive with common strategies.

Eck and Wezel's approach to Othello as a deterministic decision problem using artificial neural networks to evaluate Q-functions gives us reason to hope that neural networks can be employed to evaluate search tree nodes in stochastic decision problems as well.

In the area of applying artificial neural networks to search algorithms, in contrast to the work of Greer and Kocsis et al., we demonstrate a neural network for bounding move rewards, rather than merely ordering search nodes to be evaluated by some other means. Instead of using Q-learning to estimate node values directly, as Eck and Wezel and Cai and Ferrari do, we use a similar method (action selection and policy rollout) to estimate node minimum value for network training, but additionally use other knowledge about the environment to estimate node maximum value based on possible movement around the deck composition space of Dominion. This has the potential to greatly improve search performance in environments in which it is very expensive to calculate these bounds manually.

CHAPTER 4

Modeling Dominion

4.1 Focusing on Long-Term Strategy

As Fischer, Fynbo, and Nellemann have all pointed out [9] [10], it is fairly effective to divorce the two major aspects of Dominion play into two parts: short-term hand strategy and long-term buy strategy. Short-term strategy is relatively simple: play cards in an effective order. Long-term strategy is more complex and far-reaching: every card you buy, gain, or trash changes the composition of your deck and the overall probability of drawing certain cards at certain times. It is this long-term strategy that both of the projects previously mentioned have focused on. Simple heuristics can be developed to handle the short-term strategy with ease, and anything a heuristic could not be developed for, a Monte-Carlo (simulating) approach could handle with ease, given the small tree height and branching factor of short-term game decisions such as “Choose a card to discard” or “Choose a Treasure card to play first.”

4.2 Representing Decks as Vectors

Much of what makes a particular buy strategy viable or not depends on the current contents of the player’s deck. If a player already has so many +Coin cards that they are consistently getting more than 8 Coins per round, additional +Coin cards cannot improve their performance until cards that grant additional Buys are obtained, since a player is limited to buying a number of cards equal to the number of Buys they have, regardless of how many Coins they have. Because of phenomena like this, we have elected to represent the player's game state by creating a vector out of its deck composition.

There are a number of ways in which we could create these vectors. One simple way would be to count up the number of cards of each type. Here is an example deck:

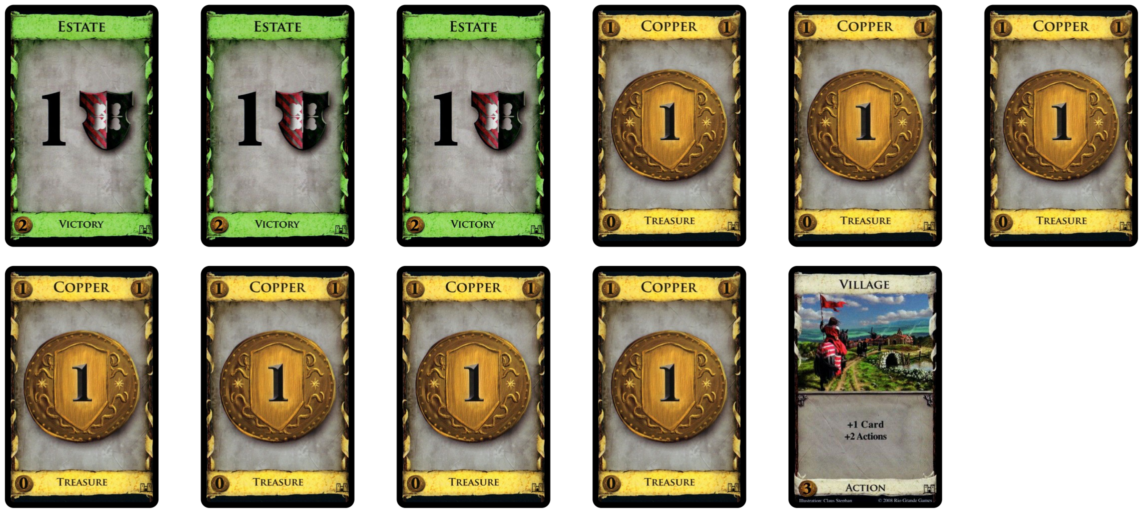


Figure 4.1: Sample Deck for Vectorization

There are 32 different cards in the game of Dominion, including the 7 base cards, so deck vectors in this scheme would have 32 dimensions. This deck would have a 3 in the Estate dimension, a 7 in the Copper dimension, and a 1 in the Village dimension.

But consider a deck with twice as many of each card. It would have a 6 in the Estate dimension, a 14 in the Copper dimension, and a 2 in the Village dimension. It would essentially look twice as good as the first hand, but consider how it would play. The player still draws 5 cards and has the same expected number of each card for a given random hand. Thus, when "vectorizing" Dominion decks, we normalize the vector by dividing by the total number of cards in the deck and focus on *deck compositions*. Both these decks have a composition of 3/11 in the Estate category, 7/11 in the Copper category, and 1/11 in the Village category.

With all expansions to Dominion combined, this scheme would have 205 total dimensions in the deck vector. Can we do better than that, and can we improve the agent's performance in cases when unfamiliar cards are in play? We believe so.

Consider the three Treasure cards in the base game:

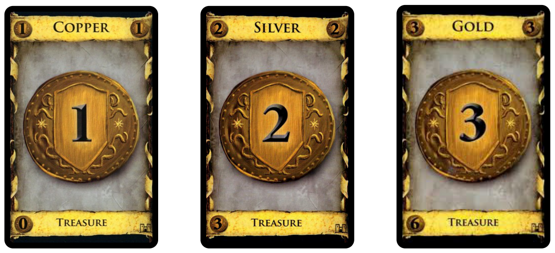


Figure 4.2: Dominion base Treasure cards

When introduced to the additional Treasure card Platinum in the Prosperity expansion, our experience with Copper, Silver, and Gold gives us an intuition as to how much we would like to buy a Platinum:

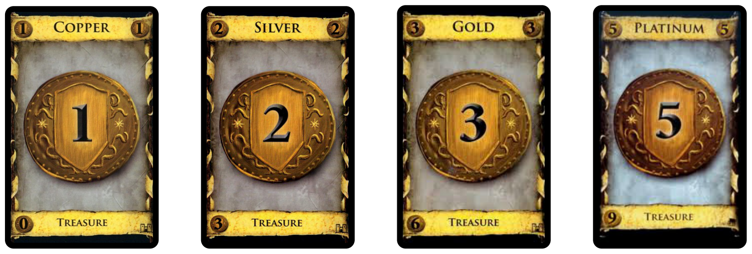


Figure 4.3: Dominion base Treasure cards, plus Platinum

Every player who has played a few games of Dominion immediately grasps the power of Platinum when introduced to it for the first time. This is not because of the identity of the card, but the effects of the card.

4.3 Representing Effects as Dimensions

We have thus chosen to represent Dominion deck vectors as compositions of effects, and let the particular pairings of effects in individual cards take a back seat to the more important overall deck composition. Consequently, each of the basic Treasure cards discussed above would have different values in the “+Coins” dimensions: 1, 2, 3, and 5 for Copper, Silver, Gold, and Platinum, respectively. The original hand, instead of having values in dimensions for “Copper,” “Estate,” and so on, would have a value of 6/11 in the “+Coins” category, 1/11 in the “+cards” category, and 2/11 in the “+Actions” category. It should be noted that we have not included a “+Points” effect – this is because Victory cards are only counted at the end of the game, and do not give points *every* turn. In future expansions, there are cards which give “Victory Point Tokens”, and the effect of gaining these tokens is something that would be included in a deck effect vector. There are 205 different Kingdom cards in Dominion plus a handful of special non-Kingdom cards (that may be in the supply separately, like Coppers and Silvers, or can only be gained as an effect of another card, like the Prizes from the Tournament). There are approximately just as many different card effects: future expansions tend to include cards with several of the “simple” effects from the original Dominion set (like plus Actions or plus cards) and up to one “complex” effect that is unique to the card. Representing decks as vectors in the vector space according to card effects, then, allows our agent to perform well even with unfamiliar cards, provided they have at least some of the same effects that we have already seen, without increasing the complexity of the deck vectors compared to just counting individual cards. In order to distinguish between Treasure cards (which do not use up an action when played) and Action cards (which do, and cannot be played when actions run out), we add a synthetic “-1 Action” effect to all the Action cards in the game for the purposes of our vectorizer.

For a listing of all the effects in Dominion, along with the frequencies of those effects, see Table 6.3 on page 41.

4.4 Three Extra Dimensions

In building and play-testing the agent to play Dominion, we found it helpful to the heuristic search algorithms to provide three additional dimensions to represent game state, on top of player deck composition: number of players, number of cards in the player's deck, and estimated remaining number of turns in the game. Using these three extra factors, our algorithms can optimize the search tree effectively. We will define the game vector as being these three parameters plus the deck vector.

4.5 Heuristic Functions of Game Vectors

For our algorithm, we are interested in four heuristic functions of game vectors:

1. Average available money after playing a hand
2. Average number of buys after playing a hand

   These two heuristics allow us to estimate future buy turns based on anticipated deck composition changes.

3. Average number of points gained after playing out a game

   This value helps us calculate how many points a player would have if they stopped trying to improve their deck and just tried to score points for the rest of the game. This is calculated by simulated rollouts with a simple policy that always seeks to gain the immediate most points possible. If a game decision changes the score, the policy chooses the decision which increases the score the most. The value calculated essentially reports the number of points a player would have after only buying Victory cards for the rest of the game. This serves as our minimum bound for search node evaluation.

4. Average maximum game vector change after playing a hand

   This heuristic helps us in estimating upper bounds for points at the end of the game: knowing that a player can change their deck composition by a certain distance in the vector space allows us to search for local maxima within that radius, giving a best-case scenario for how powerful of a deck can be built by the end of the game.

4.6 Knowledge Representation

With the well-formed idea of a multi-faceted deck performance landscape in a vector space in mind, we must now consider how best to represent that landscape.

The immediate approach we applied to the problem was to constrict a grid of points in an r-dimensional vector space of known values for particular deck compositions. This worked well for very small subsets of the problem, but for very large values of r (such as 27, which was the final vectorization of deck effects and game parameters) this turned out to be unfeasible. Storage requirements for a grid in r dimensions with d points along each edge are on the order of d^r, and interpolation time in r dimensions is also exponential in r. We were able to cut down on both of these problems slightly by using simplexic (the r-dimensional analog of a triangle) grids, operating under assumptions about the sums of certain parameters of the deck vector, and by experimenting with methods of interpolation requiring fewer data points, but these methods led to unsatisfactory accuracy in extrapolating information from sampled data.

To simultaneously solve the triple problems of knowledge space complexity, interpolation time, and interpolation accuracy we use a feedforward neural network (FFNN) with backpropagation [17] to learn the heuristic function landscapes in terms of deck vectors. We use a simple feedforward neural network with a single hidden layer size of r, with output nodes for each of the heuristic functions in terms of the r input components of the deck vector. Because we have access to a Dominion simulator and can generate games of Dominion with varying vectorizations by merely dealing out some random cards, the problem of approximating our heuristic functions at playtime become a simple supervised learning problem.

Neural Network Basics

The following is a simplified high-level explanation of artificial neural networks. For more detailed information, see Engelbrecht's Computational Intelligence [7].

A feedforward neural network (FFNN) is a type of artificial neural network (ANN) used in function approximation, data classification, handwriting recognition, and hosts of other applications within the sphere of machine learning. It consists of a layer of inputs (in our case, the game vector values), zero or more "hidden" layers of neurons which receive values from the input layer, and a layer of output neurons which receive values from the hidden layer(s) and sometimes even the input layer directly.

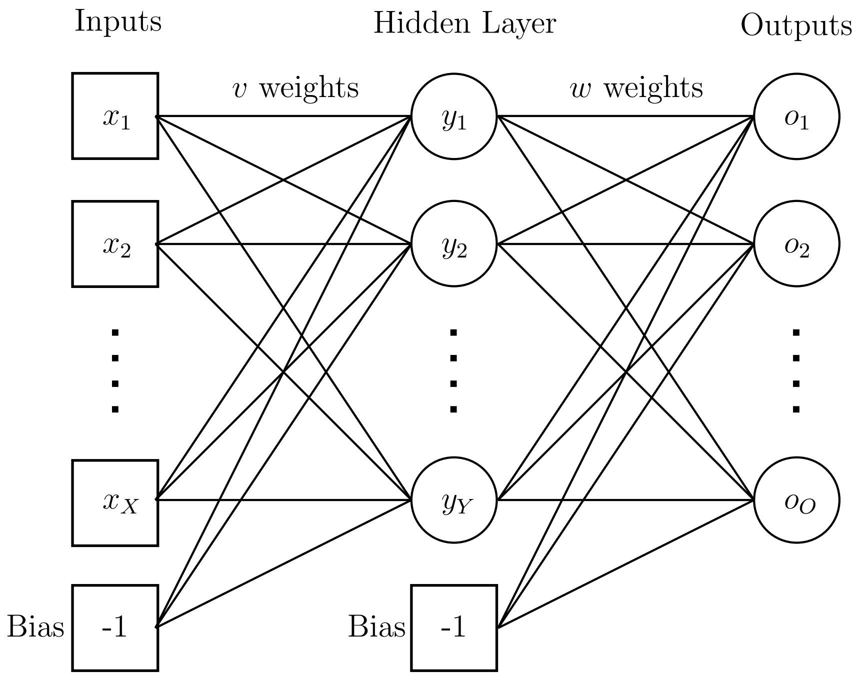


Figure 4.4: Basic Feedforward Artificial Neural Network

An input array of values x_1 to x_X is fed into the network. These values then propagate to each neuron y in the hidden layer. Each pair of hidden-layer neurons and inputs y_i, x_j has a unique weight value v_{ij}. Additionally, there is a "bias" input of -1 which also propagates to the hidden layer. Each neuron in the hidden layer then sums up the values propagating to it from the input layer and passes that value through an activation function to obtain a final output value for that neuron. Then, in the same fashion as before, values from the hidden layer propagate (along with another bias value) to the output layer of neurons to be evaluated a second time and then returned as outputs.

Each neuron in the hidden and output layers has a network function *net*, a set of weights w and an activation function f to process incoming data and feed it forward to the next layer(s) or return as outputs. The network function is responsible for combining the data from the neurons feeding to it: in most cases (including ours) it is a simple summing function. For a neuron receiving input from upstream nodes x_0 to x_I (with weights w_0 to w_I) we have:

$$net_o = \sum_{i=0}^{I} x_i w_i$$

Often neurons have an additional input of -1, with variable weight $w_I + 1$, that serves as an input bias to the neuron's network function.

The *net* function is then fed into the activation function f. The f function takes the value output by the *net* function and returns another output between 0 (or -1) and 1:

$$f(\infty) = 1 \text{ and}$$
$$f(-\infty) = 0 \text{ or}$$
$$f(-\infty) = -1$$

Typically (and again in this particular case) this function takes the form of a sigmoid ("S-shaped" function) such as $\frac{1}{1+e^x}$. It is monotonically increasing, differentiable, and easy to calculate and differentiate.

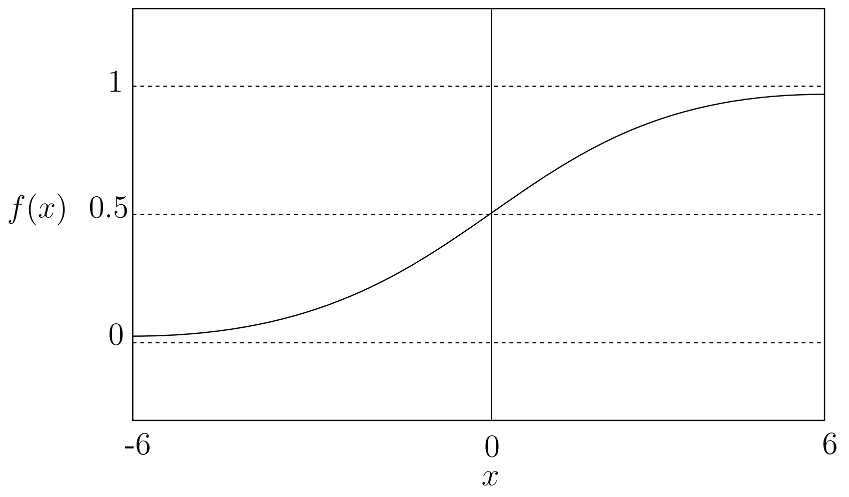


Figure 4.5: Plot of the Sigmoid Function

We then "train" this network by backpropagation [17] on previously-calculated evaluations of various heuristic functions of game vectors (coins at end of turn, buys at end of turn, deck mobility, minimum score at end of game, and maximum score at end of game) in order to evaluate these heuristic functions quickly and accurately at any arbitrary game vector which our agent might find itself in.

Table 4.1: Network Training Configuration

Parameter	Value
Input Layer Size	27
Hidden Layer Size	27
Output Layer Size	4
Fully Connected?	All Input to Hidden, All Hidden to Output
Input Bias for Hidden Layer?	Yes
Input Bias for Output Layer?	Yes
Activation Function	Sigmoid
Learning Rate	0.5
Momentum	0.1
Stopping Condition	Error Stops Decreasing / Time Limit
Training Set Size	40000
Validation Set Size	4000

We define "Error Stops Decreasing" to be when the network's root mean-squared (RMS) error on the validation set increases to three standard deviations above the mean RMS for all rounds of backpropagation. Our training time limit was set at ten minutes of running on a quad-core Intel Ivy Bridge Core i5 desktop processor.

Test data was generated using random hands of Dominion cards. Each card's probability of being selected was proportional to the number of them in a standard 6-player Dominion Supply (10 for most Kingdom cards, 60 for Copper, 40 for Silver, 30 for Gold, 8 or 12 for Estate and Duchy in 2-player and 3-or-more-player games, respectively, 8, 12, 12, 15, or 18 for Province in 2, 3, 4, 5, or 6 players). Each random hand started with 0 to 7 Coppers and 0 to 3 Estates. The rationale for this was that the vast majority of decks seen in Dominion will contain the 3 Estates and 7 Coppers from the start of the game.

CHAPTER 5

AI Agent Description

5.1 Overview

Armed with this neural network approximating these heuristic functions, we developed a search algorithm to find the best move for each situation.

Each time the simulator is prompted to make a game decision, it first considers whether this decision affects its game vector by simulating making the decision. If the decision is not found to be game-vector-affecting, it makes the decision using simple heuristics that were developed when the other agents were developed. For more information on these heuristics, please see Chapter 2.4 on page 11. It should be noted that there is much room for improvement in this agent, particularly in its short-term strategy. The focus of this project is on long-term game strategy, as it has the greatest impact on performance. In future work, short-term strategy improvements could be more beneficial.

If the simulator finds that the decision affects the overall game vector, it searches the tree to find the best overall next move.

5.2 Bounding the Search Tree

We represent the search tree with nodes that represent each turn's action. Because the search tree is built on actual action choices given by the Dominion simulator when it prompts the agent for a choice, the root action in a search tree is any valid deck-changing action in the Dominion simulator. Examples of these actions include:

buying cards, trashing cards, gaining cards in ways other than buying them, and upgrading cards.

The search tree for buy strategies in Dominion is fairly large. Because there are ten cards in the Kingdom, plus the seven base cards (Copper, Silver, Gold, Estate, Duchy, Province, Curse), the branching factor for single-buy buy moves can be as high as 17. There are some specialty base cards that are used only in certain expansions, such as the Province (like a Gold that gives +5 Coins) and the Colony (like a Province that gives +10 points) from the Prosperity expansion or the Potion (a special treasure type for the Alchemy expansion). Realizing that players may have multiple buys to use, we must increase this branching factor greatly. With a particularly powerful deck, a player might be able to purchase (or otherwise gain) three or four cards in their turn. This means the worst-case branching factor is on the order of N^B, where N is the number of cards available and B is the number of buys.

5.2.1 Bridges and Branching

An extreme but concrete example of the branching factor problem in this game comes from a card called Bridge:

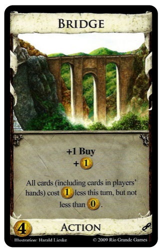

Bridge
+1 Buy
+1
All cards (including cards in players' hands) cost 1 less this turn, but not less than 0.
4
Action

Figure 5.1: Search Tree Branching with Bridge

The Bridge gives players additional Buys *and* reduces the price of all cards by 1 Coin. Players often use the Bridge in combination with certain other cards to score a large number of points in a single turn.

Consider the following hand of Dominion:

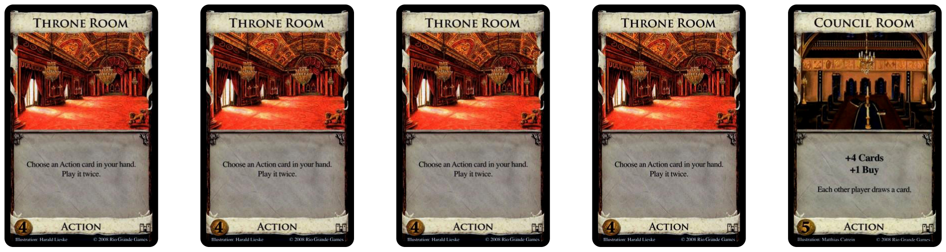


Figure 5.2: Extreme Hand with Bridges

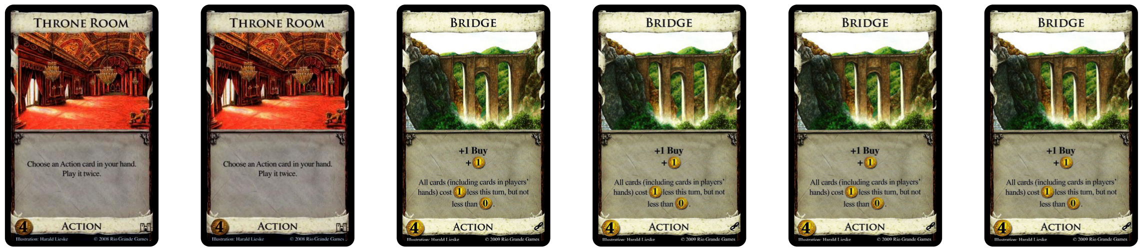

Throne Room
Choose an Action card in your hand. Play it twice.
Action
Throne Room
Bridge
+1 Buy
+1
All cards (including cards in players' hands) cost 1 less this turn, but not less than 0.
Action
Bridge
Bridge
Bridge

Figure 5.3: First Six cards of Deck (Extreme Bridges)

The hand goes as follows: Play Throne Room 1 to play Throne Room 2 twice:

playing Throne Room 3 twice:

playing Throne Room 4 twice:

playing Throne Room 5 twice:

playing Council Room twice to draw 8 cards and gain 2 buys

playing Bridge 1 twice to reduce card cost by 2 Coins and gain 2 buys

playing Bridge 2 twice to reduce card cost by 2 Coins and gain 2 buys

playing Bridge 3 twice to reduce card cost by 2 Coins and gain 2 buys

playing Bridge 4 twice to reduce card cost by 2 Coins and gain 2 buys

The player now has 11 buys (the one buy the player started with at the beginning of their turn, plus 10 extra) and Provinces (which normally cost 8 Coins) are free, so the player buys 11 Provinces. A 4-player game setup has only 12 Provinces, so if there are 11 Provinces left by this point, this is almost certainly a victory for the player employing this strategy. But the true branching factor of this buy move is approximately 17^{11}, since there are 17 cards available for purchase in a normal game of Dominion, all of them are free after this many Bridges, and the player has 11 buys.

This decision can get out of hand very easily, so we must bound it. We use a simple branch-and-bound algorithm to cut down this tree significantly and provide good results even when the tree cannot be completely searched. In branch-and-bound search, the node with the greatest minimum value is kept memorized, and all new nodes encountered with a maximum value less than that node's minimum value are not explored. These nodes can be safely discarded because their value will not exceed the estimate.

5.2.2 Turns Remaining

We estimate the number of turns remaining in the game with Monte Carlo rollouts. We copy the current game state (shuffling each player's decks so as to not accidentally cheat) and simulate the game with heuristic agents to find how much longer it is expected to take. In the future, neural networks could be employed to "learn" game length to a greater degree of accuracy and in less time.

5.2.3 Minimum Node Value

The minimum node value is given by one of the heuristic functions discussed earlier. This function essentially approximates the number of points the player would have if they ceased to buy anything but Victory cards for the rest of the game.

5.2.4 Maximum Node Value

The maximum node value heuristic uses the neural network's "deck composition change" output to estimate how much the deck could change each turn. It then

```
position <- get_current_game_position
best_score <- 0
search_radius <- 0

while (remaining_turns > 0)
      score <- find_local_max_score(position, radius)
      if score > best_score
         best_score = score

      position.remaining_turns--
      radius = radius + find_local_max_deck_change(position, radius)
```

Figure 5.4: Search Node Maximum Value Algorithm

uses a stochastic gradient ascent algorithm (recall that neural network functions are easy to differentiate) to find the maximal-scoring game vector within that distance of the current position and finds the minimum point value at that position. This algorithm quickly finds an upper bound for the best possible sequence of moves which take the player to a high-score-generating state, then buying victory cards for the remaining turns.

CHAPTER 6

Experiment

6.1 Hypotheses

We have four hypotheses for this project that we would like to test by experimentation:

1. Feasibility
2. Performance
3. Performance under Constraint
4. Extensibility

6.1.1 Feasibility

We say that this approach makes the search feasible if the algorithm can use the heuristic function neural network to effectively prune the search space and terminate in a reasonable amount of time. We define "effectively prune" to mean that the algorithm either searches or prunes the entire space (or a substantial portion of it, so that it is highly unlikely to miss a better move than the one it chooses). We would like our agent to spend less than ten seconds, on average, thinking about its moves, so that humans may comfortably play against it.

6.1.2 Performance

We say that this approach has good performance if the algorithm significantly outperforms the simple strategy agents, i.e. it has a significantly greater average score and win rate than all of the simple strategies.

6.1.3 Extensibility

We say that the approach is extensible if the algorithm can utilize its knowledge based on similar cards to make "good" decisions regarding new, unknown cards in new expansions. We define "good" decisions to be ones that cause the agent to continue to perform competitively with the heuristics even in unfamiliar card environments.

6.1.4 Performance under Constraint

We say that the approach performs well under constraint if the algorithm continues to perform competitively even if its knowledge is reduced. We will reduce the knowledge by building heuristic function neural networks that use only a subset of the known 24 effects in the original game of Dominion.

6.2 Results

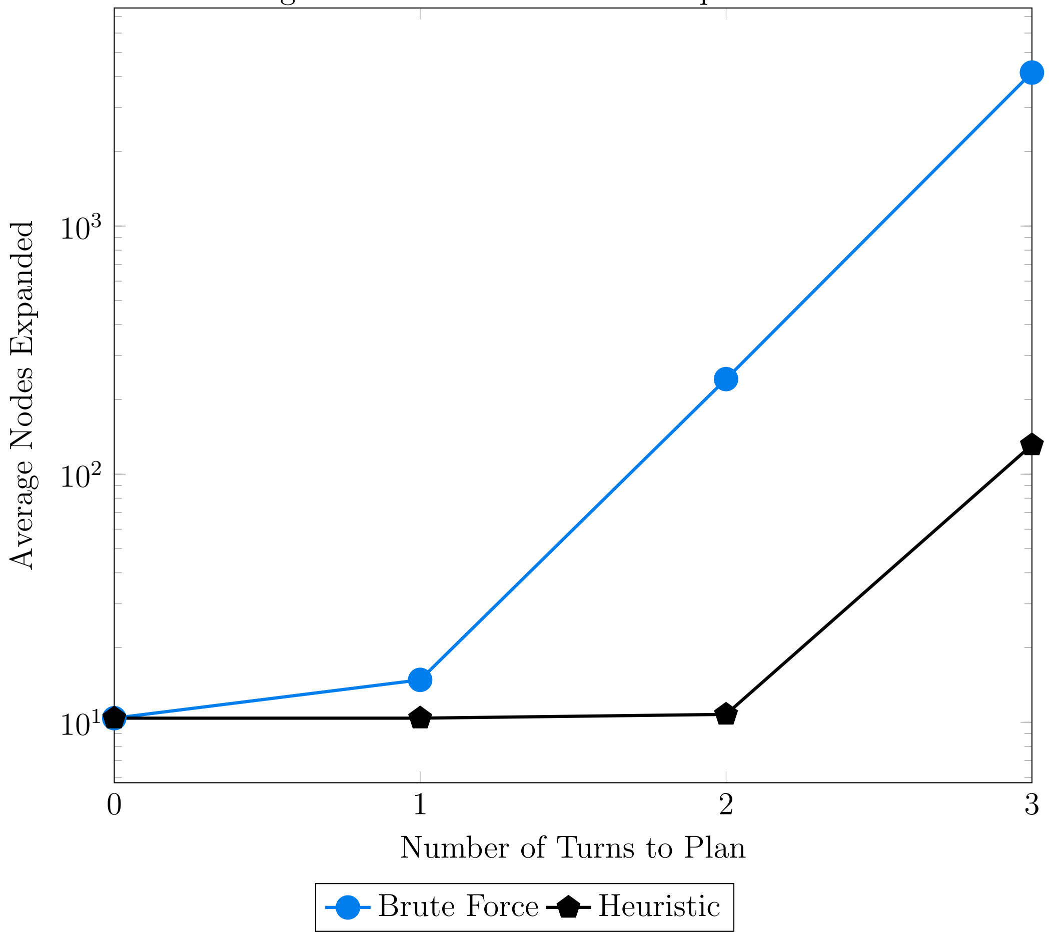
In order to test these hypotheses, we experimentally ran games and sample hands with the planning agent and the simple strategy agents and gathered relevant data to test each hypothesis.

6.2.1 Feasibility

Unfortunately the agent takes a great deal of think time when allowed. While the branching and bounding cut the search tree greatly, still more time was spent thinking than we would have liked. In the following results we have restricted the planning agent to expanding only 20 search tree nodes, but we are quite pleased with the results of even this handicapped planning agent. The resulting agent makes decisions in a few seconds consistently.

For a small number of turns ahead, starting from the default starting deck, we ran both heuristic and brute force searches in the search tree space 100 times and averaged the resulting number of nodes expanded.

Figure 6.1: Number of Nodes Expanded



6.2.2 Performance

100 games were played out between the three heuristic agents and the planning agent. Note that the “Win Rate” values do not sum to 1.0. This is because per official Dominion rules, in the event of a tie (and if both players played the same number of turns), both players are considered to be the winner (and must “rejoice in their shared victory”). In the set of 100 games played, there were 8 2-way ties and 2 3-way ties.

Table 6.1: Agent Performance

Agent	Score Mean	Est. Score Variance
Planning Agent	29.71	27.81
Fifth Card	23.89	68.95
Big Money	23.22	76.25
Card Value	20.48	144.20

We performed statistical hypothesis testing on these results. Using Wilcoxon's signed-rank test [15] we tested the mean score of the planning agent against each of the three simple-strategy agents to determine whether the difference is significant. The scores of two players in each of the 100 games constitute input pairs of data to the test.

Wilcoxon's test was employed instead of a parametric test like the z-test or the t-test because the score data was found to be not normally distributed.

Table 6.2: Agent Performance Significance

Matchup	Z-Value
Planning vs Fifth Card	6.58
Planning vs Big Money	6.83
Planning vs Card Value	7.1

The p-values for each of these z values is very small, much less than .00001, which indicates clearly that there is a statistically significant difference between our planning agent and the simple strategies.

Figure 6.2: Agent Average Scores

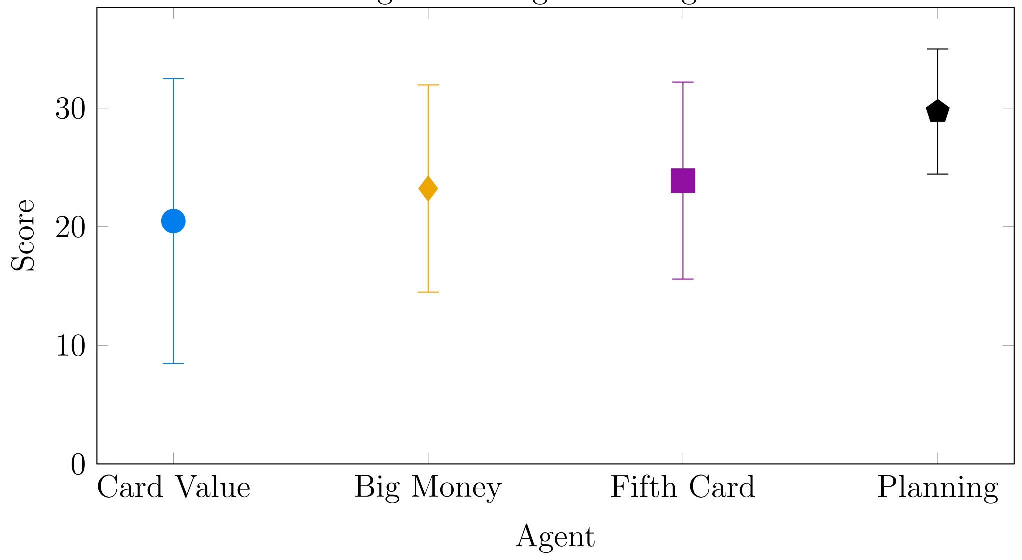
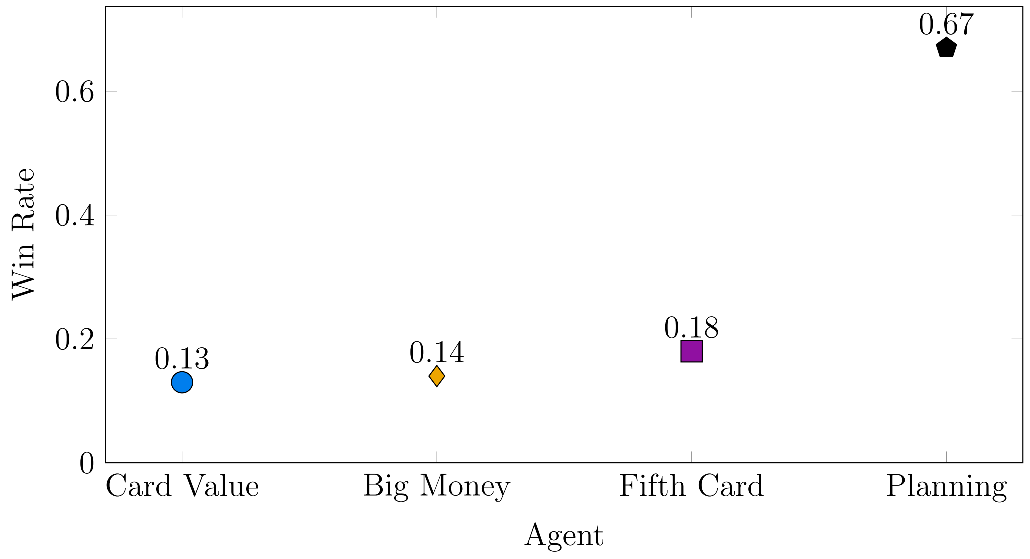


Figure 6.3: Agent Win Rates



6.2.3 Performance under Constraint

In order to test the AI's performance under constraint, we trained a series of networks on the same data, but with different amounts of recognized effects. The effects were ordered in order of decreasing frequency among the Dominion cards, and only the first n effects were chosen to learn on. These limited networks had a hidden layer size equal to the number of inputs chosen. 100 games were then played with each constrained network to find the minimum, maximum, and average score of agents informed by each network, as well as the overall win percentage.

The shaded regions in the score graphs represent one standard deviation above and below the mean score for each agent.

Table 6.3: Ordered List of Effects in Dominion

Effect Name	Base	Dom	Tot
Minus Actions	0	24	24
Plus cards	0	18	18
Plus Coins	3	10	13
Plus Actions	0	8	8
Plus Buys	0	4	4
May Trash a Card	0	4	4
Upgrade a Treasure card In Hand	0	1	1
Upgrade a Card by up to 2 coins	0	1	1
Trash a Copper for +3 Coins	0	1	1
Trash This Card	0	1	1
Steal Treasure cards	0	1	1
Play an action card 2 times.	0	1	1
Other players discard down to 3 cards.	0	1	1
Other Players: +1 Card	0	1	1
Other Players Put Victory card Back	0	1	1
May Discard Deck	0	1	1
Look for 2 Treasure cards	0	1	1
Gain a card costing up to 5 coins	0	1	1
Gain a card costing up to 4 coins	0	1	1
Gain a Silver on Deck	0	1	1
Each Player Reveals Top Card, You Decide if Discard	0	1	1
Draw Until 7 cards, May Discard Actions	0	1	1
Discard cards for +cards	0	1	1
Curse Attack	0	1	1

Figure 6.4: Restricted Agent Scores

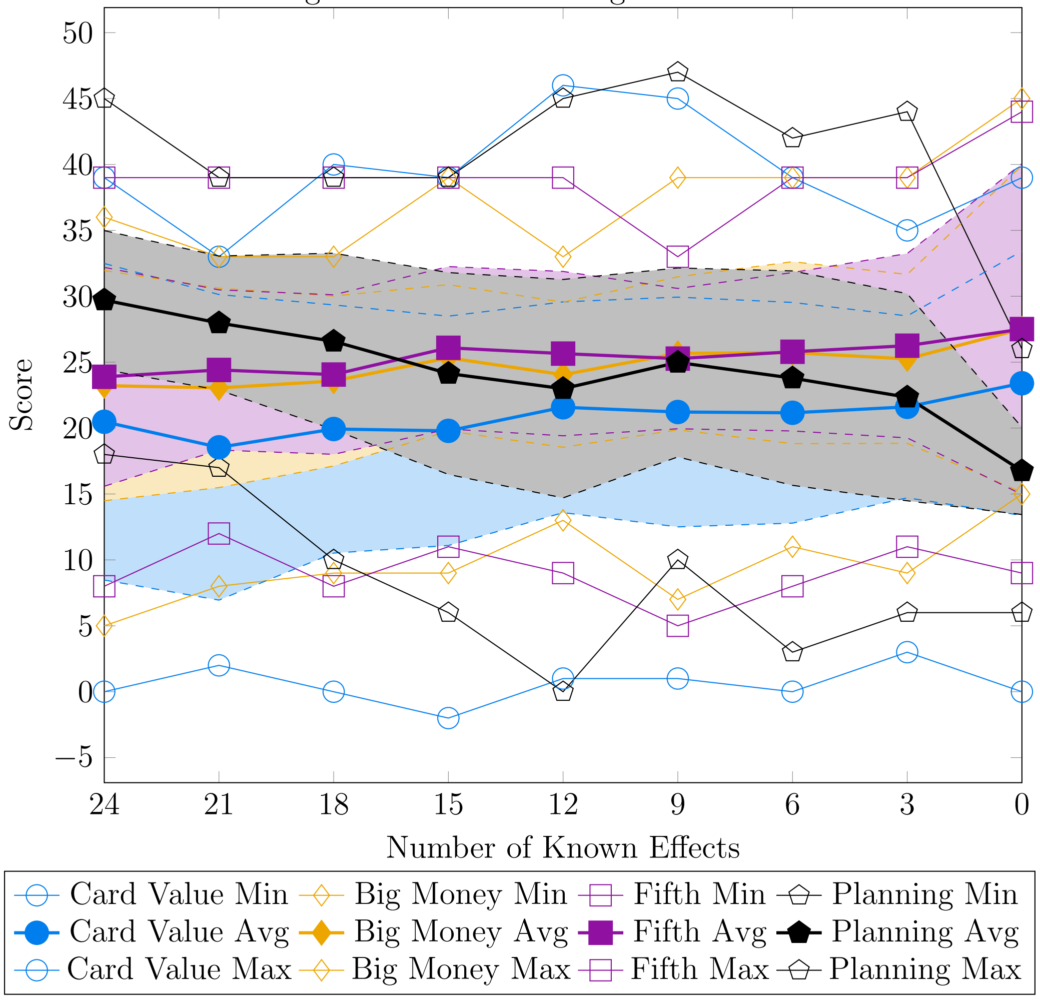
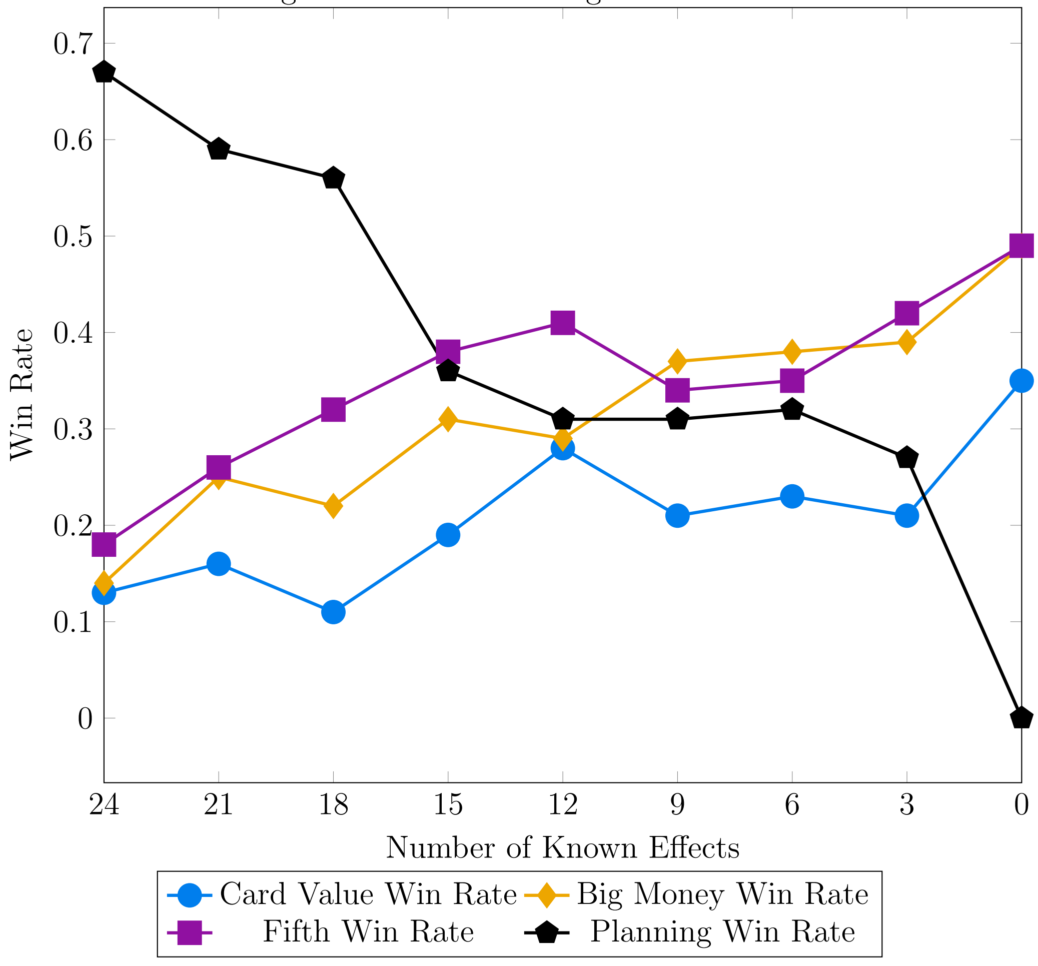


Figure 6.5: Restricted Agent Win Rate



It is clear from these results that limiting the agent's knowledge about the game space restricts its performance, but notice that a small amount of limitation leads to a small performance decrease. For the last limited-knowledge case which gave a mean score greater than the next best agent (18 known effects), we performed another Wilcoxon test and found a p-value of .0537, indicating a 94% confidence that the planning agent performs better on average than Fifth Card (which was the next runner up).

6.2.4 Extensibility

This was perhaps the most surprising result in these experiments. The agent, when trained on the effects of Dominion cards, is able to maintain a competitive edge against the next best agent, even in games whose Kingdoms are completely made up of new cards. We found the raw score results, again by Wilcoxon's test, to be statistically significant to a p-value of 0.001. A list of the effects in Dominion plus Intrigue is provided in Table 6.4 on page 45. Effects in bold are new effects from Intrigue.

Table 6.4: Ordered List of Effects in Dominion + Intrigue

Effect Name	Base	Dom	Int	Tot
Minus Actions	0	24	23	47
Plus cards	0	18	9	27
Plus Coins	3	10	7	20
Plus Actions	0	8	9	17
Plus Buys	0	4	2	6
May Trash a Card	0	4	0	4
Discard cards for +cards	0	1	1	2
Upgrade a Treasure card In Hand	0	1	0	1
Upgrade a Card by up to 2 coins	0	1	0	1
Upgrade a Card by up to 1 coins	0	0	1	1
Tribute Effect	0	0	1	1
Trash this card for +2 Coins	0	0	1	1
Trash a Copper for +3 Coins	0	1	0	1
Trash a Card	0	0	1	1
Trash This Card	0	1	0	1
Trash Opponent Card; They Gain Costing 1 Less	0	0	1	1
Trash 2 cards For Silver In Hand	0	0	1	1
Torture Players	0	0	1	1
Swindle Players	0	0	1	1
Steal Treasure cards	0	1	0	1
Reveal Hand: If No Action cards, +2 cards	0	0	1	1
Put Victory cards From Deck In Hand	0	0	1	1
Play an action card 2 times.	0	1	0	1
Other players discard down to 3 cards.	0	1	0	1
Other Players: +1 Card	0	1	0	1
Other Players Put Victory card Back	0	1	0	1
Name Card, Draw if Top Card Of Deck	0	0	1	1
May Discard Deck	0	1	0	1
Look for 2 Treasure cards	0	1	0	1
If 3 Action cards In Play, +1 Card, +1 Action	0	0	1	1
Gain a card costing up to 5 coins	0	1	0	1
Gain a card costing up to 4 coins	0	1	0	1
Gain a Silver on Deck	0	1	0	1
Gain Card Costing Up to 4 Coins + Get Bonus	0	0	1	1
Each Player Reveals Top Card, You Decide if Discard	0	1	0	1
Each Player Passes a Card Left	0	0	1	1
Draw Until 7 cards, May Discard Actions	0	1	0	1
Discard an Estate for +4 Coins	0	0	1	1
Curse Attack	0	1	0	1
+3 cards, Put One Back	0	0	1	1
+2 Coins or +4 Card Attack	0	0	1	1
+2 cards or +2 Actions or Trash 2 cards	0	0	1	1
+2 Actions or +3 cards	0	0	1	1
+1 Global Discount	0	0	1	1
+1 Card or +1 Action or +1 Buy or +1 Coin	0	0	1	1

Figure 6.6: Unfamiliar Cards Scores

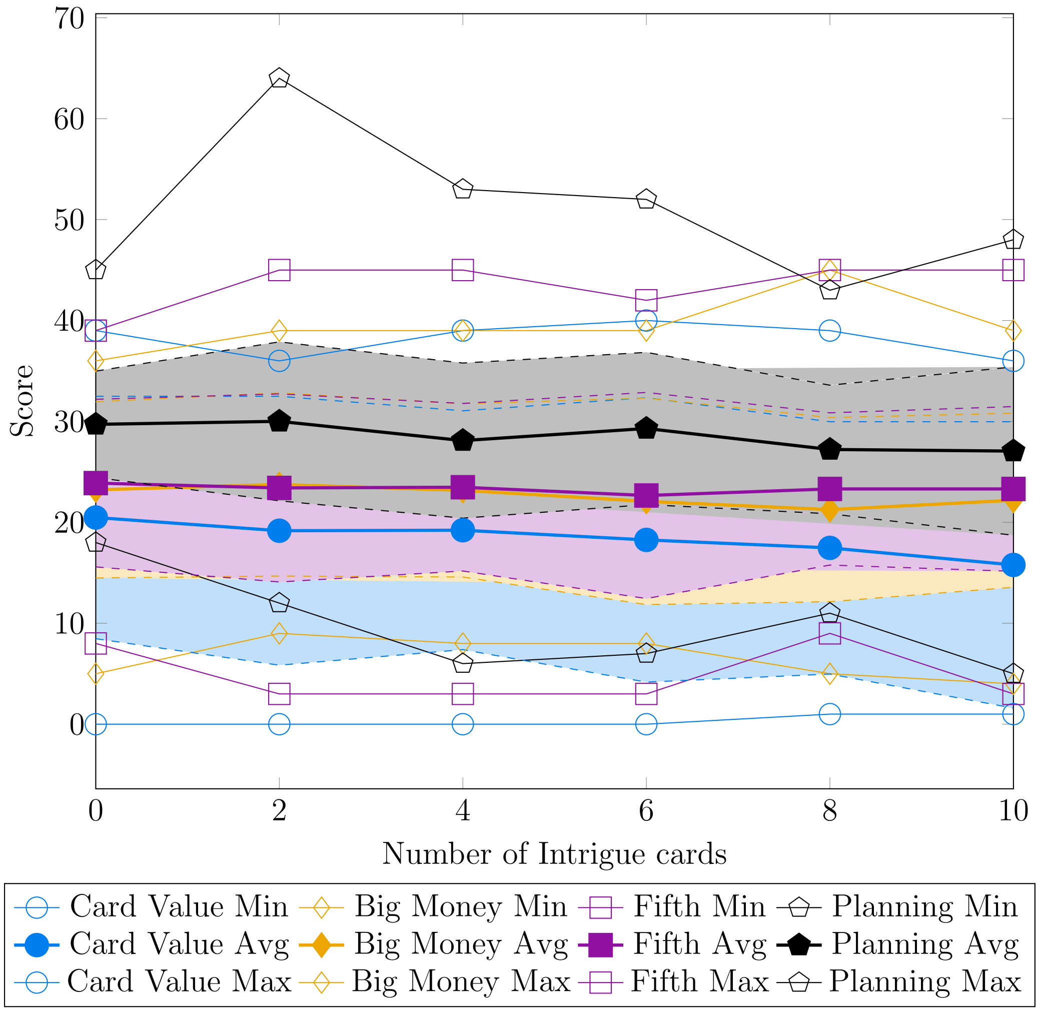
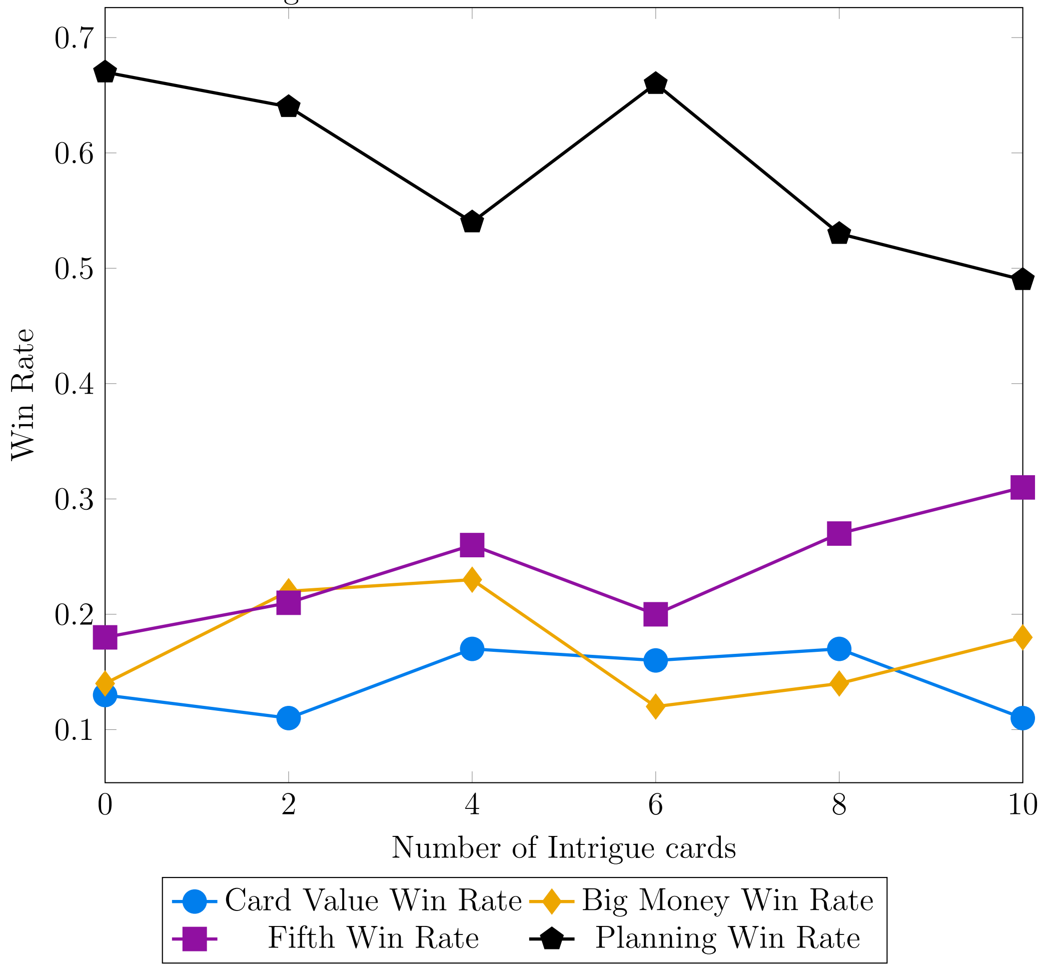


Figure 6.7: Unfamiliar Cards Win Rate



CHAPTER 7

Conclusions and Future Work

Using artificial neural networks to approximate heuristic functions for search algorithms, we were able to develop a successful agent for Dominion. Dominion's mixture problem nature allowed us to express game states as vectors in a large vector space, and the mechanics of the game were consistent enough to allow a neural network to easily approximate various functions of game state. The differentiability of neural network outputs allowed for simple gradient ascent algorithms to estimate node max-values quickly, simulating taking a few moves to move to an ideal state and then purchase Victory cards from that ideal state, much as human players will often plan an "engine" using Kingdom cards and try to harness the power of that engine to buy many Victory cards in the last several turns.

It is unfortunate that the algorithm did not cut the branching factor of the search tree sufficiently for the agent to make fully-informed decisions during games. Thankfully, even manually limiting the agent to 20 expansions of the top node in the priority queue produced very good results.

Our hope that this exploration into using artificial neural networks to inform tree search in deckbuilding games inspires further research in the areas of both deckbuilding games (which are good models for real problems in engineering and finance) and artificial neural network-assisted heuristic search algorithms.

7.1 Future Work

During this experiment we experimented with re-training the network on one extra output function of deck vectors: the maximum search tree node value. Because our algorithm for upper bounding a node's score is based entirely on other outputs of the neural network (minimum value and deck mobility), this is a feasible thing to do. Unfortunately, though using the network's max heuristic function our agent was able to explore many more planning nodes than before, we had mixed results with the accuracy of the learned function, and the idea had to be scrapped due to time restrictions.

For this project, we developed our own Dominion simulator in order to facilitate internal simulations of moves by the various agents. This necessarily restricted our ability to interface our simulator with simulators written for other Dominion AI projects in order to pit our agent against theirs. With considerable additional effort in passing information between simulators, our agent could be tested against, for example, Fisher's [9], to make more conclusions about the types of approaches which are effective in this particular problem domain.

The primary focus of this research has been on the long-term play strategy, and unfortunately we were unable to develop the short-term decisions of the agent further than that of the heuristic agents. This gives us the benefit of being better able to see how planning impacts agent performance, but for the sake of further development of a cogent Dominion AI, we would like the opportunity to expand our work in this area. The short-term play of Dominion is a domain well-suited to Monte-Carlo style rollout simulation, and it would be interesting to see how improved hand strategy improves the AI's overall performance.

The game-theoretic aspect of Dominion strategies as relates to Attack Card purchases constitutes a subset of the overall game strategy. This project focuses on the larger picture of general long-term strategy in a more isolated fashion, but future work could be done that delves into the attack game theory of Dominion.

For this project, only cards from Dominion and the first expansion, Intrigue, were implemented for the simulator. It would be enlightening to see how the agent performs in even more game expansions.

Currently the agent uses playouts with Big Money agents to estimate how many turns are left in the game. This turns out to be a fairly accurate estimate (usually correct within 1 turn) when playing with the heuristic agents, but against other opponents this prediction could prove flawed. A neural network to learn how game length is impacted by other aspects of the game would be a logical next step for this project.

Bibliography

[1] Erik Arneson. Dominion. *About.com Board/Card Games*, 2011.

[2] Guy Broeck, Kurt Driessens, and Jan Ramon. Monte-Carlo tree search in poker using expected reward distributions. In Zhi-Hua Zhou and Takashi Washio, editors, *Advances in Machine Learning*, volume 5828 of *Lecture Notes in Computer Science*, pages 367–381. Springer Berlin Heidelberg, 2009.

[3] Chenghui Cai and S. Ferrari. A q-learning approach to developing an automated neural computer player for the board game of Clue. In *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 2346–2352, June 2008.

[4] Murray S. Campbell and T.A. Marsland. A comparison of minimax tree search algorithms. *Artificial Intelligence*, 20(4):347 – 367, 1983.

[5] Tristan Cazenave and Bernard Helmstetter. Combining tactical search and Monte-Carlo in the game of Go. In *IN: CIG05*, pages 171–175, 2005.

[6] Philip H. Crowly. Hawks, doves, and mixed-symmetry games. *Journal of Theoretical Biology*, 204(4):543 – 563, 2000.

[7] Andries P. Engelbrecht. *Computational Intelligence: An Introduction*. Wiley Publishing, 2nd edition, 2007.

[8] Markus Enzenberger and Martin Müller. Fuego an open-source framework for board games and go engine based on Monte-Carlo tree search, 2009.

[9] Matthew Fisher. Provincial: A kingdom-adaptive AI for Dominion. Online Project Report.

[10] Rasmus Bille Fynbo and Christian Sinding Nellemann. Devoloping an agent for Dominion using modern AI-approaches. Technical report, IT University of Copenhagen, 2010.

[11] Matthew L. Ginsberg. Gib: Steps toward an expert-level Bridge-playing program. In *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99*, pages 584–589, 1999.

[12] Kieran Greer. Computer Chess move-ordering schemes using move influence. *Artif. Intell*, 120(2):235–250, 2000.

[13] Kocsis, Uiterwijk, Postma, and van den Herik. The neural movemap heuristic in Chess. In *CG: International Conference on Computers and Games*. LNCS, 2003.

[14] Levente Kocsis and Csaba Szepesvri. Bandit based Monte-Carlo planning. In Johannes Frnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer Berlin Heidelberg, 2006.

[15] Richard Lowry. Wilcoxon signed-rank test online calculator. Last visited on 21/4/2014.

[16] Christopher D. Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, March 2011.

[17] D.E. Rumelhart, G.E. Hintont, and R.J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[18] J Schäfer. The uct algorithm applied to games with imperfect information. Technical report, 2007.

[19] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99127, Jun 2002.

[20] Donald X. Vaccarino. Dominion, October 2008.

[21] Nees Jan van Eck and Michiel C. van Wezel. Application of reinforcement learning to the game of Othello. *Computers & OR*, 35(6):1999–2017, 2008.

[22] Joel Veness, Marc Lanctot, and Michael Bowling. Variance reduction in Monte-Carlo tree search.

[23] Christopher J.C.H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

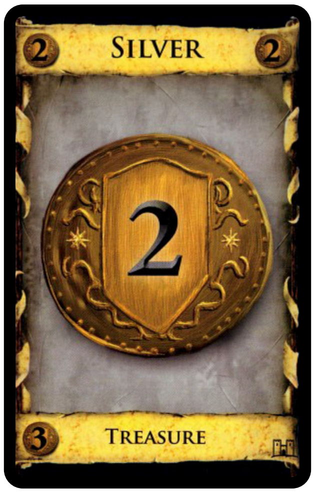
[24] Bin Wu. The computational intelligence of computer Go. In Yongheng Wang and Xiaoming Zhang, editors, *Internet of Things*, volume 312 of *Communications in Computer and Information Science*, pages 633–638. Springer Berlin Heidelberg, 2012.

APPENDIX A

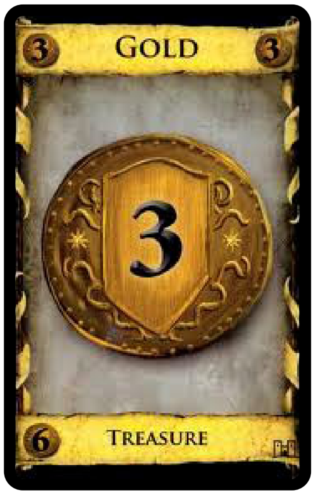
Example Cards

These are the cards used in the various examples in this paper.

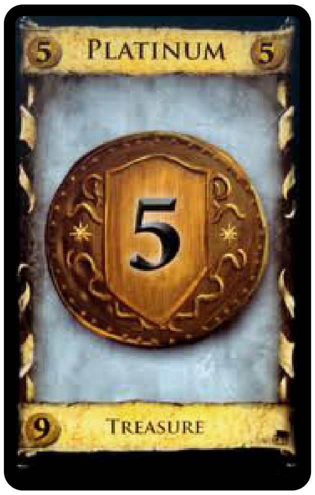
Copper is the most basic Treasure card. Each player starts with seven of them, and they are useful for gaining better cards. They can be bought for 0 coins, which leads some beginners to buy them whenever they have extra Buys. More advanced players realize that doing this effectively dilutes the deck, instead looking for ways to get rid of Coppers instead of gaining more of them.



Silver is one of the most consistently good 3-coin Buys in the game. It is nearly always one of the best purchases to make early in the game.



Gold is a powerful card, and one of the best buys for 6 coins.



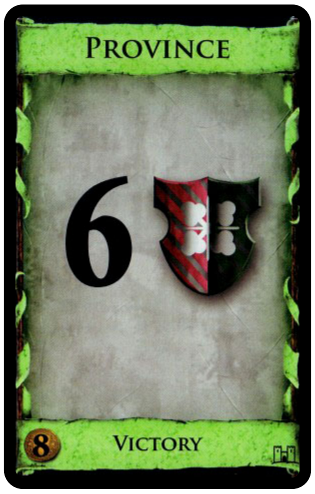
Platinum is from the Prosperity expansion, and is essentially a supercharged Gold.

ESTATE
1
2
VICTORY

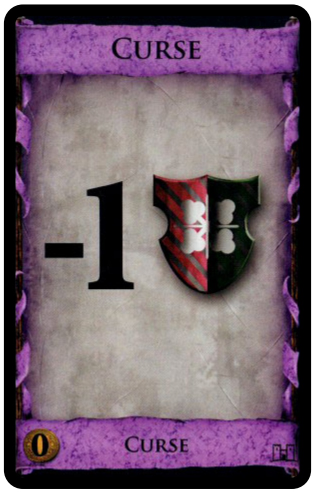
Estate is the cheapest Victory card. Starting out with three of them tends to do more harm than good, as they only provide three points but serve to make 30% of a player's starting hand useless.

DUCHY
3
5
VICTORY

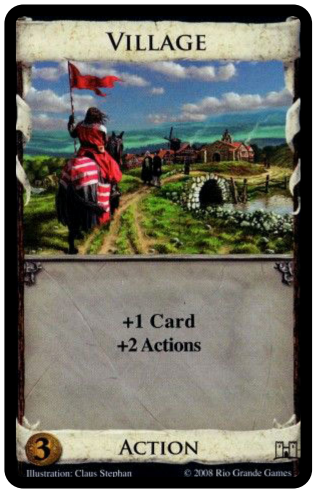
Duchies are worth more points per coin than Estates, but they are not as good as Provinces. They are a good option in the last few turns of the game, when adding "useless" cards to a large deck with only a few turns remaining is less detrimental to deck performance.



Provinces are (in the original game) the best Victory card available. Unless it is very early in the game, typical strategy is to always buy Provinces if it is possible to.



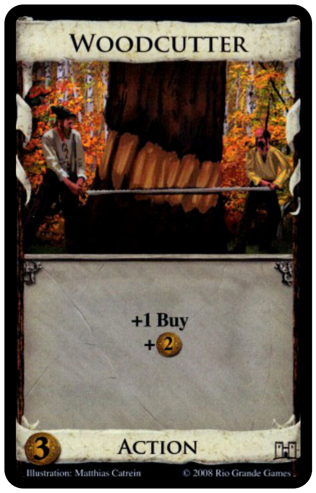
Curses are negative points. Few players would willingly buy one, but they can be given one by certain Attack cards, such as the Witch.



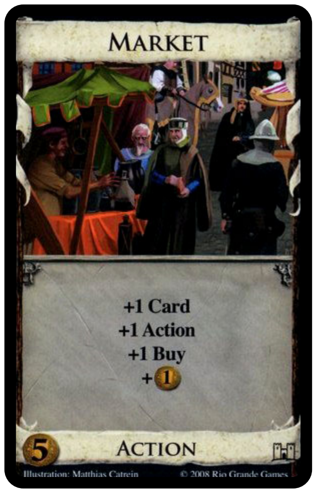
The Village is useful for combos. It does not itself provide much benefit, only giving +1 Card, but its +2 Actions effect makes it possible to chain Villages with cards that give a high +Cards, such as the Smithy. This chaining comes from the fact that the +2 Actions of the Village leave still one more action after playing a pure +Cards card such as the Smithy.

SMITHY
+3 Cards
4
ACTION
Illustration: Matthias Catrein
© 2008 Rio Grande Games

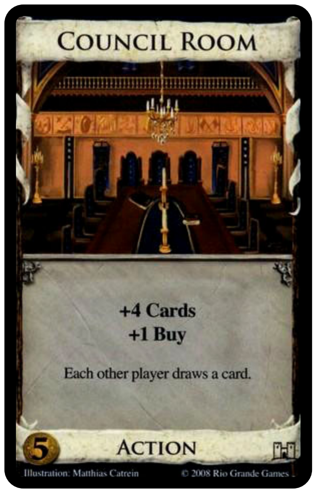
The Smithy is a good counterpart to the Village as a cheap +Cards Action card. It is important to note that the Village/Smithy combo discussed here is only as useful as the *other* cards in the deck: it is not useful to play many Villages and Smithies to draw more Villages and Smithies and ultimately produce few coins for card purchasing at the end of the turn.



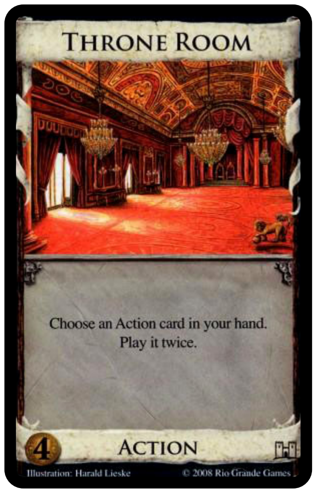
The Woodcutter gives +2 Coins and an extra buy, but comes at the subtle cost (over Silver) of being an Action card. This means that if one draws a Woodcutter with no extra Actions remaining, the Woodcutter cannot be played and a Silver would have been preferable.



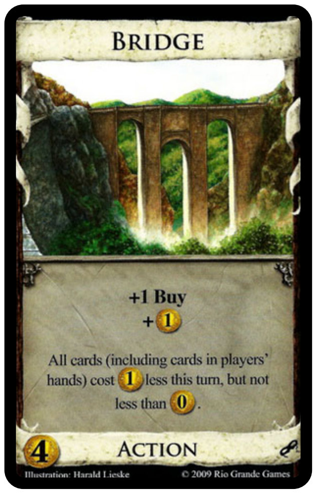
The Market is one of the best 5-cost cards in the original set. It gives small amounts of every benefit, including +1 Card and +1 Action, which means it chains well with itself, as well as most other Action cards.


COUNCIL ROOM
+4 Cards
+1 Buy
Each other player draws a card.
5
ACTION
Illustration: Matthias Catrein
© 2008 Rio Grande Games

The Council Room gives a high amount of card draw and an extra +1 Buy, but has the drawback of also giving card draws to the other players.


THRONE ROOM
Choose an Action card in your hand.
Play it twice.
4
ACTION
Illustration: Harald Lieske
© 2008 Rio Grande Games

The Throne room is one of the most powerful (despite its low cost) cards in Dominion when combined with other cards. It effectively multiplies the effectiveness of other cards.


BRIDGE
+1 Buy
+1
All cards (including cards in players' hands) cost 1 less this turn, but not less than 0.
4
ACTION
Illustration: Harald Lieske
© 2009 Rio Grande Games

Bridge is a card which leads to some very interesting game situations. Through a large number of played Bridges, a player can feasibly reach a game state in which all cards are free (for the rest of the turn) and the player has a large number of Buys. Players have been known to use Bridges to buy out the rest of the Province deck and end the game.