Analysis of Dominion: the original deck-builder
15-780 Project Report

Sarah Loos (sloos) and Patrick Xia (pjx)

Computer Science Department, Carnegie Mellon University

Abstract. We analyze the game of 2-player *Dominion*, a popular deck-building card game published in 2008. We develop a framework that treats the game of *Dominion* in a different way depending on what phase it is in: in the beginning of the game, we abstract the game as a fully-observable Markov decision process, and near the end of the game, when a tree search becomes tractable, we switch to a strategic player that fully takes into account the possible actions of the second player. Our focus is on a subset of the game (only the base cards) to allow for concise analysis, but the general approach can extend to the entire game.
Our approach on the early-game focused on solving the game via requires some additional analysis, as some simplifying assumptions have made for a poor player. However, our endgame analysis offers some marginal improvement against state-of-the-art computer players, though the improvement is marginal and often swamped by the variance in the game. Our endgame analysis was also able to recover a popular advanced human strategy, indicating that improved versions should be able to fare competitively against humans.

1 Background

The popular card game *Dominion*, published by Rio Grande Games, presents an interesting new challenge for good game AI. It has pioneered what is called the "deck-building" genre of card games, where the primary game mechanic is that players make decisions on which cards to include in their deck. The game is very popular, with a significant online following, and some rudimentary work on computer players has been done. The official game implementation, at Goko, offers a very rudimentary AI to play against, but most players of medium skill are able to easily outpace their algorithms.

At its core, *Dominion* is a partial-knowledge game that requires deep game tree analysis in its execution (as decisions made now change the composition of the deck and therefore the value of future moves). Though many amateur attempts have been made at a good game AI for *Dominion*, most of these attempts rely on heuristic from what is commonly agreed to be "good gameplay decisions" and not based on any rigorous formulation of the problem from a planning or decision perspective. In this project, we examine the planning that is required for good deck building, and the decisions that are necessary for good endgame play.

2 Game Mechanics

The goal of *Dominion* is to maximize the number of cards offering victory points in one's deck at the end of the game. We examine a stripped-down version of Dominion in which there are two types of cards: Treasure cards and Victory cards, as shown in Figure 1. Players begin the game by drawing five cards; every turn, players are allowed to buy one card with Treasure they have in hand (adding the card to their deck for the remainder), and then discard their cards. All discarded cards are eventually reshuffled, so deck sizes increase as the game goes on.

Fig. 1: Base cards for *Dominion*

There is a tension between the goal of maximizing victory points and having a deck that allows for future purchases of victory points; since cards that offer victory points do not add to treasure, having a deck that is high in victory points makes it more difficult to buy victory points in the future. Therefore, good endgame analysis of exactly when to stop buying additional Treasure and to start buying victory points is essential to a good player's strategy.

An interesting point to mention here is that the full set of cards includes dozens of cards (see Figure 2 for a somewhat-out-of-date sampling) that change from game to game (ten cards are chosen randomly out of the entire set). For the purposes of analysis, we have restricted the game to the base set of treasure cards and victory cards. It is important, however, to note that the generalized form of our approach is certainly not limited to this subgame and that our strategies can certainly deal with the added cards that are present in a full game of *Dominion*.

3 Our approach

The approach for a good *Dominion* AI in the wild right now is essentially a search over heuristic strategies: people define the parameters of what's important about a game of *Dominion* and then tune those parameters by hand by observing the percentages of games such a strategy wins. If this iteration is done by a machine, the approach would almost be akin to genetic programming. However,

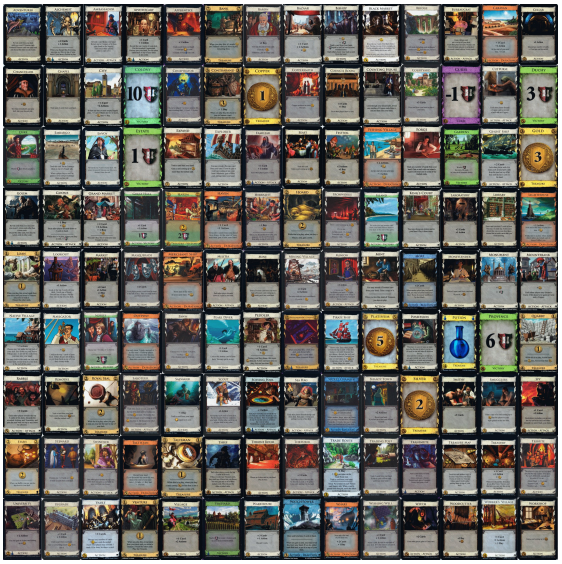

Fig. 2: Kingdom cards for *Dominion*, circa 2010

this approach requires a lot of intuition about the game and also misses some of the larger game state — novel strategies cannot be developed because the strategies are already essentially encoded; the parameters are only a small form of tuning.

From the perspective of the literature and our AI course, the traditional approach when presented with this game (which primarily has to deal with building a deck to account for the randomness of drawing cards) is to model it as a partially-observable Markov decision process. Here, a Dominion strategy maps from the current observable game state to the actions that are available to the current player. However, this also abstracts away the second player entirely by subsuming him into the state and transition model.

4 Implementation

We base all of our strategies on the open-source project Dominiate [4]. Dominiate is an all-purpose *Dominion* simulator and comes with a library of community-contributed strategies. A lot of the work on the project has the goal of making it easier for people unexperienced with coding to write strategies of their own, which explains many of the design decisions (it is implemented in CoffeeScript, a language that compiles to JavaScript so that strategies can be run completely in the browser). This also makes the library of AIs completely of the form earlier mentioned—"if the deck has x amount of Treasure and the piles are roughly y deep, then buy card z."

However, because Dominiate is a complete Dominion simulator, the simulator itself needs to keep track of a lot of internal state in order for the simulation to return the correct results. Therefore, it is very simple (but runtimes are slow, again, due to choice of language and runtime environment) to create a minimax tree search player. Our "build" step is currently done outside of the simulation environment because we focused solely on the base set of cards and did not require a heavy simulation platform—however, further work in this area can totally be done by borrowing the action semantics from the *Dominiate* simulator itself.

5 Extensive Form Representation

In Figure 1 we illustrate an extensive form representation of just one round of the game. We assume that the value of the hand for the player is known. In this example extensive form, the equilibrium is just the very basic dominating strategy. While this simple equilibrium strategy is a result of the simplifications we made to the state space, adding in Kingdom cards (that differ from game to game) in the endgame results in a state space that is too large for simple analysis and is not incredibly useful to understanding the dynamics behind this game.

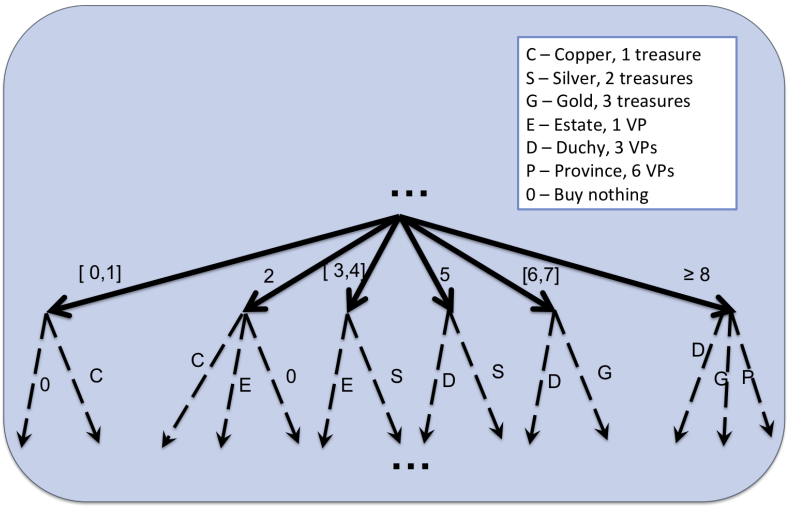

Fig. 3: Extensive form representation for a single hand for one player.

While this approach (generalized equilibrium solving) is useful for highly-strategic two player games like poker, it is probably too large of a hammer for *Dominion*. The reason here is that *Dominion* has relatively little player interaction in the beginning stages of the game, and the depth of the game tree

is fairly large, which makes for a lot of wasted computation. We believe the game may be better suited to optimization analysis rather than a search for an equilibrium. We, however, point the interested reader to [5] for a general-purpose equilibrium solver for 2-person zero-sum games if one wants to extend our approach in this direction.

6 2-ply Minimax

As mentioned in Section 4, our minimax tree traversal algorithm is based on the internals of the open-source project *Dominiate*. The heuristic value of the game at any given node is simply the difference between the first and second player's scores. Since all of the important information is fully observable (specifically, the score of one's deck), this endgame analysis is easy to implement on the behalf of one player.

Unfortunately, as mentioned in the earlier section, *Dominiate* was never designed for speed, and such a search can only happen at realtime speeds if severely depth-limited. Since the game of Dominion suffers from very high variance, many iterations of testing are required to determine whether or not one strategy beats another. We implemented a 2-ply minimax algorithm that looks ahead by two player moves to choose the best course of action, using a worst-case opponent model (the opponent can buy anything that their deck allows them to).

We noticed during trial runs of the 2-ply minimax algorithm that in some cases, the computer player would choose to buy a Duchy (which gains 3 victory points) instead of a Province (which gains 6 victory points) even when the player could afford the Province. This is counterintuitive player because a Duchy is a strictly inferior card to a Province. Upon further analysis, we noticed that this only happened when the Province pile only had two cards remaining and the player was only slightly behind: this action therefore makes sense in retrospect because buying the penultimate Province means that the player loses if the other player responds by buying the last province. An example might make this clearer: suppose Player 1 is 2 points behind. Buying a Duchy puts him at 1 point ahead and buying a Province puts him at 4 points ahead. But if Player 2 buys a Province on the next turn, Player 1 loses by 4 points, whereas if he bought a Duchy, Player 1 would only be behind by 4 points and possibly could buy a Province on his last turn to end the game.

We tested a pure version of this strategy (which, unfortunately, upon a literature review before writing this writeup, is not novel, but is an "advanced strategy" used by humans known as the Penultimate Province Rule) to see the advantage that a strategic player would have against a player with no strategy at all. The results of 100,000 iterations are shown in Figure 4.

The resulting player is approximately .4% better than the naïve one. Unfortunately, this is not a very large improvement, but it shows that very simple search can improve the outcome of games and offer an advantage to a player. Deeper tree search, if used in combination with a successful heuristic, might be able to offer even larger improvements. The importance here, though, is that we

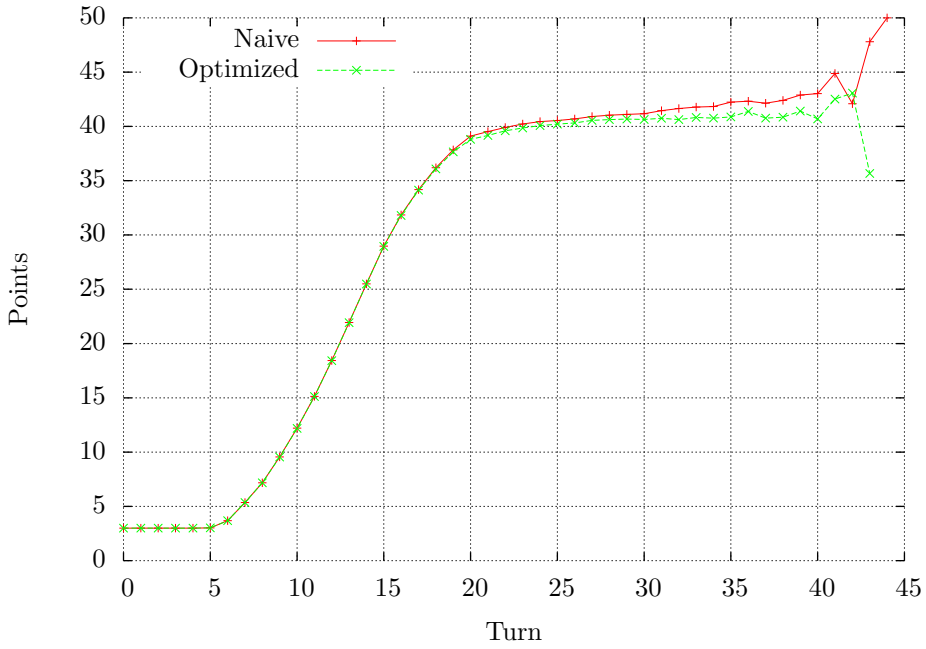

Fig. 4: A plot of a player that uses the "penultimate Province rule." We can see that the optimized player gets fewer points on average, but wins slightly more games due to that slight peak.

have presented a generalized framework to be able to find new strategies in the presence of new cards as well (the endgame need not be the simplified endgame that we have considered here)—discovering new strategies like the "Penultimate Province Rule" can come simply from observing seemingly-anomalous behavior of a tree-search baed player.

7 Q-Learning

So far our approaches in Sections 6 and 5 have focused on the interplay between two players during the endgame of *Dominion*. However, individual actions of the opposing player don't have a big impact on your own choices, especially early in the game. For example, the other player can not change the composition of your deck, or the value of your hand. And, once you buy a victory card, you will have those points until the end of the game. The opposing player will, however, influence when the game ends.

In this section and in Section 8 we analyze *Dominion* as an optimization problem for a single agent, rather than as a two-player game. We do this by

abstracting the actions of the opposing player to its most important role: determining the length of the game. To do this we run 200,000 games using our benchmark "Big Money" strategy and record the total number of turns taken in each game. This results in the distribution found in Figure 5.

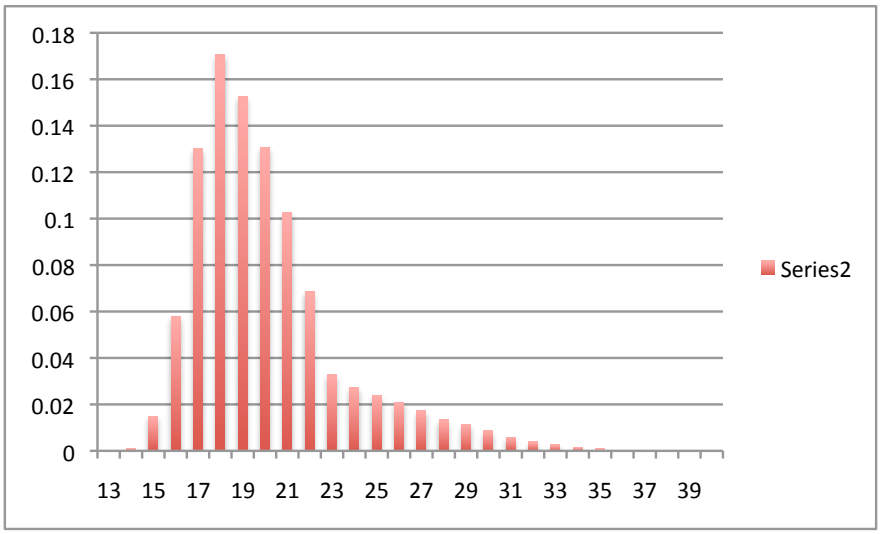

Fig. 5: Distribution of total number of turns.

Now we can consider the game of *Dominion* as a Q-Learning problem, where the probability of the game ending at any given turn (modeled as a sink state) is derived from the sampled data in Figure 5. The only state that has a reward is the sink state, and that reward is the current number of victory points in the player's deck.

If we were to model the game precisely, we would model the state as the exact composition of the deck, as well as each individual hand that could be drawn from each of those decks, in addition to the number of turns taken. However, due to the combined combinatorial nature of the deck composition, and the possible hands that can be drawn out of each deck, this state space is intractably large. However, if we had the computational power to handle it, we would model the system's state, actions, and rewards as:

State: {all possible decks, all possible hands given the deck, turns taken}
Action: {Province, Duchy, Estate, Gold, Silver, Copper, nothing}, where each card is only available if the value of the hand exceeds the cost of that card.
Reward: {If in sink state $\rightarrow$ victory points in deck}

Since it is necessary to reduce the state space in order to make the problem tractable, we first look at simplifying our model of the hand and the deck. Instead

of modeling each card in the deck, the state only tracks the total number of cards in the deck, the sum of all the victory points in the deck, and the sum of all the treasures in the deck. Then we simplify the state again by taking the value of each hand to be its expected value, i.e., $5 * (\frac{sum\ of\ treasures}{total\ cards})$.

We also reduce the complexity of the actions by only allowing the largest affordable treasure or the largest affordable province to be purchased. Since we don't restrict the total number of treasures and provinces, this is a dominating strategy.

State: {number of cards, value of all cards, victory points, turns taken}
Action: {buy largest coin, buy largest victory card, buy nothing}
Reward: {if in sink state → victory points}

After running Q-Learning through $500,000,000$ exploration episodes, the average reward over a thousand episodes, each starting at a random point in the state space, improved from 31.36 to 33.30 rewarded victory points. The strategy that the Q-Learning approach gives when starting at the initial state and always choosing the action with the maximum corresponding Q-Value is outlined in Section 8.

Even though we have abstracted the state so much, we still have trouble getting the Q-Learning algorithm to converge on this problem. So, in the next section, we analyze *Dominion* as a Markov Decision Process (MDP), which can be solved directly.

8 Solving the MDP

The MDP mentioned in Section 7 can also be solved directly with a variety of MDP solvers; Q-learning is an approach typically used when the environment is not known. Although state-of-the-art POMDP solvers, such as ZMDP[6], can also solve MDPs, our MDP has special properties that make it easier-to-solve: specifically, the graph is acyclic (since we introduced the turn number into the state[1], which means that only one iteration through the state space is necessary: on turn 30, we set the utility of every state to be the number of points it contains, and then we iterate, starting from turn 29 and going down, updating the value of U, the expected utility from choosing a given action (which, of course, is the value of the current state times the probability the game ends this turn, plus the probability the game continues and the value of the next state). It is clear that this approach will result in an optimal path through this state space.

After solving the MDP, we obtain that the optimal value of this game is 31.9285, with the optimal choices to be made at every turn given in the table

[1] One might ask why we've done this seeing that it blows up the state space, but it was very difficult to model the termination probability well if not given the state number. It's empirically not a Poisson process (see Figure 5), so more training is needed on some other aspects of the state space. Future modeling could negate this assumption.

below. We note that the Q-learning player has learned well, having matched the optimal value of the game.

Turn	MDP action	MDP VP	$\binom{Q-learning\ action}{500{,}000{,}000\ episodes}$	$\binom{Victory\ Points}{500{,}000{,}000}$
1	Buy Silver	3	Buy Silver	3
2	Buy Silver	3	Buy Estate	4
3	Buy Silver	3	Buy Silver	4
4	Buy Silver	3	Buy Silver	4
5	Buy Silver	3	Do Nothing	4
6	Buy Silver	3	Buy Estate	5
7	Buy Silver	3	Buy Silver	5
8	Buy Gold	3	Buy Silver	5
9	Buy Duchy	6	Do Nothing	5
10	Buy Duchy	9	Buy Duchy	8
11	Buy Gold	9	Buy Silver	8
12	Buy Duchy	12	Buy Duchy	11
13	Buy Duchy	15	Buy Silver	11
14	Buy Duchy	18	Buy Silver	11
15	Buy Duchy	21	Buy Duchy	14
16	Buy Duchy	24	Buy Duchy	17
17	Buy Duchy	27	Buy Estate	18
18	Buy Duchy	30	Buy Estate	19
19	Buy Silver	30	Buy Estate	20
20	Buy Duchy	33	Do Nothing	20
21	Buy Silver	33	Do Nothing	20
22	Buy Duchy	36	Buy Estate	21
23	Buy Silver	36	Do Nothing	21
24	Buy Duchy	39	Buy Estate	22

Of course, our strategy does not work in an actual game of *Dominion* because our simplification of the state space means that it's impossible to buy that number of Duchies. In addition, the use of the expected value to determine what cards one can buy is a problem because it is very difficult to amass 8 treasure worth of expected value, but it's hard to make a deck that rarely ever has 8 treasure (loosely speaking, buying Gold makes your treasure very "lumpy", which increases the variance, which means that there is a good chance that you can draw 8+ hands even with relatively low expected value).

9 Conclusions

Our idea of splitting the game into a "building" phase and a "strategic" phase, on first glance, does not seem to have made a very good player. The "building" phase player optimizes for a very flat strategy—buying low-valued victory point cards—that will get beaten even by the most amateur human player. There, however, is some merit in modeling the game like this. First, improving the

model that our "building" player uses results in a much stronger player—one that doesn't attempt to buy all the Duchies, for example, because there simply aren't that many Duchies available. Secondly, our "strategic"-phase player can already be bolted on top of a heuristic-based player that is seen in the wild today, which offers improvements over any of the AIs available in the *Dominiate* library. This suggests that further work on this analysis will result in a truly superior computer player for the game of *Dominion*, and one that doesn't require individual human tuning for every additional card added to the library.

In fact, there exists a happy medium between tracking all possible permutations of the deck and the simplified version that we used. We can add granularity to the treasure cards, add granularity to the victory point cards, and add a hand value to each state, which only produces a state that is a few hundred to a few thousand times larger (depending on the specific constants used). Although this will make the computation take a much longer period of time, we note that this computation can easily be parallelized within each turn computation, which makes it tractable for large computing clusters.

Our experience in using Q-learning for an acyclic Markov model seems to indicate that there is sometimes good convergence to an optimal strategy. Further work may be warranted on this observation, as acyclic Markov models are certainly simpler than the general case. Since Q-learning frameworks are often present in many toolkits and libraries, it can often be easier and less error-prone to implement a Q-learning solution (because there exists a clear separation between the optimizer and the model under consideration) than to use an MDP solver, which often requires additional integration work. In addition, Q-learning solutions can either attempt to learn an unknown game simply from observing it being played, or be able to learn the value of various cards in *Dominion* without knowing their specific interactions.

References

1. http://www.riograndegames.com/games.html?id=278
2. Varley, Allen (2009-08-09). "Dominion Over All". Escapist. http://tinyurl.com/mkhpk5
3. http://forum.dominionstrategy.com/index.php?topic=619.0
4. https://github.com/rspeer/dominiate
5. Gilpin, A. Peña, J, and Sandholm, T. First-Order Algorithm with O(ln(1/epsilon)) Convergence for epsilon-Equilibrium in Two-Person Zero-Sum Games. *Mathematical Programming 133(1-2)*, 279-298.
6. Smith, T. ZMDP Software for POMDP and MDP Planning. Github repository. trey0/zmdp