Computational Game Theory:

Solving Obstgarten

Author: Jonas Schumacher
E-Mail: contact@jonas-schumacher.com
Website: `https://www.jonas-schumacher.com`
Github: `https://github.com/jonas-schumacher`

Contents

1 The Game

I was playing this nice game with my 3-year old nephew lately when I thought about how big our chances were to win against the evil crow in the middle of the game board. In the game, the players try to harvest fruits from 4 different trees while they are competing with an evil crow, who'd also love to eat those fruits. Being a game for children, the rules are quite easy:

1.1 Preparation

In the beginning there are 4 trees with 10 fruits each (cherries, apples, plums and pears). The 9-piece puzzle of the crow in the middle of the game board is empty.

1.2 Playing

One by one, the players throw a die, which can show one of the following symbols:

1. Cherry
2. Apple
3. Plum
4. Pear
5. Fruit basket
6. Crow

In the first 4 cases, the active player gets to harvest *one* specific fruit from a tree.

In case number 5, the player can choose *two* fruits from any tree(s) of his choice. This is, by the way, the only strategy the players can use. Otherwise, the game would be fully determined and the players could not influence their odds against the crow.

In case number 6, the player must add one piece to the puzzle in the middle of the game board.

1.3 End of the game

If all trees are empty and there are still pieces of the crow missing, the players win. If the puzzle is completed *before* the last tree is fully harvested, the crow wins and all players lose.

This means that this game is a cooperative game; all players play against the crow and it should make no difference to the result of the game if there were 4 players throwing 1 die or 1 player throwing the die 4 times.

2 Analytics

2.1 States

First, let's think about how a certain state of the game is characterized. The state of the game is like (a unique) photo of the game board: it shows the result of what has happened so far but it doesn't reveal the whole history of the game.

In Obstgarten, the state is determined by the fruits still hanging on the trees and the number of crow pieces missing in the middle (One could also count the harvested fruits instead and/or the number of crow pieces already on the game board).

Figure 1: One possible state of the game

In figure 1, the state (Cherry, Apple, Plum, Pear, Crow) is (3,4,2,3,2), because there are still 3 cherries, 4 apples, 2 plums and 3 pears not yet harvested, and the crow still needs to 2 pieces in order to win.

You might have noticed that if we switched cherries and apples, we would get a slightly different state: (4,3,2,3,2) instead of (3,4,2,3,2). Are these two states really that different,

if we only want to calculate the probabilities to win against the crow? Because the die shows red (cherries) and green (apples) with the same probability ($\frac{1}{6}$), we can make use of that symmetry to reduce the state space:

Instead of defining our state as (Cherry, Apple, Plum, Pear, Crow), we'll define it as (1st tree, 2nd tree, 3rd tree, 4th tree, Crow) with the trees in descending order regarding the number of remaining fruits. The two states above would therefore both result in the same state: (4,3,3,2,2)

The game always starts in state $(10, 10, 10, 10, 9)$ = **initial state**.

The players win (= **stationary state**), if the state is $(0, 0, 0, 0, x)$, with $x > 0$. That is, all fruits are harvested with the puzzle not yet being completed.

Equivalently, the players lose (= stationary state), if the state is $(y_1, y_2, y_3, y_4, 0)$, with at least one $y_i > 0$. That means the puzzle is completed, but there is at least one tree with remaining fruits.

2.2 State space

Having described the possible states, let's calculate the possible number of states. In order to do so, we'll compare the state space with a lottery: There is a bowl with 11 numbers (numbered from 0 to 10) and we'll draw a number from that bowl exactly 4 times. After each draw, we'll put the drawn number back into the bowl. After the 4th number is drawn, we'll sort them from highest to lowest. How many different combinations can we get?

Statistics tells us that we're facing a combination with repetition with n = 11 (number of elements in the original set) and k = 4 (numbers drawn). The number of multisets (= number of states in our example) is:

$$\binom{n+k-1}{k} = 1001$$

So excluding the puzzle in the middle, there are 1001 possible states. Now, for each and every "tree combination", there are 10 different puzzle states (numbered from 0 to 9), so that in total we have

$$N = 1001 \cdot 10 = 10,010$$

different states.

Remember: If we hadn't made use of the game's symmetry, the trees would be as unique as the puzzle in the middle and we would face a variation instead of a combination. The state space would then yield $N = 11^4 \cdot 10 = 146,410$ number of states.

At this point, we're already getting a feeling for the complexity of this supposedly "simple" game.

2.3 Strategies

Let's talk about strategies shortly, before calculating the winning probabilities. With Obstgarten, we're facing a game which is highly independent from the players strategy, because in 5 out of 6 cases, the players don't have any choice. That's why the game is relatively easy to play compared to a strategic game like Chess which doesn't depend on chance at all.

The only way players can modify their odds is the 5th side of the die – the fruit basket. In order to keep things simple, I'm suggesting 3 different strategies:

1. **"Positive": Always pick the fruits from the tree with most fruits left**
2. "Random": Pick fruits randomly
3. "Negative": Always pick the fruits from the tree with least fruits left

If not stated differently, I'll use the first strategy for the upcoming examples. I have a good feeling this strategy might pay off :-)

2.4 Modeling as a Markov Chain

Now we can finally start modeling the game. You might have noticed that, given a certain state, it doesn't matter what has happened so far: the game has no memory. That's why we can model it as a Markov Chain.

In order to apply a Markov Chain to our game we need to calculate the probabilities for moving from one of the $10,010$ states to another, that is we need to calculate $10,010^2 = 100,200,100$(!) entries. If we stuck to our original state space, we would face even $146,410^2 = 21,435,888,100$ calculations.

Let's get back to our example from the previous section. *Which states* can be reached with *which probability* from our given state of (4,3,3,2,2)?

$$p = \begin{cases} \frac{1}{6} & \text{for (3,3,3,2,2) = 1st fruit picked,} \\ \frac{2}{6} & \text{for (4,3,2,2,2) = 2nd or 3rd fruit picked,} \\ \frac{1}{6} & \text{for (4,3,3,1,2) = 4th fruit picked,} \\ \frac{1}{6} & \text{for (3,3,2,2,2) = fruit basket (positive strategy),} \\ \frac{1}{6} & \text{for (4,3,3,2,1) = crow,} \\ 0 & \text{for all other states} \end{cases} \tag{2.1}$$

For each state the leaving probability adds up to 1.

	A	J	K	L	M
1		[10, 10, 10, 10, 1]	[10, 10, 10, 10, 0]	[10, 10, 10, 9, 9]	[10, 10, 10, 9, 8]
2	[10, 10, 10, 10, 9]	0	0	0,666666667	0
3	[10, 10, 10, 10, 8]	0	0	0	0,666666667
4	[10, 10, 10, 10, 7]	0	0	0	0
5	[10, 10, 10, 10, 6]	0	0	0	0
6	[10, 10, 10, 10, 5]	0	0	0	0
7	[10, 10, 10, 10, 4]	0	0	0	0
8	[10, 10, 10, 10, 3]	0	0	0	0
9	[10, 10, 10, 10, 2]	0,166666667	0	0	0
10	[10, 10, 10, 10, 1]	0	0,166666667	0	0
11	[10, 10, 10, 10, 0]	0	1	0	0
12	[10, 10, 10, 9, 9]	0	0	0	0,166666667
13	[10, 10, 10, 9, 8]	0	0	0	0
14	[10, 10, 10, 9, 7]	0	0	0	0

Figure 2: States and probabilities

Figure 2 shows an extract from the **transition matrix** with some of the probabilities emphasized in color:

- **blue**: probability for reaching state (10,10,10,9,9) from initial state (10,10,10,10,9): $4 \cdot \frac{1}{6}$ [by throwing any of the 4 fruits]
- **yellow**: probability $\frac{1}{6}$ [by throwing "crow"]
- **green**: stationary state [the games is over because the crow has won]

2.5 Solving the Markov Chain

I will not got into detail here. My calculations are based on a lecture at KIT (Karlsruhe university) and a book on stochastic models [Waldmann and Stocker, 2012]. I'm sure you can find a proper explanation in any literature on (discrete) Markov Chains. Roughly

put, given the initial state and the transition matrix (using a fixed strategy), one can calculate the probability for ending up in any of the stationary states. Remember: the stationary states represent the end of the game – it means either the players or the crow have won.

Even though the game could take very long (just imagine the players constantly throwing "cherry", when there's still "apple" and "crow" left), the probability for that to happen will become lower and lower with every new die thrown. And, when moving towards an infinite number of dice thrown, the game will eventually end.

That means that our 100% probability starting in initial state $(10, 10, 10, 10, 9)$ will eventually be divided into the stationary states I already showed in section 2.1. The winning probability can now be easily calculated as:

$$\sum_{x=1}^{9}(0, 0, 0, 0, x)$$

2.6 Python Program and Results

You can find the code on my Github account: jonas-schumacher

The code can be divided into 4 major parts:

1. Setting game parameters: number of fruits per tree and chosen strategy
2. Setting and naming all $10,010$ states
3. Calculating the transition matrix
4. Solving the linear equation for each stationary winning state

After running the program for each strategy we can finally **calculate the winning probabilities**:

1. Positive strategy: 68.40%
2. Random strategy: 63.20%
3. Negative strategy: 53.24%

What a surprise! Our idea of always picking from the "fullest" tree, seems to be a good one ;-) What I also find interesting is that, even if you play the worst strategy *I* can imagine (can you come up with a worse one?), the chances of winning are still over 50%. Being a game for children, I think that's remarkable – did the creators of the game know that? Or did they just play a thousand times to find out? But if so, then the next question would be: is a thousand times really enough? We'll discuss this totally different approach in the upcoming chapter 3.

```
Analyze victory state [0, 0, 0, 0, 9] = 0.16%   0.12%   0.06%
Analyze victory state [0, 0, 0, 0, 8] = 0.93%   0.71%   0.40%
Analyze victory state [0, 0, 0, 0, 7] = 2.75%   2.18%   1.35%
Analyze victory state [0, 0, 0, 0, 6] = 5.62%   4.65%   3.12%
Analyze victory state [0, 0, 0, 0, 5] = 8.93%   7.68%   5.58%
Analyze victory state [0, 0, 0, 0, 4] = 11.71%  10.50%  8.26%
Analyze victory state [0, 0, 0, 0, 3] = 13.23%  12.37%  10.53%
Analyze victory state [0, 0, 0, 0, 2] = 13.20%  12.88%  11.87%
Analyze victory state [0, 0, 0, 0, 1] = 11.88%  12.11%  12.06%
```

Figure 3: Winning probabilities using positive, random and negative strategy

The great thing about the analytical solution is that we can tell which of the winning states yield which probability. Figure 3 shows these probabilities for the positive, random and negative strategy (per column). In the case they win, players using the positive strategy will most likely end up with 3 crows left, whereas bad players will only have 1 crow left – hence really pushing their luck!

Keep in mind: even though the game itself is highly dependent on chance, the analytical solution is not! Each time the algorithm is being run, it will yield the same result.

3 Simulation

3.1 Theory

What would be an alternative, far easier way of solving the problem above? One could simply play the game repeatedly and count the number of times the players win. The winning probability would then be:

$$\pi = \frac{Number\ of\ victories}{Total\ number\ of\ games\ played}$$

Of course, I didn't force my nephew to play the game a million times. Instead I made use of the powerful tool of simulation. The basic idea this time is to program the game itself by letting virtual players throw a virtual die until the game is over. The random effects of the die can be simulated by pseudo-random numbers. For a more detailed approach to simulation I refer to [Waldmann and Helm, 2016].

3.2 Monte Carlo Simulation in Python

As for the analytical code, you can find this code on my Github account: jonas.schumacher

Instead of analyzing the structure of the games as in section 2, I only simulated the behavior of the players using the 3 strategies described in the analytical solution 2.3. Due

to the effect of the pseudo-random numbers, each game can take a different course and end in a different result. That's why the game needs to be played many times in order to get rid of the random effect.

As you will see, the code is a lot shorter and easier than the analytical one. It only consists of the game itself and the simulation, which repeatedly starts a knew game and counts the number of victories. At the end, I added a graph to see if the winning probabilities converge.

3.3 Results

Table 1 shows the winning probabilities for an increasing number of simulation runs. Since random numbers were used to generate the game's outcome, I've used two different seeds (s = 123 and s = 54321) to be able to compare and replicate the results. The last row shows the analytical solution from section 2.6

	Positive		**Random**		**Negative**	
number of simulations	s = 123	s = 54321	s = 123	s = 54321	s = 123	s = 54321
$10 = 10^1$	80	90	70	60	70	80
10^2	74	71	66	63	56	54
10^3	68.4	66.5	65.0	64.5	54.1	51.4
10^4	67.85	68.52	63.86	63.52	53.35	53.14
10^5	68.46	68.55	63.46	63.09	53.32	53.32
10^6	68.46	68.35	63.25	63.22	53.27	53.27
10^7	68.40	68.40	63.22	63.21	53.25	53.25
10^8	68.40	68.41	63.19	63.21	53.23	53.24
comparison to analytics	**68.40%**		**63.20%**		**53.24%**	

Table 1: Simulation results

10 simulation runs don't seem to be enough to distinguish the three strategies, but after the first 1000 simulation runs, we're already getting a rough impression of the winning probabilities. Then after one million runs, the probability seems to be more and more stable: we can see three distinguishable categories and the effect of the different seed is almost negligible.

4 Final thoughts

4.1 Comparison of the analytical and the simulated solution

Figure 1 also includes a comparison between the simulated and the analytical results. As we expected (or hoped), the simulated results converge to the analytical solution. In fact, by comparing these results, I found an error in my analytical code for the negative strategy and was able to correct it. That's how these two different approaches can support each other and make the results even more credible.

4.2 Philosophical remark

I intentionally picked a very easy game for this analysis which turned out to be rather difficult instead. I find this hidden complexity especially interesting: The first thing I noticed is that there is a huge difference between **playing** a game and **understanding** its underlying structure. If we compare this game to a real-world problem or life itself, we can get an impression of how unimaginably complicated the world is. Even if we know some of the rules and are able to find a rather mediocre strategy of succeeding in it, we are still far away from understanding what's behind all of it. But if you ask me: it's a lot of fun **finding out as much as we can** about it.

4.3 Feedback

I hope you enjoyed the analysis of this supposedly easy children's game. Feel free to spread this document with a reference to my website: `https://www.jonas-schumacher.com`

And please let me know if you found any code errors or flaws in my logic and how you liked the analysis in general: contact@jonas-schumacher.com

References

Waldmann, K.-H. and Helm, W. E. (2016). *Simulation stochastischer Systeme: Eine anwendungsorientierte Einführung*. Springer-Verlag.

Waldmann, K.-H. and Stocker, U. M. (2012). *Stochastische Modelle: Eine anwendungsorientierte Einführung*. Springer-Verlag.