Lab 1: Set, the game

Adam Selker, Nick Sherman

Due March 27, 2020

1 Problem Description

The original game of Set is a game of card matching, where players try to match cards such that the properties on each card are either all the same or are all different for a given set. In the original game, there are four different properties. They were:

- Color, with values green, purple, and red
- Shape, with values diamond, oval, and squiggle
- Shading, with values open, shaded, and solid
- Number, with values of one, two, and three

The "sets" that players have to pick from cards are of size 3, and the cards must have properties that are either unique or the same for each of the different properties. An example of a "valid set" where all properties are unique for each card can be seen in 1.

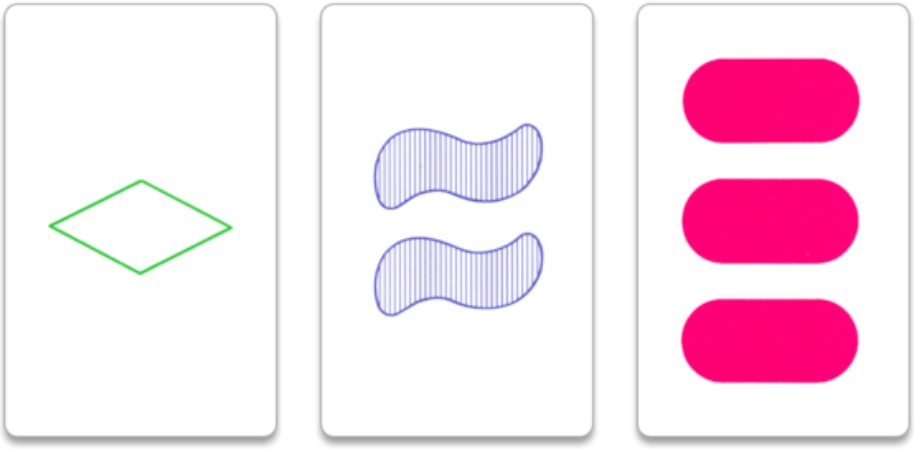

Figure 1: A sample image of a valid set in the game Set.

In the game of set, twelve cards from a deck of 81 unique cards are played face-up. From these, players search for a valid set among the dealt card, and are able to say "Set" and claim the valid set once found. The

game continues until all valid sets among the dealt cards have been claimed. The person with the highest number of valid sets at the end of the game wins.

For the duration of this paper, we are going to use the word "Set" to mention the process through which a valid set is found and proved to be valid. This is analogous to playing the game, just for a single round. We are also going to assume that the deck being used has unique cards which include every possible combination of values. (e.g. In the original game of Set there are 81 cards because it is 3^4, and there are $p = 4$ properties and $v = 3$ values for each property.

2 Proof of NP-completeness

2.1 Set is in NP

To prove that Set is in NP, we must show that, given a potential solution, we can test whether it is valid in polynomial time. This is fairly straightforward. A valid set is correct if, for each property, all of the cards share that property, or all of the cards differ in that property.

Start with a potential solution, in the form of v cards with k properties each. We can iterate through the k properties, and for each one, compare the first two cards (the order of the cards can be picked arbitrarily). If the first two are the same, we ensure that the other $v - 2$ cards are the same (v time). If not, we ensure that all v differ (v^2 time for the naive approach). This gives us a worst-case time of $k \cdot v^2$, which is polynomial with both k and v.

If the valid set is incorrect, at least one of those checks is guaranteed to fail; if it is correct, they are all guaranteed to pass. Since we can check correctness in polynomial time, the game of Set is in NP.

2.2 Set is NP-hard

To prove that Set is NP-hard, we will reduce an NP-complete problem, k-dimensional matching, to Set. We do this by constructing a set of cards such that finding a valid set in those cards corresponds to finding a perfect matching in the matching problem.

Start with the matching problem. We have $v > 0$ values, and $k > 0$ properties. The matching problem takes the form of some set C. Each element of C has k properties, and for each property, some value between 1 and v. (These will correspond to Set cards, each of which has k properties (shape, color, etc.).) We want to know whether there is some subset $M \subseteq C$ such that M is a perfect matching. This is true iff, for every property k and every value v, exactly one $m \in M$ has that value for that property. It is important to note that in our reduction we are assuming that every property will have the same number of potential values, as there are versions of k-dimensional mapping where this is not the case. This means that our solver will only solve problems with the same setup.

Next, we construct our Set problem, which will have k properties and $v + 1$ values. We make a set C' of cards, with one corresponding to each element of C. We also add another element c to C', which has values $(v + 1, v + 1, v + 1...v + 1)$. Note that c is the only element of C' for which any value is $v + 1$, since the matching problem only had v values.

Now, if there is a valid set M' within C'. If so, we can guarantee that it includes c. This is because for each property k, either all $m \in M'$ must have the same value, or they must all have different values. However, for at least one k, they must all have different values. This is because, if every category were the

same across all cards, the cards would be identical. By the definition of the game, no two cards can be identical. So, there is some k such that all $m \in M$ have different values for that property. M' has $v+1$ elements, so by the pigeonhole principle, one m must have the value $v+1$. c is the only element with any value $v+1$, so that element must be c.

c does not share any values with any other card in M'. Therefore, the valid set M' cannot have any properties which are the same across cards. All of the properties must be different between every card in M'. Again, by the pigeonhole principle, this means that for every category k, exactly one card in M' will have every value between 1 and $v+1$. We can remove c from M' to produce M, and this property will still be true, since c had the only values $v+1$; we are reduced to v pigeons in v distinct holes.

M is a subset of C, since all elements of C' (and therefore of M'), other than c, are in C. Also, for each property k and each value v, exactly one $m \in M$ has that value for that category. This is identical to the definition of a perfect matching. Therefore, M is a perfect matching on C.

If there is no valid subset M within C, then adding an element with values $(v+1, v+1, ...v+1)$ will not create a valid set in C'. This is because although there can be a unique value in the $v+1$th element, there is some conflict with prior elements which adding to the set will not resolve. In order to resolve the conflict, one or more element(s) would need to be changed in the original set C in order to create a valid subset M to which the new element could be added to create M'.

Every step of this reduction takes polynomial time at most. Apart from solving the Set problem, all we do is add a single card, and then remove it again. Therefore, since the NP-complete problem of k-dimensional matching can be reduced to the game of Set in polynomial time, Set is NP-hard.

2.3 Set is NP-complete

We have proven that Set is NP-hard, by showing that there is a polynomial-time reduction from some NP-hard problem (we used k-dimensional matching) to Set. We have also proven that Set is in NP. These are the two criteria for a problem to be NP-complete. Therefore, Set is NP-complete.

3 Implementation

For the implementation portion of the lab, we decided to implement a full solver that both finds all valid sets in a game of Set and can take a n-dimensional mapping, map it to Set, and then solve it. We encoded Set and n-dimensional mapping similarly to CNF; the text file starts off with comments (lines that start with c) then move onto the line containing any relevent information (starting with p). In this case, the "p" line includes number of variables, dimensions, and number of solutions. We then have all cards within the set, starting with a value of 0 for what value a card holds in a particular dimension. These lines do NOT end with 0, unlike CNF.

From here, we read the information in, convert it to a list of tuples, and then perform the reduction if we are solving n-dimensional mapping and solve or just solve the Set problem. The test cases that we made all are solved correctly by the solver.

Our code can be viewed here (full link: https://github.com/aselker/aa_lab1_set/tree/master). All test cases are in the folder test_cases.

When we run all of our test cases, we get the following output:

```
File name: test_cases/1.txt
The input read in is: [(2, 1, 1), (1, 1, 2), (0, 2, 2), (2, 3, 1),
(2, 3, 3), (0, 0, 0), (0, 1, 1), (2, 1, 3), (0, 1, 0), (1, 3, 2),
(2, 2, 0), (3, 3, 1)]
The number of variables (num_vars) is: 4
The number of dimensions (num_dims) is: 3
Converted to set format we get: [(2, 1, 1), (1, 1, 2), (0, 2, 2),
(2, 3, 1), (2, 3, 3), (0, 0, 0), (0, 1, 1), (2, 1, 3), (0, 1, 0),
(1, 3, 2), (2, 2, 0), (3, 3, 1), (4, 4, 4)]
No matchings found. There were supposed to be 0.

File name: test_cases/2.txt
The input read in is: [(1, 1, 2), (0, 2, 3), (2, 3, 1), (1, 2, 3),
(3, 3, 3), (2, 3, 0), (3, 3, 1), (1, 0, 2), (3, 1, 3), (1, 1, 3),
(2, 1, 1), (2, 2, 2)]
The number of variables (num_vars) is: 4
The number of dimensions (num_dims) is: 3
Converted to set format we get: [(1, 1, 2), (0, 2, 3), (2, 3, 1),
(1, 2, 3), (3, 3, 3), (2, 3, 0), (3, 3, 1), (1, 0, 2), (3, 1, 3),
(1, 1, 3), (2, 1, 1), (2, 2, 2), (4, 4, 4)]
No matchings found. There were supposed to be 0.

File name: test_cases/3.txt
The input read in is: [(2, 1, 1), (3, 1, 3), (0, 3, 0), (0, 2, 0),
(0, 0, 3), (2, 2, 0), (3, 1, 1), (0, 0, 1), (0, 1, 0), (3, 0, 1),
(1, 3, 0), (3, 2, 2)]
The number of variables (num_vars) is: 4
The number of dimensions (num_dims) is: 3
Converted to set format we get: [(2, 1, 1), (3, 1, 3), (0, 3, 0),
(0, 2, 0), (0, 0, 3), (2, 2, 0), (3, 1, 1), (0, 0, 1), (0, 1, 0),
(3, 0, 1), (1, 3, 0), (3, 2, 2), (4, 4, 4)]
Got 1 sets as solutions; there are supposed to be 1
A final matched set is: ((2, 1, 1), (0, 0, 3), (1, 3, 0), (3, 2, 2))
All possible matched sets are: [((2, 1, 1), (0, 0, 3), (1, 3, 0), (3, 2, 2))]

File name: test_cases/3d_mappingtest.txt
The input read in is: [(0, 1, 1), (1, 0, 0), (2, 2, 2)]
The number of variables (num_vars) is: 3
The number of dimensions (num_dims) is: 3
Converted to set format we get: [(0, 1, 1), (1, 0, 0), (2, 2, 2), (3, 3, 3)]
Got 1 sets as solutions; there are supposed to be 1
A final matched set is: ((0, 1, 1), (1, 0, 0), (2, 2, 2))
All possible matched sets are: [((0, 1, 1), (1, 0, 0), (2, 2, 2))]

File name: test_cases/4.txt
The input read in is: [(1, 3, 3), (1, 3, 2), (2, 2, 0), (3, 0, 0),
(1, 1, 3), (3, 1, 3), (1, 2, 1), (0, 2, 1), (1, 3, 1), (1, 0, 2),
(1, 0, 0), (3, 3, 3)]
The number of variables (num_vars) is: 4
The number of dimensions (num_dims) is: 3
Converted to set format we get: [(1, 3, 3), (1, 3, 2), (2, 2, 0),
(3, 0, 0), (1, 1, 3), (3, 1, 3), (1, 2, 1), (0, 2, 1), (1, 3, 1),
(1, 0, 2), (1, 0, 0), (3, 3, 3), (4, 4, 4)]
```

No matchings found. There were supposed to be 0.

4 Works Consulted

Chaudhuri, K., Godfrey, B., Ratajczak, D., & Wee, H. (2003). On the complexity of the game of set. http://pbg.cs.illinois.edu/papers/set.pdf

This work was used to understand the reduction to the game Set.

https://en.wikipedia.org/wiki/File:Set-game-cards.png

This website was used for the image used of a sample set.