Solving the Generalized Form of the Game of Set Efficiently

Steven Takeshita’19

Advisor: Zachary Kincaid

**Motivation and Goal**

The Game of Set was created in 1974 and published in 1991. The game consists of $3^4 = 81$ unique cards. Each card has 4 properties and one of three values. A valid set of three cards is one in which for each property, they have either the same value or all different value. At the beginning of the game 12 cards are shown, and players must locate valid sets. Once a set is found or no set exists, three new cards are added. The most number of sets collected at the ends win. Therefore, a natural extension of this problem is to determine how fast can we find sets. The problem of finding a set in which the number of properties and values are generalized has been shown to be NP-Complete, and therefore the goal of my project is to create a solver that locates a set efficiently in practice for a game with $p$ properties and $v$ values. The Game of Set is an interesting problem for dynamic programming in that when three new cards are added, the algorithm should be able to build off of previous knowledge. Though not readily obvious, a way to utilize this past information could be helpful in applying similar principles to other problems that could lend itself to dynamic programming.

**Problem Background and Related Work**

The Game of Set has been researched thoroughly for education and to motivate example a cap set, but little research exists about solving a generalized version. Chaudhuri Et. Al (2003) proved that the generalized version of Set is in fact NP-Complete through a reduction from perfect-Dimensional Matching, a well known NP-Hard problem, and determined that for the 3 value game in which p changes, the brute force algorithm yields $\theta{n^2 p}$. Even with the optimum tuning using Turan’s Theorem, they showed that the worst-case lower bound of $\roof{n^2/4 – n/2}$.

This paper yields computational complexity bounds, but does not solve the problem in practice. Solvers do exist on the internet and can be readily found. They employ a brute force algorithm to locate all Sets in a given board. For example, Steve Nolte created an online JavaScript set solver (<http://www.stevenolte.com/set/set.html>). Other solvers exist that employ image processing to allow for quickly identifying sets without manually entering in the data (<https://github.com/vimalloc/set-solver>, <https://github.com/Koff/SET-Card-Game-Matlab-Visual-Solver>). However, all these existing solvers are constrained to have four properties and three values. Therefore, the solver I will build will be able to solve the generic case of Set in which it has $p$ properties and $v$ values.

**Approach**

Generalizing the Game of Set has been done academically, but no practical solution to locate sets exists. The reduction to a NP-complete problem creates a viable solver to theoretically always verify whether a set exists, but this does not verify a set within reasonable time if we consider the the poly-time transformation and the fact that it needs to recalculate a set every time new cards are added, not using information from previous rounds to verify it faster. Therefore, my approach will use previous knowledge of the absence of sets to quickly discover sets in a new showing of cards. Set’s changing probabilities of discovering a Set also lends itself to the dynamic algorithm approach. A director of research at Google (<https://norvig.com/SET.html>)

discovered that on a fresh layout of the game with 12 cards, the ratio of games in which there exists a Set to those that do not contain a set, is 29:1. But as the game plays on, the ratio drops to 15:1, a 50% less chance of finding a set. For 15 cards, the ratio drops to 4% of its original ratio. Therefore, as the game is iteratively played in which Sets are taken out, the game becomes immensely harder. With a dynamic algorithm approach, the solver I build should be able to beat the SAT solver especially with this added difficulty as Sets are removed from the board.

**Plan**

The project will be divided into two major and one minor parts. The first is to create a randomization algorithm to create a random starting board ($p\*v$ cards) from the cards with $p$ properties and $v$ values. These cards need to be randomly selected from $v^p$ possible cards and when sets of size $v$ are iteratively removed, they must be able to retrieve $v$ new cards such that the new cards are random and without replacement. Both the SAT based solver and dynamic algorithm must be able to take the output as input.

The second part is finding a reduction to the NP-Complete problem, SAT. Then I will code the algorithm up to solve the equation and find a Set in the board.

The last part will be to create a more efficient algorithm. This algorithm will use a dynamic algorithm to speed up the search for iterative rounds. I will then compare this algorithm against the SAT solver program for varying values of $p$, $v$, and number of sets to find.

3/2 – Create randomization algorithm and begin reduction to SAT.

3/9 – Finish reduction to SAT solver and begin implementing it.

3/16 – Finish the implementation of SAT solver. Run correctness/stress tests.

3/30 – Begin work on more efficient algorithm implementation.

4/6 – Finish more efficient algorithm implementation. Run correctness/stress tests.

4/13 – Fix lingering bugs in code. Ensure randomization works with implementations.

4/22 – Finish slides for Oral Presentation and practice

4/27 – Finish 10 pages of final report

5/3 – Finish final report and send to Professor Kincaid to review

5/7 – Finish revisions and submit written proposal.

**Evaluation**

To test the randomization algorithm for choosing cards, I will randomly generate many $n\*p$ sized decks, and calculate the distribution of this starting deck. To verify that the algorithm to pick the next p cards is uniform from the remaining cards, I will find the distribution of p cards that are iteratively chosen for a random beginning deck and sample this many times. The randomization will be successful if both are uniform distributions.

With two built solvers for the same problem, I will time how long it takes to iteratively find a given number of sets. I will adjust the number of sets, values and properties. Hopefully at some levels, most likely higher number of sets, values, and properties, the dynamic algorithm solver will finish faster than the SAT based solver.

**Problem Definition**

Chaudhuri, Kamalika, Godfrey, Brighten, Ratajczak, David, Wee, Hoeteck. (2003). On the Complexity of the Game of Set.

<http://www.masterbaboon.com/2010/09/solving-the-game-set/>

--- this one solves it iteratively – sees it as a dynamic programming problem. Maybe use.

<https://www.reddit.com/r/dailyprogrammer/comments/3ke4l6/20150909_challenge_231_intermediate_set_game/> reddit literally did it…

<https://link.springer.com/chapter/10.1007/978-3-642-54423-1_3#Bib1> computational complexity fo set

Dear Professor Kincaid,

This week I wrote my proposal, outlining the approach I hope to take, a timeline, and evaluation metrics. I was able to read papers on how to reduce to NP-Complete problems but have not found any papers that reduce it to SAT. So next week, I will hopefully be able to find a way to reduce it to SAT that is easily programmable. Also, I plan to work on creating a randomization algorithm.

If you could read my draft of my proposal that’d be great. Thank you!

Best,

Steven