ParaSet Proposal

Ellen Guo (ekg2134), Ryan Xu (rx2189), Cindy Zhu (cwz2102)

1 Introduction

Set® is a card game played with a deck of 81 unique cards. Each card has four properties: color, shape, shading, and number of figures. Each property has three values as follows:

- Color: Red, Green, or Purple.
- Shape: Oval, Diamond, or Squiggle.
- Shading: Solid, Shaded, or Open.
- Number of Figures: One, Two, or Three.

A valid set consists of three cards where for each property, the cards must either all have the same value or all different values.

To play the game, deal 12 cards. Any number of players try to find a valid set as fast as they can; when they find it, the three cards are removed and new cards are added.

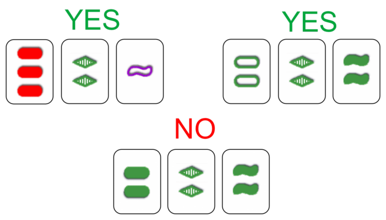

Figure 1: Examples of Sets and Non-Set. The first set (top left) has cards where the colors are different, the shapes are different, and shadings are different, and the number of figures are different. The second set (top right) has cards wehre the colors are the same, the shapes are different, the shadings are different, and the number of figure are the same. The last set is invalid because though the colors are the same, the shapes are different, and number of figures are the same, the shadings are wrong: two are solid, but one is shaded (so not all the same or different).

2 Problem Overview and Brute Force Solution

We can generalize the game of Set® by creating a game where C cards are dealt, with p types of properties that each take on v possible values. Thus, the classic game of Set uses $C = 12$, $p = 4$, and $v = 3$ (and has a deck of $v^p = 3^4 = 81$ unique cards). A card c has p properties, each of which we will denote as $c[i]$ for $1 \leq i \leq p$.

A valid set therefore consists of v cards $c_1, c_2, \ldots, c_v$ such that for every property i, either $c_1[i] = c_2[i] = \cdots = c_v[i]$ or $c_1[i] \neq c_2[i] \neq \ldots \neq c_v[i]$.

The problem that we present is as follows: **Given C distinct cards (having different properties), determine all valid sets that can be made from a subset of the C cards.**

The most obvious single-threaded approach to the problem is brute force; each combination of the C cards is be checked for validity. However, it can be observed that given $v-1$ cards, at most *one* unique card can be added that completes the set, if possible at all. We will refer to a distinct group of $v-1$ cards as a "pre-set", regardless of if it is possible to create a full set from these cards. Therefore, we can improve on the brute force solution slightly by generating all pre-sets and checking whether the required card to complete the set is present in our C dealt cards.

Note that for $v = 3$, like in classic Set®, each pre-set is a pair of cards. In this case, an optimization exists in which we can more efficiently generate the pre-sets by splitting the C dealt cards into two partitions C_1 and C_2. Then we generate half our pre-sets by taking all pairs of cards in C_1, and the second half by taking all pairs from C_2. Due to the pigeonhole principle, this ends up generating all necessary pairs to be checked (as all possible sets include at least two cards from one of the partitions). However, for $v > 3$, this optimization does not work, and we must use our original approach of generating all pre-sets from the original C dealt cards.

3 Parallelization

We will first start by implementing the sequential brute force approach described above. We can try to use a Naive Parallelization approach, where we parallelize this using dynamic partitioning, where a new spark is generated for each of the pre-sets. Then each spark will do the work within the loop, so generating the unique card (if it exists) needed to complete a valid set and then checking if this new card is in the C cards dealt. This approach might result in too many sparks being created, so we can try to control the granularity by only going parallel to a certain depth (like in the nfib example from class).

Aside from parallelizing the work for each of the pre-sets, we can also parallelize the generation of the pre-sets themselves from the C cards dealt. Using the classic game of Set® ($C = 12$, $p = 4$, and $v = 3$) as an example, we can generate $\binom{12}{3}$ unique presets by fixing the first card as the first card from C, then choosing $C - 1 = 11$ cards for the second card of the preset to create 11 new presets. Then, we can fix the second card as the second card from C, then choose $C - 2 = 10$ cards for the second card of the preset to create 10 new presets. This can continue until the eleventh card generating 1 new preset. We can also generalize this to any C, p, and v. Since the generation of the presets is independent for each of cards in C when we fix the first cards, we can do these in parallel. We also ensure there are no duplicate presets generated this way. We can dynamically partition this or also try to only go parallel to a certain depth based on the ratio of C and $v - 1$, or preset size so that we aren't creating sparks if the calculation is small.

4 Areas of Investigation

Speedup as a function of C, v, and p: One thing we're curious about is how variations in C (up to p^v), v, and p impact our speedup calculations. In particular, what if $p > v$? Or $v > p$? What about the ratio of $p : C$ or $v : C$?
When we switch to sequential: Like in the Fibonacci example from class, the level at which we switch from parallel to sequential processing would definitely affect our speedup; a similar question is how can we optimize spark production? Too many sparks (aka in the Naive Parallelization) would be bad, but so would not enough parallelization.

5 References

https://pbg.cs.illinois.edu/papers/set.pdf
SET Family Game on Amazon