CS6046: Assignment 1

THE GAME OF CATS AND DOGS

V S S ANIRUDH SHARMA, EE18B036

Indian Institute of Technology, Madras

CS6046: ASSIGNMENT 1

V.S.S.Anirudh Sharma : EE18B036

March 21, 2021

1 Introduction

This report discusses an optimal algorithm for guessing the right word in the game of cats and dogs with minimal number of guesses.

2 The Game

The Game begins with an adversary picking a word from a given set of words. This list of word contains unique 4 lettered words, each with no repeating letters. The player must now guess the word with as little number of guesses as possible.
Every guess made by the player is reciprocated with 2 numbers: Cats($C_{actual}(guess)$) and Dogs($D_{actual}(guess)$).
A Cat is an alphabet which is common to both your guess and the adversary's chosen word but is not in the same position. A Dog is an alphabet which is common to both your guess and the adversary's chosen word and also is in the same position.
The player must guess the word in minimal number of guesses.

3 The Algorithm

We have explored 3 different algorithms during the process of coming up with an optimal algorithm. Here, we discuss the optimal algorithm alone. Here,

$$\#D_a(b) = \text{Number of dogs in b w.r.t a} \tag{1}$$

$$\#C_a(b) = \text{Number of cats in b w.r.t a} \tag{2}$$

Algorithm 1 Optimal Algorithm for Cats and Dogs

Result: $x_t = y$
The adversary first chooses a word y from H, the list of all the words
$H_0 \leftarrow H$
$x_0 \leftarrow$ random choose from H_0
while $x_t \neq y$ **do**
  Get $c_t \leftarrow C_y(x_t)$ and $d_t \leftarrow D_y(x_t)$
  $H_{t+1} \leftarrow \{w : (D_w(x_t) = d_t) \text{ and } (C_w(x_t) = c_t) \text{ and } (w \in H_t)\}\}$
  $x_{t+1} \leftarrow$ random choose from H_{t+1}
  $t \leftarrow t + 1$
end

The code for the same is as follows

```
def update(H,x,c,d):
  ans = []
  for w in H:
    cw,dw = cd(x,w)
    if cw==c and dw==d:
      ans.append(w)
  return ans


def alg(y,words):
  t = 0 #number of attempts to guess
  H = words
  x = random.choice(H)
  while(x!=y):
    c,d = cd(x,y) #getting cats and dogs
    H = update(H,x,c,d)
    x = random.choice(H)
    t += 1
  return(t)
```

4 Understanding the Algorithm

For a given x_t and y (the actual word the adversary picks), we get
$c_t = C_y(x_t)$ and $d_t = D_y(x_t)$
Now we must try to keep all possible values of x_i (one of which **must** be y) in the set to pick x_{t+1} and discard the remaining.
Since we know that $C_y(x_t) = c_t$ and $D_y(x_t) = d_t$, we keep all x_i from H_t in the next H_{t+1} such that $C_{x_i}(x_t) = c_t$ and $D_{x_i}(x_t) = d_t$ and discard the rest of words, since we know for sure that this H_{t+1} will contain y. We then select the x_{t+1} randomly from H_{t+1} and this carries on until $t = t_0$ when $x_{t_0} = y$.

5 Toughest word for the algorithm

The toughest words for the algorithm have been 'HINS','KINS', 'PINS'and 'REPS', taking 16 guesses each. But this varies with execution since the algorithm involves randomly picking from the bag of words. It has been observed that despite the toughest words changing, the worst case number of guesses is almost always 16.

6 Average iterations

The number of guesses made by the algorithm averaged over each word in the given words played 10 times = 5.483967517401392. Even this value changes with execution, revolves around 5.5
The following code segment has been used to find the toughest words and the average time

```
average = 0
worst_words = []
worst = 0
worst_time = []
for i in range(n_given_words):
  T = 0
  w = given_words[i]
  local_worst = 0
  for j in range(10):
    t = alg(w,given_words)
    T += t/10
```

```
    if t>local_worst:
      local_worst = t
  if local_worst>=worst:
    worst = local_worst
    worst_words.append(w)
    worst_time.append(local_worst)
  average += T/n_given_words
```

7 Performance of the algorithm

A list of all possible 4 lettered words has been generated to understand the performance of the algorithm.
The average iterations by the algorithm is of logarithmic complexity with respect to number of words in the bag, as shown in figure 1. Here a new word is searched for in each experiment done to find the average. Also, form figures 3 and 4, we must note that even in the case of all the words, the worst we usually face is 20-22 iterations. The worst case number of guesses is given in figure 4
So basically, this algorithm guesses the 1 word out of haystack of 358800 words in at most 22 guesses and 10 guesses on an average.

8 Is this optimal?

In our algorithms, we try to make the size of set of possible right answers smaller with each guess. Sub-optimality is when there is scope to further reduce the size but the algorithm missed the opportunity.

For each guess x_t we make, the only feedback we get is c_t and d_t. The best we can do is to look further only at the words which give the same c and d with x_t as y. Once we group all such words for the next guessing round, there is no way we can make it any smaller before any further guesses.

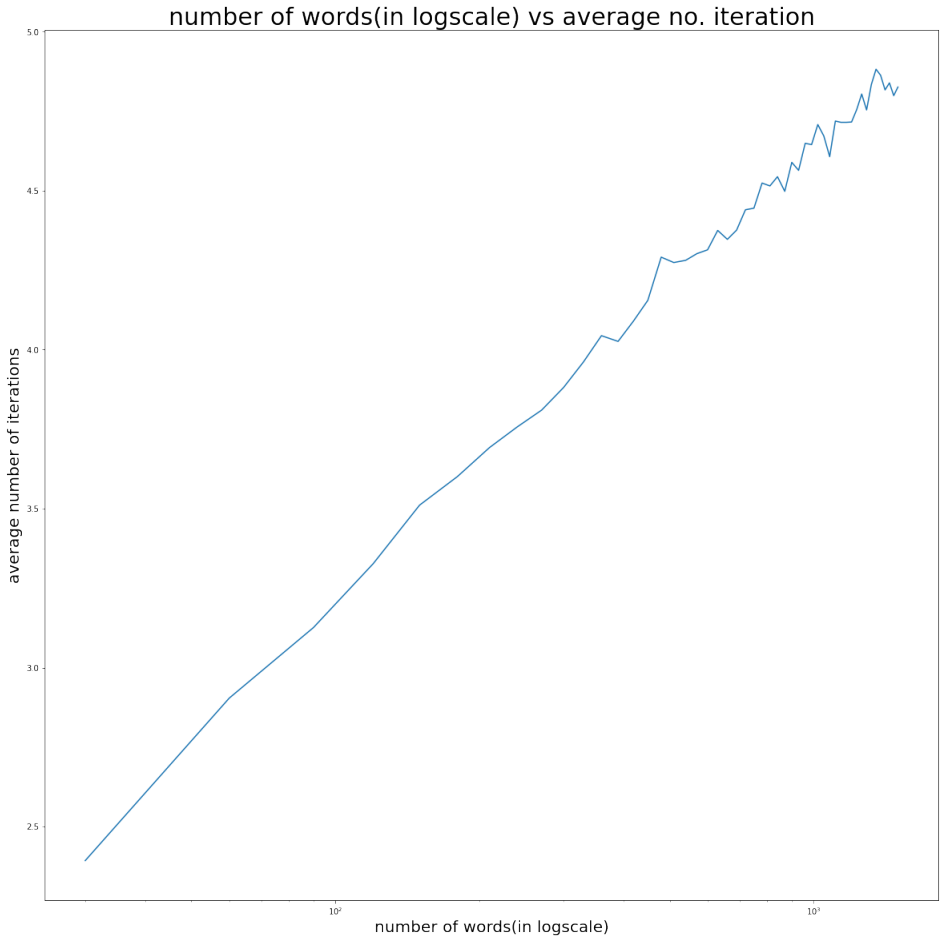

Figure 1: number of words in the bag vs average guesses

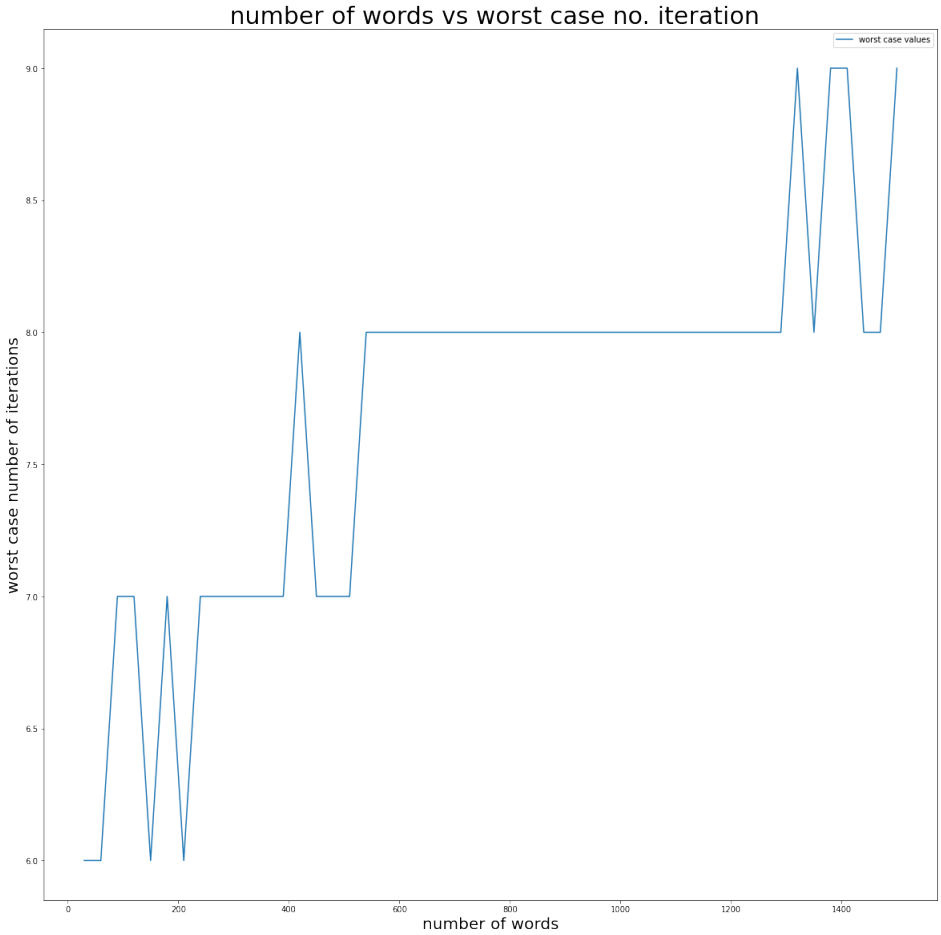

Figure 2: number of words in the bag vs worst case number of guesses

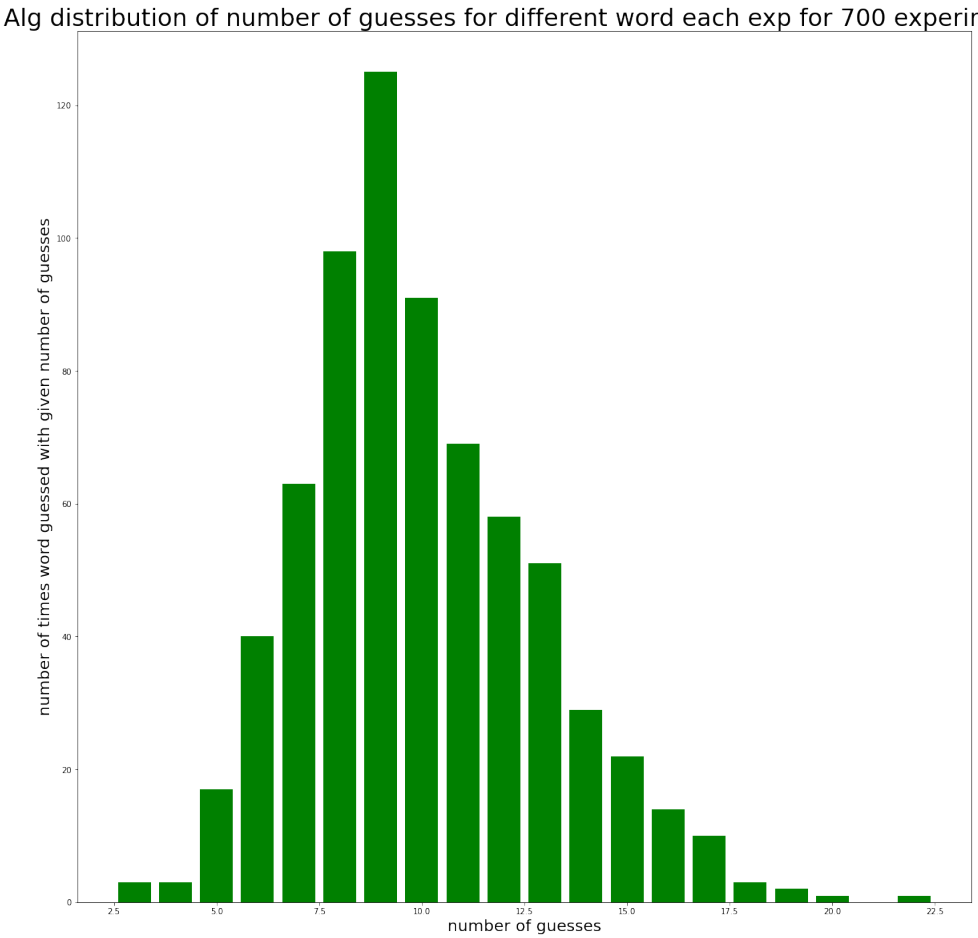

Figure 3: number of words in the bag vs average guesses

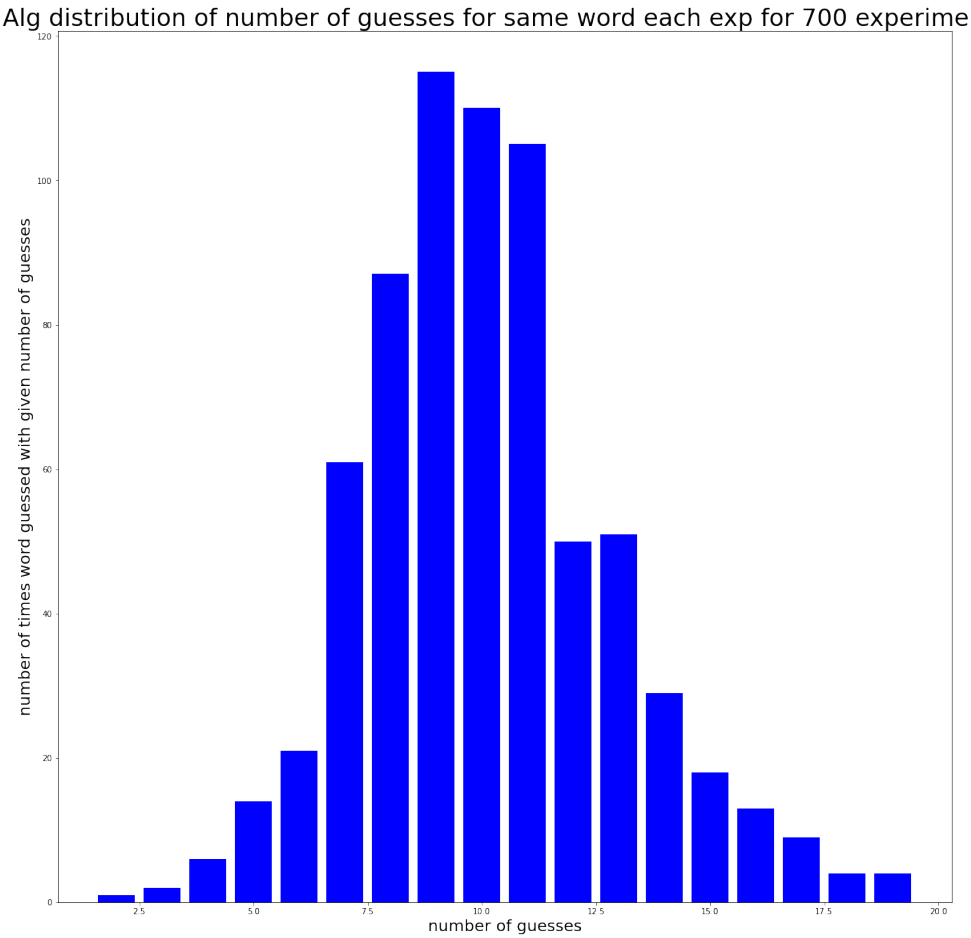

Figure 4: number of words in the bag vs average guesses