MATH 389: Twenty Questions

Alex Cole, Ajay Pillay, Anourag Shah, John Gitterman

April 26, 2024

1 Introduction

Let a and b be two positive integers where $a, b > 2$. Two players, A and B, play the following game:

1. Each player secretly chooses and writes down a number. Player A picks an integer n with $n \in [1, a]$, and Player B picks an integer m with $m \in [1, b]$.
2. Both players take turns (starting with player A) asking *yes* or *no* questions about the other player's secret number.
3. Step 2 repeats until either Player A or Player B identifies their opponent's number.

We can consider two variations of the game − one where both players are allowed to lie once and one where both players must answer all questions truthfully.

This paper details our explorations into the game as we played it among ourselves and discovered how the game unfolded in various scenarios such as when $a > b$, $a = b$ and $a < b$. We aimed to uncover patterns in the game and to devise optimal winning strategies under the different versions of the game. In Section 3, we discuss the version of the game where lying is not allowed. In Section 4, we discuss the version of the game where switching your chosen number is allowed. In Section 5, we discuss the version of the game where both players are allowed to lie once.

1.1 Acknowledgments

We would like to thank Timothy Cheek as well as the rest of the MATH 389 staff for their mentorship and guidance throughout this project.

2 Definitions

Definition 1. The values of $a, b \in \mathbb{N}$ are considered to be the upper bounds from which Player A and Player B respectively can pick within the natural numbers.

Definition 2. The basic game is the version of the game is where Players A and B choose a number unknown to the other player in their respective bounds.

Definition 3. The changing game is a version of the game where both players are allowed to switch their number once and must notify the other player they have done so. A player can say they have changed their number and retain the same number they initially chose.

Definition 4. The lying game is a version of the game where both Player A and Player B can answer untruthfully to their opponent's question once per game.

Definition 5. Player A's tree is the tree starting at range $[1, a]$ that Player B asks questions to narrow the range.

Definition 6. Player B's tree is the tree starting at range $[1, b]$ that Player A asks questions to narrow the range.

3 Initial Exploration

Below describes some of the algorithms that were explored for playing the game.

3.1 Combinations of Questions

First, we explored asking questions at will that went beyond standard "higher or lower" questions. These questions included:

- Is the number even/odd?
- Is the number prime?
- Is the number k digits long?
- Does the number end with integer k?
- Is the number greater than or less than an integer k?
- Is the number divisible by an integer k?
- Is your number some integer k?

Where k is some arbitrary integer contained within the range $[1, a]$ or $[1, b]$ respectively, depending on the player asking the question.

3.2 Binary Search

The next strategy we explored was a binary search. This algorithm consists of halving the range of values iteratively until there are only two values left. With this strategy, we were able to guarantee that with each successive guess, given the current range of values $[n]$ (where $[n] = \{1, \ldots, n\}$), we could eliminate at least $\lfloor \frac{n}{2} \rfloor$ options. This could be done by asking the question: "Is your number greater than k?" where $k = \lfloor \frac{n}{2} \rfloor$. This algorithm is efficient

compared to arbitrary guessing and runs in $\lceil \log_2(n) \rceil$ iterations, depending on the initial range $[n]$.

3.2.1 Example

Consider a game where Player A picks $m = 373$ with the limit being $a = 1000$. For brevity, we will show the sequence of questions Player B asks Player A in order to guess their number.

1. Is the number greater than 500? $\rightarrow$ No (New range [1,500])
2. Is the number greater than 250? $\rightarrow$ Yes (New range [251,500])
3. Is the number greater than 375? $\rightarrow$ No (New range [251,375])
4. Is the number greater than 325? $\rightarrow$ Yes (New range [326,375])
5. Is the number greater than 350? $\rightarrow$ Yes (New range [351,375])
6. Is the number greater than 362? $\rightarrow$ Yes (New range [363,375])
7. Is the number greater than 368? $\rightarrow$ Yes (New range [369,375])
8. Is the number greater than 372? $\rightarrow$ Yes (New range [373,375])
9. Is the number greater than 374? $\rightarrow$ No (New range [373,374])
10. Is the number greater than 373? $\rightarrow$ No **(Number is 373)**

We observe that Player B takes 10 turns to successfully find Player A's number. This efficiency holds for any number in the range through the binary search algorithm, as the maximum number of questions that would need to be asked is $\lceil \log_2(1000) \rceil = 10$.

We discuss the binary search method in greater detail in the subsequent sections.

3.2.2 Expected Number of Questions

To determine the efficiency of binary search more accurately, we sought to find a way to determine the expected number of questions needed to guess a number within a given range using binary search. We used a decision tree structure to visualize how splits impact the number of questions needed for each number within an interval. In the following figure, it is important to note that the depth of a node in the tree represents the number of questions needed to reach it. Thus, for any leaf node, which in the following figure represents singular numbers, its depth is the number of questions needed to reach it.

Looking at Figure 1, we can see that 6 numbers reside at a depth of 3 in the tree and 4 that reside at a depth of 4 in the tree. Thus, the expected number of questions for an arbitrary number within the tree can be calculated like so in this case:

$$3 * \frac{6}{10} + 4 * \frac{4}{10} = \frac{34}{10} = 3.4$$

It is important to note here that the decision to make intervals in the right subtree of an interval hold more numbers in the case of an uneven split does not affect the expected number

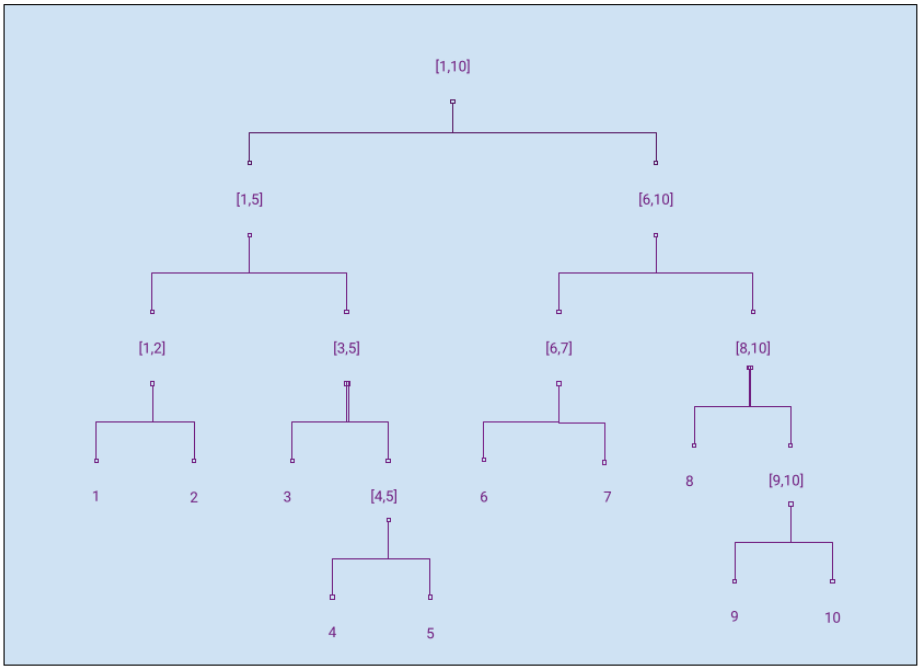

Figure 1: Decision tree for searching from 1-10

of questions. For example, in this case, if you were to split the extra numbers into the left subtree, then the numbers 1 2 6 and 7 would be at a depth of 4, while 4 5 9 and 10 would be at a depth of 3. In other words, the proportion of numbers at each depth does not change, thus neither does the expected number of questions since this is for an arbitrary number.

Theorem 1. *The expected number of questions to find an arbitrary number in a range of numbers between 1 and n using binary search is* $\lfloor \log_2 n \rfloor + \frac{2*(n-2^{\lfloor \log_2 n \rfloor})}{n}$.

Proof. For an arbitrary number range between 1 and n, we need to find the number of binary splits necessary to get sub-ranges of size 1 or 2.

Consider the largest power of 2 that n is greater than, call this y, for which the following is true:

$$y = 2^{\lfloor \log_2 n \rfloor}$$

Next, consider a binary tree that is constructed in such a way as Figure 1 for this value y is going to have a height of $\lfloor \log_2 n \rfloor$. In this binary tree, the final level of the tree is full, thus every node at the final depth would consist of a single number, as there are an equal number of leaf nodes and numbers within the range of 1 to y.

Now, consider how for the number n, at this same depth of $\lfloor \log_2 n \rfloor$, it is going to have several nodes which consist of two numbers. To be exact, it will have $n - y$ nodes that have ranges of two numbers rather than one. These $n - y$ ranges of two require an additional level down to reach a singular number. Since each range consists of two numbers, that means that there are $2 * (n - y)$ numbers total within the range [1, n] that require one extra question to be found. Thus, we can calculate the expected number with the following steps.

First calculate the fraction of numbers that require an extra question:

$$\lfloor \log_2 n \rfloor + 1 * p \tag{1}$$

Where p, based on earlier observations, is equal to:

$$\frac{2 * (n - y)}{n} \tag{2}$$

Plugging everything back in and replacing y, we get that the expected number of questions is equal to:

$$\lfloor \log_2 n \rfloor + \frac{2 * (n - 2^{\lfloor \log_2 n \rfloor}}{n} \tag{3}$$

□

3.2.3 Probability of a given player winning

With the binary search strategy, once the opposing player reaches a point where they're guaranteed to guess the number on the subsequent move, it is better for the current player to guess the other player's number instead of continuing with the binary search. This gives rise to the following theorem.

Conjecture 1. *If $b = a + k$ for some $k \in \mathbb{N}$ where $k > a$, $\mathbb{P}[A \text{ wins}] \approx \frac{a}{2a+2k}$.*

Proof. Fix $a \in \mathbb{N}$ and let $b = a + k$ for some $k \in \mathbb{N}$. Then, we know that the height of Player A's tree is $\approx \lfloor \log_2(a) \rfloor$ and the height of Player B's tree is $\approx \lfloor \log_2(b) \rfloor$.

Player A goes first. The number of questions Player B needs to ask before a guaranteed win is $\lfloor \log_2(a) \rfloor - 1$. After Player B asks the $(\lfloor \log_2(a) \rfloor - 1)$-th question, Player A should guess Player B's number uniformly at random from the remaining range of numbers Player A is searching in.

Since Player A has gone through $\lfloor \log_2(a) \rfloor - 1$ questions, the range of numbers Player A has

left to guess from is given as:

$$\begin{aligned}\frac{b}{2^{\lfloor \log_2(a) \rfloor - 1}} &= \frac{a+k}{2^{\lfloor \log_2(a) \rfloor - 1}} \\ &\approx \frac{a+k}{\frac{a}{2}} \\ &= 2 + \frac{2k}{a} \\ &= \frac{2a+2k}{a}\end{aligned}$$

So, the probability that Player A wins is precisely:

$$\mathbb{P}[A \text{ wins}] \approx \frac{1}{\frac{2a+2k}{a}} = \frac{a}{2a+2k} = \frac{a}{2b}$$

And hence the probability that Player B wins is:

$$\mathbb{P}[B \text{ wins}] \approx 1 - \mathbb{P}[A \text{ wins}] = \frac{2b-a}{2b}$$

□

Corollary 1. *From Theorem 2, a symmetric argument holds when $a = b+k$ for some $k \in \mathbb{N}$, where in this case:*

$$\begin{aligned}\mathbb{P}[A \textit{ wins}] &\approx \frac{2b-a}{2b} \\ \mathbb{P}[B \textit{ wins}] &\approx \frac{a}{2b}\end{aligned}$$

3.3 Exponential Search

Aside from attempting a binary search to find the number, we also explored applying the exponential search technique to find the number. If we seek to find some integer $m \in [1, n]$, the exponential search algorithm is applied as follows:

1. Initialize $i = 1$.
2. Check if $m \in [2^{i-1}, \ldots, 2^i]$. If not, increment i and re-check until the range has been found.
3. Perform a binary search on the range $[2^{i^*-1}, 2^{i^*}]$ to locate m, where i^* is the final value of i.

Where we omit the bounds checking before exceeding n. If the upper bound of our range on an iteration exceeds n, we just replace 2^{i^*} with n. This is due to the fact that we should not search above n as the number to guess cannot be outside of this range. The number of iterations is $\lceil \log_2(i) \rceil + \lceil \log_2(k) \rceil$ where i is the position of the target value in the range, and k is the final resultant range that binary search is run on.

3.3.1 Example

We will apply the exponential search to the same example as before, with Player A picking $m = 373$ and the limit being $a = 1000$. Similarly, for brevity, we will show the sequence of questions Player B asks Player A in order to guess their number.

1. Is the number in [1,2]? → No (New range [2,1000])
2. Is the number in [2,4] → No (New range [4,1000])
3. Is the number in [4,8] → No (New range [8,1000])
4. Is the number in [8,16] → No (New range [16,1000])
5. Is the number in [16,32] → No (New range [32,1000])
6. Is the number in [32,64] → No (New range [64,1000])
7. Is the number in [64,128] → No (New range [128,1000])
8. Is the number in [128,256] → No (New range [256,1000])
9. Is the number in [256,512] → Yes (New range [256,512])
10. Is the number greater than 384? → No (New range [256,384])
11. Is the number greater than 320? → Yes (New range [321,384])
12. Is the number greater than 352? → Yes (New range [353,384])
13. Is the number greater than 368? → Yes (New range [369,384])
14. Is the number greater than 376? → No (New range [369,376])
15. Is the number greater than 372? → Yes (New range [373,376])
16. Is the number greater than 374? → No (New range [373,374])
17. Is the number greater than 373? → No **(Number is 373)**

We observe that Player B takes 17 turns to successfully find Player A's number. Note that we performed the initial exponential search for $\lceil \log_2(373) \rceil = 9$ iterations and then performed the final binary search for $\lceil \log_2(256) \rceil = 8$ iterations giving us a total of 17 iterations. Comparing this to the binary search method in section 3.2, we can see that this strategy took 7 more iterations, and is therefore not as efficient for this range and the m selected, although this may not always be the case.

3.4 Comparing Exponential and Binary Search

Although the exponential search resulted in more questions needing to be asked in the example discussed, the exponential search does have its merits if the initial search halts earlier. That is, if the number appears in one of the earlier queried ranges rather than the final queried range. Certain choices of k could result in the exponential search requiring

fewer questions than a binary search. However, a binary search is still preferable in general due to assumed uniform randomness in the opponents' choice of a number within the range.

4 Introducing Switching

Now consider a version of the game where each player is allowed to switch the number they initially picked throughout the match. While this does not change the overall strategy that is stated above, there is an optimal time to switch the number.

The optimal time for both players to switch their depends on what a, b are in a game where both players are using binary search as the strategy.

Conjecture 2. *As $a, b \to \infty$ then the best time to switch ones number increases with regards to the amount of questions that have been asked.*

A natural question that arises is whether it matters what number each player switches to. The answer is yes. The player should change the number away from the original as the opponent can ask "Did you switch your number?" In the case where the answer to this question is "no", the opposing player would win on the next turn.

5 Introducing Lies

Now consider a version of the game where each player is allowed, but not required to use, a single lie throughout the match. This changes the outlook of the above strategies as a lie could drastically throw off a player's guesses with the binary search or exponential search methods by having them zone in on the wrong range of values.

5.1 Brute Force Binary Search

The first strategy we explored is an alteration of the binary search algorithm described above. In this strategy, we impose binary search but ask every question twice to detect if the opponent is lying. If the opponent gives the same answer both times then we know they are telling the truth. Otherwise, we must ask the question a third time to find out which of the previous two answers is the truth. Once we have caught the opponent lying we can proceed with the ordinary binary search algorithm as described above.

This strategy has a similar expected number of questions to binary search being $2*\lceil \log_2(a) \rceil + 1$ iterations in the worst case. This comes from asking each question twice, as well as an extra turn to detect the lie. It is beneficial for the opponent to use their lie at the end of the game to create this worst-case complexity. This is because once the lie has been detected, the player can proceed with a normal binary search which is twice as fast.

To demonstrate the importance of using the lie at the end, if a player uses their lie on the first turn the complexity of this algorithm becomes $\lceil \log_2(a) \rceil + 2$ iterations, as the player will have to ask the first question three times, then proceed with normal binary search. When applying these two complexities to the example in Section 4.4, we can see that using the lie

at the end of the game would take your opponent 21 turns, while using it at the beginning would take them 12 turns.

5.2 Repeating the k-th question

Another strategy we explored was if the player modified the brute force binary search strategy by repeating every k-th question for some $k \in \mathbb{N}$ and reasoning through the previous questions and answers and backtracking through the questions if necessary. We will illustrate this strategy via an example from Player A's perspective with $k = 3$. Consider a game where $b = 1,000$ and $m = 700$ (the number that Player B chose).

1. Is the number in [500,1,000]? $\rightarrow$ Yes (New range [500,1000])
2. Is the number in [750,1,000] $\rightarrow$ Yes (New range [750,1000])
3. Is the number in [875,1,000] $\rightarrow$ No (New range [750,874])
4. Is the number in [875,1,000] $\rightarrow$ No (New range [750,874])
5. Is the number in [750,1,000] $\rightarrow$ No (New range [500,749] or [750,874])
6. Is the number in [750,1,000] $\rightarrow$ No (New range [500,749])

After question 3, Player A repeats question 3. Since the answers are the same, we now have to walk through the logic up to that point and determine if the lie could've been used. If the lie was used on question 1, then the answer to question 2 must've been a No. Thus, Player B told the truth on question 1. Then, Player A asks question 2 again (backtracking). Since the answers to question 2 and 4 were different, this meant that Player B lied on one of those answers, but Player A did not have sufficient information to determine *which* question Player B lied on. Thus, Player A had to ask that question again to find that Player B lied on question 2, and thus the new range that Player A needs to search in is [500,749]. Now, Player A can continue with the brute force binary search strategy without having to repeat every 3rd new question since the lie was used.

In this strategy, if the lie ends up being saved (and hence never used), at worst this results in the same number of questions being asked as the original strategy.

5.3 Asking if the player has lied after every k-th question

This strategy modifies the previous strategy. Instead of repeating the k-th question, the player instead asks, "Have you used your lie?" and then backtrack if necessary. Similarly, we will illustrate this via an example. Consider a game where $b = 1,000$ and $m = 700$ (the number that Player B chose) and $k = 3$. From Player A's perspective, they will ask the same questions as before:

1. Is the number in [500,1,000]? $\rightarrow$ Yes (New range [500,1000])
2. Is the number in [750,1,000] $\rightarrow$ Yes (New range [750,1000])
3. Is the number in [875,1,000] $\rightarrow$ No (New range [750,874])

Now, Player A asks, "Have you used your lie?". Now, there are several outcomes as illustrated in the decision tree in Figure 2.

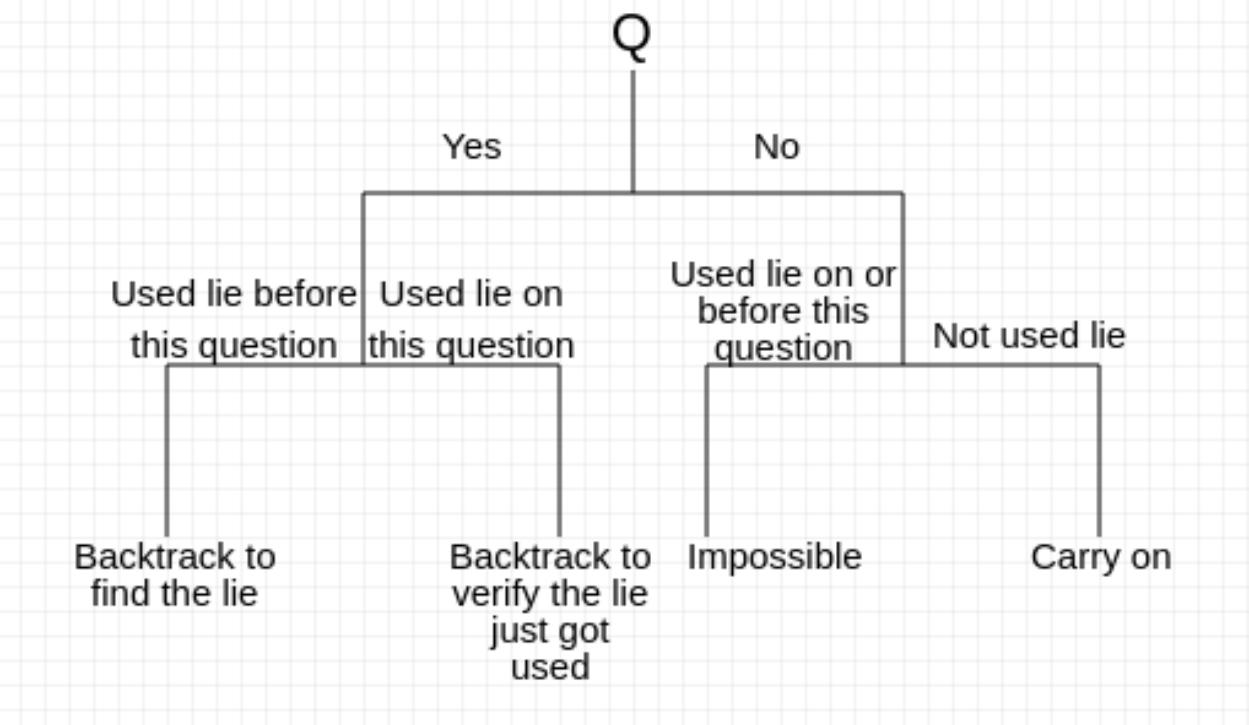

Figure 2: Decision tree for detecting the lie

The process of backtracking to find the lie or verify that the lie just got used is identical to the previous strategy, and this only happens when the other player answers "Yes" to this question. If the player answers "No", we can definitively continue with the standard binary search strategy for the next k questions. In particular, the player cannot say "No" to the question and simultaneously use their lie on this question if it were not used before. Saying "No" while intending to use their lie on this question is paradoxical since the true answer would be "No" since they did not use their lie before this question, which would imply that they are telling the truth and not lying, thus the lie could not be used on or before this question if the response is a "No".

6 Further Explorations

- How would asking more arbitrary questions such as "Does your number have the letter E in it?" change the strategy of each player?
- How could the game be played with more than two players?
- When does it become optimal to ask questions that do not split number groups 50-50?
- What happens if we allow questions with more than two possible answers? How are the probabilities effected?

- What happens to the probability of each player winning if we allowed snake questioning? i.e if Player A asked the first question then Player B would ask the next two questions and so on.
- What happens to the probability of each player winning if a player can each asking questions while the answer is "yes"? I.e if Player A asked the first question and the answer is "yes", then Player A can continue to ask another question on the same turn.