The performance of a robust Bayesian approach to the 20 questions game under different patterns of noise

Anton Donle[2704950]

Vrije Universiteit Amsterdam, De Boelelaan 1105, Netherlands

Abstract. A robust query algorithm for playing the 20 questions game with noise is introduced. The goal is to find the correct entity, even if the input contains wrong answers. This algorithm achieves robustness by making use of a Naive Bayes Classifier for calculating entity probabilities and for making a prediction. It combines this with an entropy formula, which calculates the split which provides the most information, based on the probabilities. An experiment investigates to which extent the performance of this approach depends on the pattern of input noise. The results show that wrong answers at the start lead to worse performance than wrong answers at the end. Moreover, they show that the fewer wrong answers are given, the less the performance depends on their position and pattern. The results give rise to multiple phenomena, which could only be understood in parts and further research is necessary to fully understand them. The novel approach handles noise effectively but could be further improved by assigning increasing weights to the answers throughout the game.

Keywords: 20 Questions Game · Knowledge Graph · Robustness.

1 Introduction

In today's data-driven society, where large amounts of information are accessible to a wide audience, the ability to efficiently and accurately query for information has become increasingly important. Developing game-like approaches to database querying can be used to improve the way users can retrieve specific information from databases. This paper focuses on designing a search algorithm that can handle input errors arising from different sources, such as the closed world assumption (see Section 2.3), imperfect knowledge of the user, and missing information in the knowledge base.

Algorithms and computer programs are not inherently resilient to errors. In case of malfunctions or when given wrong input by the user, algorithms mostly fail to find the right answer. This issue is particularly critical in the context of knowledge graphs due to their inherent incompleteness. knowledge graphs lack certain information, and the modeling choices affect how entities are linked to each other. Consequently, users can struggle to find the information they are looking for.

The 20 questions game is a game where players try to guess which entity the other person is thinking of by asking up to 20 yes or no questions before making a guess [17]. Ulan's game is a variation of this game, where the person answering the questions is allowed to give wrong answers [15]. In this paper the knowledge graph will be queried for attributes of entities and the answers to these questions are used to gather information about which entities the player might think of.

A regular non-robust version of the algorithm would aim to split the dataset in half on each question to maximize the estimated information gain. It might ask "Is the thing you are looking for of **type human**?". If the other player answers with yes, it would discard all entities that are not human. In the robust version, non-human entities are not eliminated but instead assigned a lower probability score. This approach ensures that potentially relevant entities are still considered. Not all attributes are as straightforward as **is human**. For example the attribute **speaks language: English** is different from the attribute **speaks language: American English**. So by affirming the question one might accidentally rule out all people that are Americans.

Working with probabilities introduces a new challenge because the search space does not reduce after each question. To overcome this, novel approaches have to be used to make a prediction and to calculate information gain without reducing the search space. In this paper, we introduce an algorithm that combines a Bayesian approach for modeling probabilities and a weighted entropy formula to measure and maximize information gain on each question. In Fig. 1 the conventional algorithm and the robust algorithm are visualised with a simple example of geometric shapes, where for the robust algorithm the color in the shape indicates its probability.

In order to analyze the performance of the algorithm under different patterns of randomness the algorithmic formulas will be kept constant while exposing the algorithm to different patterns of wrong information, referred to as noise. This introduces the research question:

To what extent does the performance of a robust Bayesian approach for a 20 questions game algorithm depend on the patterns of the noisy input?

To answer the research question an engine and a bot were developed. The engine uses the Yago data set [12] in combination with the FB15K database [8], reduced to the 134 most famous human entities according to Wikipedia in 2021. The bot exposes the algorithm to different patterns and numbers of wrong answers. The amount of correctly identified entities out of those 134 is measured.

Applications of this robust querying algorithm might be when users are trying to search a database but can not recall the name of the entity or do not know what exactly they want. It could also be beneficial when there are limited search results and the query is defined too narrowly to find any or enough relevant results. Identifying patterns that the algorithm can deal with well, and where it struggles, could be used to improve it.

In an error-free environment, the conventional algorithm would reduce the search from a scan over the whole database to a binary search. However, in our approach, efficiency is sacrificed for higher resilience. Instead of attempting to analyze or minimize the number of questions required for the engine to make a confident guess, it will continue to ask questions, allowing for earlier mistakes to be rectified. The engine will only make a guess after asking the full 20 questions.

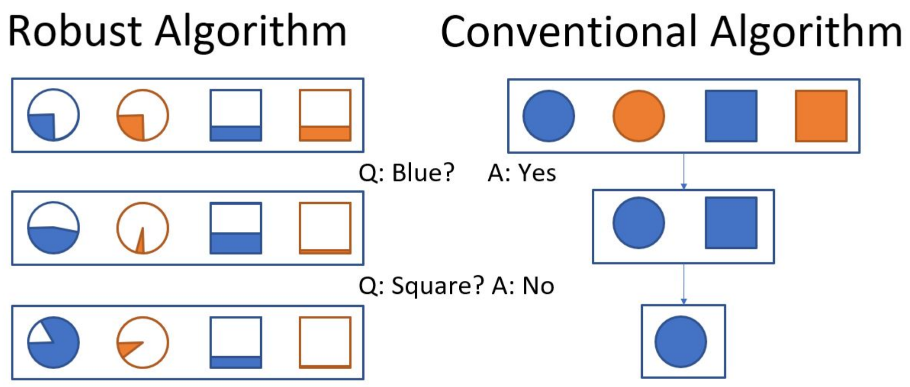


Fig. 1. Visualisation of conventional and robust algorithm

2 Related Work

The idea of introducing noise to the 20 questions game for guessing a number has been conceptualized and allegorically solved.

In his book, Ulam proposed a variation of the 20 questions game where the answerer is allowed to lie in one or two cases [15]. He proposes picking a number in the space of 2^{20} and introduces the question of how many more questions it would take with a binary search to find the correct result. In the past years, researchers have found complete algorithms that can account for wrong answers. The game is solvable in a setting where there are always perfect splits. This always works, if the number of wrong answers is less than a third of the total answers [2]. Their algorithm makes use of quantum modeling and works in one phase to narrow down the search space and a second phase to distinguish between the most likely entities.

The complete algorithm proposed by Ben-Or et. al [2] is related to our domain, but not fully applicable. The main difference is that their research concerns a theoretical mathematical setting and we are working in an applied domain, limited to the knowledge in the knowledge base. So it is not possible to achieve the same results and to find a complete algorithm that always solves the game. However, we can make use of a similar Naive Bayes assumption for probability modeling.

2.1 Theoretical Background

The theoretical basis for the noisy 20 questions game has been researched in-depth and different strategies to deal with the environment have been proposed [4, 9, 10, 16].

Jedynak et. al [9] have proved that for playing the 20 questions game, reducing the entropy maximally for every step is also optimal over the whole course of the game. Thereby they are showing that greedy algorithms can perform optimally in this setting. Chung et. al [4] have found that in a theoretical noisy 20 Questions problem, a wrong answer in the MSB (most significant/leftmost bit) provides a much larger estimation error than an error in the LSB (least significant/rightmost bit). To work around this, they propose a variety of means to achieve unequal error protection. Unlike the previous work, they state that adaptive models greedily minimizing entropy work well in practice but have no theoretical guarantee to work.

In our model, we are relying on previously described mathematical analysis. Building on the proof of Chung et. al [4] we are using a fully greedy algorithm that does not look ahead more than one step because it is proven to still be optimal. Our model is similar to an adaptive algorithm in a real-world scenario, similar to the adaptive bisection policy introduced by Chung et al. [4]. In this paper, we will investigate to what extent the MSB/LSB is comparable to the first and last question and it will be analyzed to what extent Unequal Error Protection (UEP) could increase the performance of the algorithm. Concluding, our approach involves building on mathematical insights and incorporating established formulas while introducing novel methods to develop a more robust algorithm, while exploring the potential of UEP in this particular domain.

2.2 Real World Applications

Real-world applications of the 20 Questions game have been investigated and established by researchers in different domains.

Wu et al. made a "Guess Who?" application for their chat bot, based on prior probabilities that are learned by a machine learning model [18]. For generating questions, they use the Shannon Entropy and a first-order Markov assumption to avoid asking very similar questions twice in a row. To make the game less repetitive, they chose to ask a random question out of the best 10. In the domain of knowledge bases Alkaed et. al [1] have investigated how different probabilistic approaches perform on a larger version of the Yago 4 dataset. To maximize information gain, they counted the occurrences of an attribute over the whole dataset and combined this with how many previously asked questions those attributes co-occurred with. Questions were generated based on which attributes had the largest sum. The prediction was made based on which entity matched the given answers the best. Some shortcomings in this work were that it was not generalizable and that it only analyzed the performance under a very general noisy input.

Suresh et al. have made an application of the 20 questions game to recommender systems [14]. Due to the real-world nature of the application, they are not able to guarantee perfect splits. Their work is in the context of e-commerce, so the entities are assigned prior probabilities derived from user preferences. They make use of greedy naive Bayes for entity probability modeling with the objective of maximizing information gain in at most 5 questions. A challenge they are facing is that generating questions that produce better splits are harder to understand by the users, while their goal is to filter the results best suitable to a user's preferences.

While there are many similarities between previous works on applications of the 20 questions game with our application, all of them have some key differences to this work. Suresh et. al work in a similar imperfect domain [14]. They also make use of greedy naive Bayes for calculating probabilities, with the exception that all entities are assigned a probability of 0 if they don't match the criteria. Hence, our focus diverges from theirs: while they aim to gain as much information as possible with a minimal number of questions, we are trying to gain robust information without reducing the number of questions.
Wu et al. are using a first-order Markov [18], in our algorithm instead, all previously asked questions are considered by taking into account their probabilities for calculating the entropy. For example, a question that has a high similarity with a previously asked question will result in a low entropy loss. In our approach, unlike in the work of Alkaed et. al, more precise statistical methods were used. Instead of adding a flat value, we use naive Bayes and instead of taking the largest sum, the overall entropy loss is calculated, for all possible questions [1]. This work aims to achieve more generalizability and focuses on analyzing specific patterns of randomness.

2.3 Closed World Assumption

As discussed by Cortes-Calabuig et al. [5] the default for working with knowledge bases is an Open World Assumption in order to make up for the incompleteness of the knowledge graph.

An Open World Assumption entails that statements that are not in the knowledge base are assumed to be unknown rather than wrong. Working with a Closed World Assumption (CWA) means that every attribute that is not in the knowledge base is assumed to be wrong. In this paper, the CWA will be used, so the probabilities of all entities can be adjusted after each question, even the entities without specified attribute. Otherwise gaining information and narrowing down the search space would be only half as efficient. This approach will embrace the shortcomings of the CWA and attempt to find robust workarounds. However, this is much more relevant in the context of interacting with a different agent like a human player, since both the algorithm and the answering bot are taking their information from the same knowledge base.

3 Methodology

In this paper, the terms Bayesian algorithm and engine will be used to refer to the algorithm that generates questions and adjusts the probability of the entities in order to solve the 20 questions game. The game can be played by either a human or a bot, but for the purposes of conducting statistical testing of the algorithmic methods, this paper focuses on the automated playing of the game. The term bot will be used to refer to an automated player of the game. It picks an entity and receives all information about that entity. It interacts with the engine by answering questions and after 20 questions it checks whether the engine predicts the correct entity.

3.1 Knowledge Graphs

The information about entities is stored in a knowledge graph, which consists of 134 entities that are tagged as popular entities. In total, there are 28,000 statements. The engine generates questions by utilizing what we will refer to as 'attributes'. An attribute is a combination of a predicate and an object. For instance, the attribute **type human** was utilized to identify entities that satisfy the query pattern **?entity rdf:type yago:human**. Executing the query results in a subset of entities that possess the attribute **type human**. All other entities that do not have this attribute are defined to not be of **type human**, due to the Closed World Assumption (see 2.3).

The engine uses SPARQL queries to retrieve all available entities and to retrieve the corresponding entities for each attribute. The bot sends a query to retrieve all attributes of the chosen entity to answer the questions. Working with this specific knowledge graph introduces a level of unreliability in experiments because the entities have a high variance in attributes. The number of attributes ranges from 8 (DMX rapper) to 76 (Queen Elizabeth), with an average of 25.8, a STD of 12.1, and a variance of 146.8. The exact distribution of attributes is visualized in Figure 2.

All entities share 2 attributes so those cannot be used to gain any information. The imbalance and lack of information make guessing some entities a much more complex task than others. Some entities have few attributes, and those attributes are relatively unique, so it is much harder to distinguish between those 'edge' entities. In order to make up for this and to reduce the amount of randomness in the experiments, all bots will play the game with every entity.

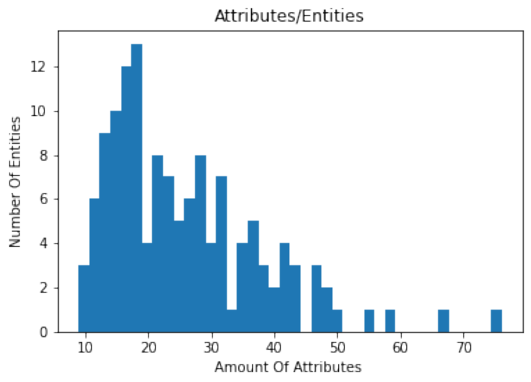


Fig. 2. Attribute Distribution

3.2 The Bayesian Algorithm

In this section the following mathematical notations will be used:

- $\mathcal{E}$ the set of all entities
- E a specific entity E
- X a specific attribute X
- $\mathcal{X}$ the set off all defined attributes
- $\mathcal{X}_n$ the n-th defined attribute
- $\mathcal{E}_{\neg X}$ the set of all entities without attribute X
- $\mathcal{E}_X$ the set of all entities with attribute X
- $|\mathcal{X}|$ the number of defined attributes

Entity Probability Modelling and Classifying with Naive Bayes For modelling the probability of each Entity Naive Bayes Classifier was used. The relative likelihood of each entity $P(E)$ is initialized uniformly with (1/134). For this, the Python dictionary data structure is used. After each question, a new attribute X is defined and the Naive Bayes Classifier calculates the probability of each entity E based on the set of all previously defined attributes $\mathcal{X}$ using Bayes' Theorem [3,11]:

$$P(E|\mathcal{X}) = \frac{P(E)P(\mathcal{X}|E)}{P(\mathcal{X})} \tag{1}$$

To compute the likelihood $P(E|\mathcal{X})$, the Naive Bayes Classifier assumes that the attributes X are conditionally independent from each other, given the entity E:

$$P(E|\mathcal{X}) = P(E)P(\mathcal{X}_1|E)P(\mathcal{X}_2|E), ..., P(\mathcal{X}_n|E) \tag{2}$$

$P(\mathcal{X}_n|E)$ is the likelihood of attribute $\mathcal{X}_n$ given entity E. For example, E could be Michael Jackson, and $\mathcal{X}_n$ could be **is Human**. If the entity has this attribute the value should be 1, and if the entity does not have the attribute this value should be 0. To introduce robustness and to account for some noise in the answers, an error margin of 0.1 has been added. The probability value if $E \in X$, which means entity E has attribute X, is 1 - 0.1 = 0.9 and if $E \in \neg X$, entity E does not have attribute X, the probability is 0+0.1 = 0.1.

Moreover, since the denominator $P(\mathcal{X})$ does not depend on the entity E and is constant for all $\mathcal{E}$, it only serves as a scale for the probabilities. So the marginal probability of the attributes $P(\mathcal{X})$ can be ignored. Hence the probability of an entity E given $\mathcal{X}$ will be calculated as follows:

$$P(E|\mathcal{X}) = P(E)\prod_{i=1}^{|\mathcal{X}|} P(\mathcal{X}_i|E) \tag{3}$$

where $|\mathcal{X}|$ is the number of defined attributes that can have a value from 1 to 20. After each question, the probabilities for all entities are updated, and normalized. After 20 questions, the Naive Bayes Classifier predicts the maximum probability over all entities.

For a better understanding an example will be given where E1 = Michael Jackson *(MJ)*, E2 = Joe Biden *(JB)*, X_1 = is Human *(h)* X_2 = has occupation Singer *(s)*

$$P(MJ|h,\, s) = P(MJ) * P(h|MJ) * P(s|MJ) = \frac{1}{134} * 0.9 * 0.9 \tag{4}$$

$$P(JB|h,\, s) = P(JB) * P(h|JB) * P(s|JB) = \frac{1}{134} * 0.9 * 0.1 \tag{5}$$

The probability for Michael Jackson is significantly higher since he is a singer. Joe Biden is not a singer so the probability of $P(s|JB)$ is 0.1 instead of 0.9 for Michael Jackson.

Question Generation Based on Weighted Entropy To maximize the information gain on each question the entropy is minimized. So for each question, the attribute that gives the largest entropy loss is chosen. The entropy loss H is calculated, for each possible attribute X, using the Shannon Entropy formula [13]:

$$H(X) = -\left[P(X)\log_2(P(X)) + [P(\neg X)\log_2(P(\neg X))\right]. \tag{6}$$

The basic version of this formula works with the probability of an attribute based on its occurrences among the remaining entities. This would be maximal for the question that produces the most balanced split. Since in the probabilistic approach, the search space does not reduce, the formula would produce the same results after each question. So a novel method of taking the probabilities of attributes into account is introduced as 'weighted entropy':

$$H(X) = -\left[W(X)\log_2(W(X)) + [W(\neg X)\log_2(W(\neg X))\right]. \tag{7}$$

$$W(X) = \frac{\sum P(\mathcal{E}_X)}{\sum P(\mathcal{E})} \tag{8}$$

$$W(\neg X) = \frac{\sum P(\mathcal{E}_{\neg X})}{\sum P(\mathcal{E})} \tag{9}$$

The weighted probability for $W(X)$ is calculated by summing the probabilities of all entities E with attribute X ($\mathcal{E}_X$) and dividing by the sum of the probabilities of all entities. Since the probabilities are normalized the sum of the probabilities of all entities is 1 and we can disregard the denominator. Similarly, the weighted probability $W(\neg X)$ is calculated by summing the probabilities of all entities without attribute X and dividing by the sum of the probabilities of all entities $\mathcal{E}$.

The information gain will be maximized by choosing the attribute X which produces the smallest entropy, hence the largest entropy loss, out of the possible attributes. That is the attribute that produces the largest value for $H(X)$. The entropy loss is largest if in Equation 7 the probabilities are equally distributed, hence $W(X)$ and $W(\neg X)$ have a similar value, hence approximating 0.5. By calculating the weight based on the probabilities, it can be ensured that the entities with a higher probability score are significantly more relevant than those with low probabilities, yet, all of them are taken into account. By focusing on the relevant entities, the algorithm can ask specific questions that produce an imbalanced split over the whole dataset, in the case that it gives us information that can be used to distinguish between two entities with a high probability score. The first question of each game is equivalent to working with the Shannon Entropy fomrula since all probabilities are initialized with the same value. The system has a high entropy when the probabilities are distributed equally over all entities. The goal state is a minimized entropy which means that there are only one or few entities with high probabilities and the other entities have relatively small probability values.

3.3 Introducing Noise with Bots

The answering bot retrieves the information about an entity from the same knowledge graph as the engine that is generating the questions. Therefore, it has complete information about the environment and unlike a human player or a bot with different knowledge, it would never produce any wrong answers.

In order to test how the engine can deal with wrong information, noise is introduced with different types of bots. This happens according to specific patterns to investigate different phenomena and aspects of the algorithm. Iterative testing under the following patterns of noise will be performed:

- Random Noise: N wrong answers out of 20
- Chunk Noise: N consecutive wrong answers starting at position X
- Specific + Random Noise: A wrong answer at question X, plus N-1 wrong answers at random positions

Each different pattern of noise will be tested with values for N between 1 and 6 and for positions for X from 1 to 20. So every pattern will be analyzed in every position given 1,2,3,4,5 and 6 wrong answers for each entity. Any value for N larger than 6 will not be tested, because the amount of noise would be disproportionately big.

The results of Random Noise will be used as a baseline of how the algorithm performs under a random pattern of noise. Since the random noise is not deterministic, the experiment will be run 10 times over the whole data set. The average performance will be used, and the standard deviation will be calculated to measure the variance in between runs.

By testing the performance under Chunk Noise, the relevance of the localization of noise can be analyzed. This might give insight into which parts of the algorithm should be more robust than others, or where an effort should be made to verify the results or implement other means of error protection. The exact configuration of Chunk Noise is visualized in Table 1. The averaged performance under Chunk Noise, given N wrong answers, will be compared to the baseline to investigate whether the algorithm performs better or worse under

Chunk Noise instead of randomly distributed noise. This insight can be used to decide whether the performance under Chunk Noise is representative of the overall performance. Under a specific pattern of noise, the algorithm always performs the same. Therefore, the results of Chunk Noise are fully deterministic. The performance over all 134 entities will be measured once for every combination of N wrong questions and X positions. Also, a run without any wrong answers is made to see how the algorithm performs in a noise-free environment.

N Wrong Answers	Wrong Questions	Number of Experiments
N = 1	X = {1}, {2}, {3}...{20}	20
N = 2	X = {1,2}, {2,3}, {3,4}...{19,20}	19
N = 3	X = {1,2,3}, {2,3,4}...{18,19,20}	18
N = 4	X = {1,2,3,4}...{17,18,19,20}	17
N = 5	X = {1,2,3,4,5}...{16,17,18,19,20}	16
N = 6	X = {1,2,3,4,5,6}...{15,16,17,18,19,20}	15

Table 1. Chunk Noise Experimental Setup

The introduction of Specific + Random Noise allows us to analyze the performance when one answer in a specific position is wrong in combination with N-1 other wrong answers in random positions. This is used to evaluate whether any singular questions are more important than others. Under Specific + Random noise there will also be statistical testing, to see whether the performance is comparable to the baseline. In contrast to Chunk Noise where a general trend of importance is generated, here the importance of singular questions is measured to see if they produce similar results.

3.4 Metrics

In the experiments, the most important measurement is how many entities are guessed correctly out of the total of 134 entities. This will be scaled to a percentage of correct guesses and serves as a measure of performance. It indicates how well the algorithm is performing, in the base case, and under specific patterns of randomness. To gain a more in-depth understanding and to help visualize how to algorithm works, the average probability of the goal entity on each step will be measured under Random Noise and Chunk Noise and averaged over all runs under a specified pattern and N.

4 Results & Discussion

The results under the three different types of noise are shown here:

Random Noise (Base Case) The performance of the algorithm under Random Noise was measured for 0-6 wrong answers in random positions and averaged over 1340 games. The algorithm ran over the data set of 134 entities 10 times for each number of wrong answers and the result of each run was measured. The standard deviation represents the variety in performance in between those 10 runs. This scenario is treated as the base case as it demonstrates the algorithm's performance in a completely random environment. In Table 2 the average performance and the standard deviation of the 10 runs are displayed. This is not a deterministic experiment, since the noise is randomly distributed for each entity.

Type of Noise	Wrong Answers	N=0	N=1	N=2	N=3	N=4	N=5	N=6
Random Noise	Performance:	100	97.9	92.5	75.4	46.3	19.2	3.2
	STD:	0	1.6	2.1	4.1	3.7	3.4	1.6
Chunk Noise	Performance:	100	98.5	92.3	80.2	64	45.9	23.8
	STD:	0	1.8	6.5	17.3	24.2	23.9	18.0
Specific + Random	Performance:	100	98.5	92.8	76.9	46.2	18.7	3.6
	STD:	0	1.8	3.5	6.4	7	4.9	1.7

Table 2. Results under different types of Noise

The results indicate that the algorithm performs consistently and has high accuracy with 1-2 wrong answers. As the number of wrong answers increases, the algorithm's performance decreases, while also randomness increases, which is expressed by a larger standard deviation for N=3,4,5 than for any other N. This could mean that with more wrong answers, the importance of their distribution increases. This could have some relation with the number of possible configurations increasing from 20 possible states for N=1 to 20*19*18= 6,840 states for N=3. For N=6 the performance drops to 3.2% and the STD reduces, as the performance is similarly poor in all runs.

Chunk Noise: The performance of the algorithm given noise in the form of chunks is visualized in Fig. 3. The x-axis indicates the starting position of the chunk of wrong answers and on the y-axis, the performance is plotted. The figure depicts the performance given different N's and the average performance in dependence on the position of the wrong answers. In Table. 2 the averaged performance and STD between the different positions of chunks can be seen. In these experiments, it can be seen that similar to Random Noise, the performance drops significantly with an increasing number of wrong answers.

To analyze whether the algorithms performance under Chunk Noise is representative of the performance under Random Noise, the average performance of Random Noise and Chunk Noise is compared and their 95% confidence intervals are calculated. As shown in Fig. 8, the confidence intervals overlap for N=1,2,3. For those values, the statistical difference between the two performances is not significant. For N=4,5,6 the confidence intervals no longer overlap. This indicates that the algorithm performs better under Chunk Noise than under Random Noise, given 4,5, and 6 wrong answers. Those findings were reproduced by a more elaborate Welch's t-test [6] (the exact values can be found in the Appendix: C) and box-plot diagrams for the individual values of N's (Fig. 9). Concluding, the algorithm performs better under Chunk Noise in a specific context; for values N=1-3, we can assume that the performance under Chunk Noise is representa-

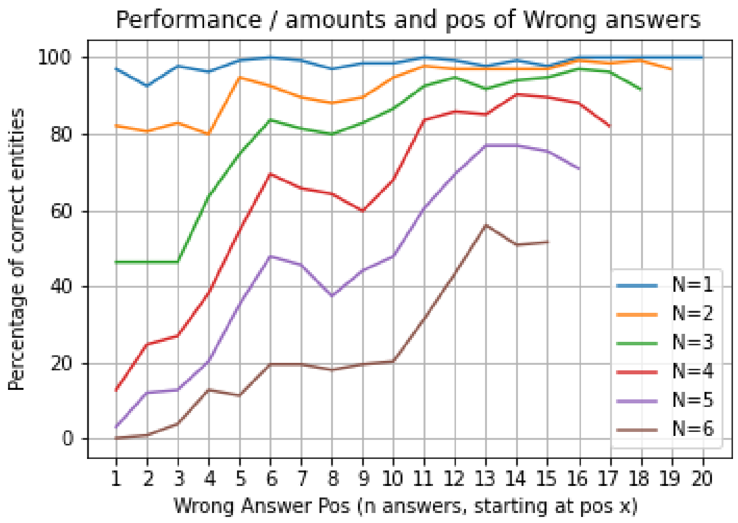


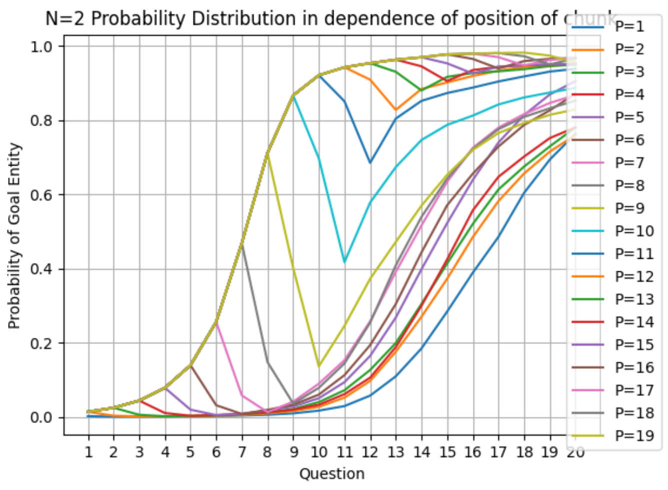
Fig. 3. Performance dependent on position and amount of wrong answers

tive of the algorithm's overall performance, and, for values larger than 3, this does not hold.

The results of Chunk Noise indicate a second trend: The later the wrong answers are given, the better the algorithm performs (Fig. 3). There is an intuitive explanation for why the algorithm exhibits the aforementioned trend. When wrong answers are given at the start, it weighs different entities stronger and focuses its search on a different space. The high STD under chunk noise indicates strongly varying performance, for the same number of errors, given different positions of chunks.

Another interesting result of the Chunk Noise experiment is that there seems to be a local maximum in performance between positions 5 and 6 and a local minimum between N=8,9,10. This spike is visible for values of N between 2 and 5. This finding is contrary to the general trend of linear growth, the later the chunk occurs. The reason for this behavior could lie in the specific data set that is used or could show some limitations of the algorithm. It could indicate that there is some irregularity in the importance of questions and that it does not follow a linear trend, with this specific data set. By design, the algorithm can not ask the same question twice for the same entity. Therefore, one reason for the local drop in performance could be that a question in that specific position is the only way to distinguish one entity from another one if they share all other attributes.

To further investigate the phenomenons of the local maximum and the increasing performance for wrong answers at later positions, the average probability of the goal entity over the course of the game was measured and is depicted



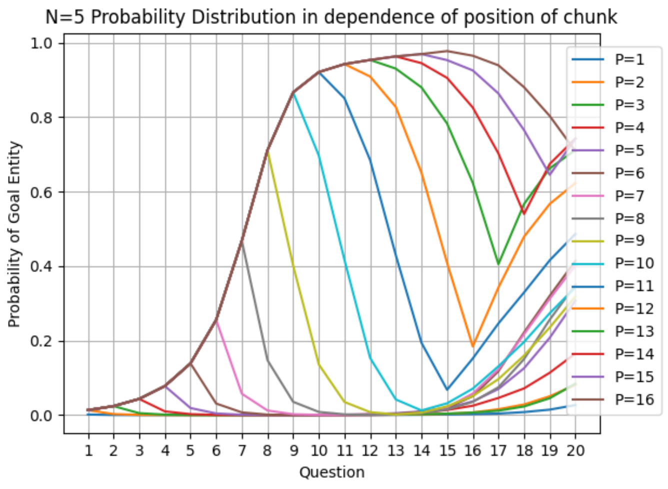


Fig. 4. Average Goal Entity Probability at position X, with N wrong answers starting at position P

in Fig. 4 for N=2 and N=5 and for all other values in the Appendix B. This gives an indication of how the probability of an entity increases with correct answers, and how the algorithm can recover from wrong answers. If the wrong answers occur at a later position, the goal entity has a considerably higher probability score than any other entity. As can be seen in Fig. 4 after 7 correct answers the relative probability is 0.5 and after 10 questions it is above 0.9. It shows that for later wrong answers, the relative probability only drops slightly, and the entity has a higher chance of remaining the most likely one. Also, it can be seen that if

the wrong answers occur earlier, the algorithm takes longer to recover and reach the same probability as before the wrong answers.
In Fig. 5 the average probabilities at position X for different values of N can be seen. This shows an interesting phenomenon, namely a local maxima at question 9. Under Random Noise no such pattern exists, as can be seen in Fig. 6.

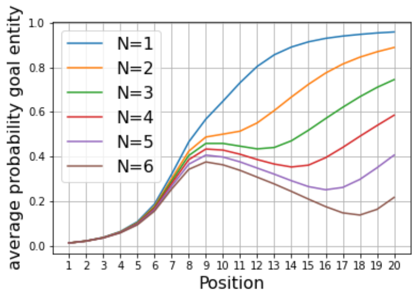


Fig. 5. Average Goal Entity Probability at position X under Chunk Noise

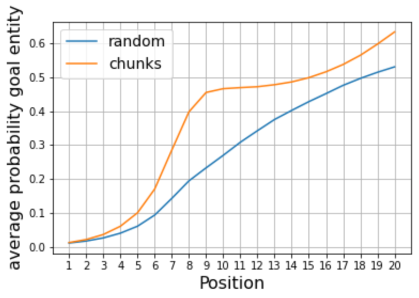


Fig. 6. Average Goal Entity Probability at position X under Chunk and Random Noise

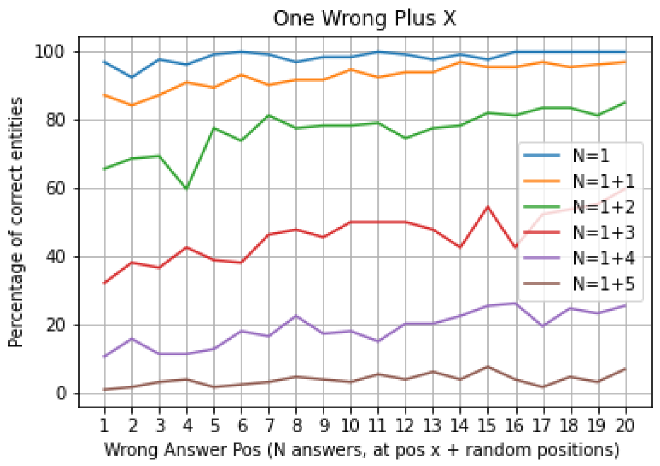


Fig. 7. Performance with a wrong answer at question X plus N random wrong answers

Specific + Random Noise: The performance of answering a question at position X wrong, plus N-1 other wrong answers in random positions is visualized in Fig. 7. Importantly, this experiment does not replicate the irregularity of a local maximum, followed by a local minimum observed in positions 5/6 and 8/9/10 under Chunk Noise. The performance under Specific+Random Noise is a close match to Random Noise (see Table 2 and Appendix C). The standard deviation is similar to the standard deviation under Random Noise. According to Fig. 6 there is no statistically relevant difference for any N. One reason for this behavior could be that for larger values of N larger parts of the noise are generated in random positions rather than in the specific position.

Unlike in the Chunk Noise experiment, the linear growth trend isn't as strongly visible here, and the performance appears to be more consistent. However, under close investigation, a general trend of linear growth is visible. Comparing this to the previous experiments suggests that the spike observed in the Chunk Noise experiment is rather related to a pattern of wrong answers rather than being attributed to one specific question. The STD between runs is higher than under Random Noise, but much lower than under Chunk Noise. It does not give evidence for the theory that one question in a specific position is the reason the algorithm cannot distinguish two entities from one another.

Concluding, combining Random Noise and Chunk Noise to the Specific+Random Noise pattern produces results that replicate the general findings under Chunk Noise of improved performance with later wrong answers, whilst at the same time not being statistically different from Random Noise. It also did not reproduce the phenomena of local maxima in performance, so the reason could not be, as suspected, a single answer, but rather an interaction of different factors, given the data set and the algorithm.
The exact performance values for all experiments can be found in the Appendix 4, 5.

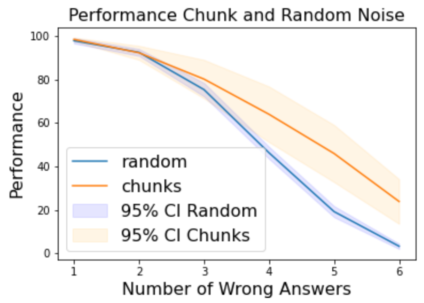


Fig. 8. Random Noise and Chunk Noise average Performance and 95% CI

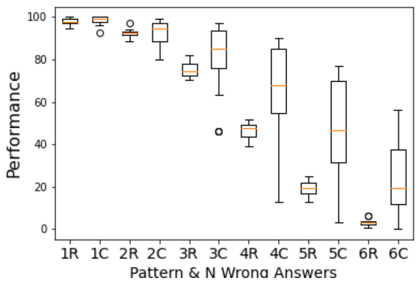


Fig. 9. Statistical Analysis results Random Noise (R) & and Chunk Noise (C), N wrong answers

5 Limitations and Future Work

While this research gives insight into the general performance of the proposed algorithm and analyzes it in depth given certain types of noise, it also reveals several areas that should be further explored. The findings under Chunk Noise show the trend of worse performance for early wrong answers. A way to deal with this would be to use Unequal Error Protection (UEP). It could be implemented by giving less relevance to the first answers and gradually increasing the weight of the answers. Other more complex methods might be to use mathematical distributions like a logarithmic scale. The suitability of specific UEP methods introduced by Chung et. al [4] could be analyzed in the context of this algorithm.

One of the key limitations of this work is that we only sampled noise from a small subset of all possibilities. In the future, this algorithm could be analyzed more in-depth, by sampling from different distributions or by trying to identify patterns where the algorithm performs best/worst given a specific N. It could not be analyzed whether there are other patterns that produce interesting results. Such patterns could be for example every second/third question or two questions at the start and two at the end, etc. The phenomena of local maxima in both relative probability and performance under Chunk Noise require further investigation to fully understand their causes and effects on the performance.

Running the experiments with the same pattern of noise on a different dataset is crucial to gain a deeper understanding. It could give insight into which phenomena were caused by the data set and which aspects are inherent to the algorithm itself.

Running the algorithm in another, denser setting that provides more possibilities for better splits is a good way to gain insight into its mathematical limitations. An issue that could arise when scaling this method to larger datasets, with more possible splits, is the computational complexity of the algorithm. This approach computes a score for every possible split, which could become too computationally expensive. Future work could explore the potential of random branching instead of exhaustively scoring all possibilities.

Another variable that could be further explored is the human error rate which was defined to be 10% in this paper. It could wield interesting insights to match the error rate to the actual amount of wrong answers N.

In this work, no human input was used. However, this approach could be tested by a human to see whether the improved robustness makes the game easier or more engaging to play. Moreover, it could be tested to what extent real-world applications can make use of this algorithm. Potential areas of application are querying (medical) databases, or product recommendations. For example, if the user specifies too many filters and rules out all possible entities, the robustness is especially useful to predict entities that are a close match to what the user is looking for.

Lastly, efficiency was not considered in this work. It could be analyzed how terminating early once a certain probability threshold has been reached, affects the performance or playability for a human player.

6 Conclusion

In conclusion, the efficacy of the Bayesian algorithm for introducing robustness to a 20 questions game algorithm has been analyzed. The algorithm works effectively for the given problem, and because of its simple, mathematical nature can be generalized easily to different datasets or other similar problems. The experimental findings provided both expected and unexpected insights. The intuitive findings were that the later a wrong answer occurred, the better the algorithm could deal with it, and the more wrong answers, the worse the overall performance. Surprisingly, the algorithm showcased a significantly better performance under Chunk Noise than under Random Noise, for values N=4,5,6. Moreover, analyzing the probabilities of the goal entity under Chunk Noise showed a local maximum in probability around question 9, while the performance showed a local maximum if wrong answers were given in a chunk, starting at position 5 or 6. These unexpected phenomena raise the question to which extent this is a result of the dataset or the algorithm itself. While trying to explore this further using Random+Specific Noise it was found that the underlying reason behind those phenomena is not a single wrong answer. The reason might lie in the interplay of the aforementioned factors. However, this work was not able to understand which aspects play which parts in causing this. Experimenting with other datasets would be crucial to answer this question. The results give rise to areas of potential performance improvement, such as the application of Unequal Error Protection (UEP) [4], different ways of computing updated probabilities based on the game's position, and the introduction of random branching as an alternative to assessing all possible splits with each question. Moreover, the variable for Error Rate was kept constant throughout experiments and the algorithm could benefit from an optimization of this variable.

7 Acknowledgement

Thanks to B. B. Kruit for the guidance and supervision of my work. Both the conventional algorithm and the engine have been created in collaboration with Daphne Post and Luca Rosic. The code for these experiments can be found on GitHub [7].

References

1. Alkaed, H.: Entity prediction task in noisy environments (2022)
2. Ben-Or, M., Hassidim, A.: The Bayesian Learner is Optimal for Noisy Binary Search (and Pretty Good for Quantum as Well). In: 2008 49th Annual IEEE Symposium on Foundations of Computer Science. pp. 221–230 (Oct 2008). https://doi.org/10.1109/FOCS.2008.58, iSSN: 0272-5428
3. Berrar, D.: Bayes' theorem and naive bayes classifier. Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics **403**, 412 (2018)
4. Chung, H.W., Sadler, B.M., Zheng, L., Hero, A.O.: Unequal Error Protection Querying Policies for the Noisy 20 Questions Problem. IEEE Transactions on Information Theory **64**(2), 1105–1131 (Feb 2018). https://doi.org/10.1109/TIT.2017.2760634, http://arxiv.org/abs/1606.09233, arXiv:1606.09233 [cs, math]

5. Cortés-Calabuig, A., Denecker, M., Arieli, O., Van Nuffelen, B., Bruynooghe, M.: On the Local Closed-World Assumption of Data-Sources. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) Logic Programming and Nonmonotonic Reasoning. pp. 145–157. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2005). https://doi.org/10.1007/11546207_12
6. Derrick, B., Toher, D., White, P.: Why welch's test is type i error robust (2016)
7. Donle, A.: 20qprob (2023), https://github.com/toschka123/20Qprob
8. Dou, J., Tian, B., Zhang, Y., Xing, C.: A Novel Embedding Model for Knowledge Graph Completion Based on Multi-Task Learning. In: Jensen, C.S., Lim, E.P., Yang, D.N., Lee, W.C., Tseng, V.S., Kalogeraki, V., Huang, J.W., Shen, C.Y. (eds.) Database Systems for Advanced Applications. pp. 240–255. Lecture Notes in Computer Science, Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-73194-6_17
9. Jedynak, B., Frazier, P.I., Sznitman, R.: Twenty Questions with Noise: Bayes Optimal Policies for Entropy Loss. Journal of Applied Probability **49**(1), 114–136 (Mar 2012). https://doi.org/10.1239/jap/1331216837, https://www.cambridge.org/core/journals/journal-of-applied-probability/article/twenty-questions-with-noise-bayes-optimal-policies-for-entropy-loss/6F51F7E5CE7D0B1ED4CEF0221D0B6E08, publisher: Cambridge University Press
10. Jedynak, B., Frazier, P.I., Sznitman, R.: Twenty questions with noise: Bayes optimal policies for entropy loss. Journal of Applied Probability **49**(1), 114–136 (2012)
11. Leung, K.M., et al.: Naive bayesian classifier. Polytechnic University Department of Computer Science/Finance and Risk Engineering **2007**, 123–156 (2007)
12. Pellissier Tanon, T., Weikum, G., Suchanek, F.: YAGO 4: A Reason-able Knowledge Base. In: Harth, A., Kirrane, S., Ngonga Ngomo, A.C., Paulheim, H., Rula, A., Gentile, A.L., Haase, P., Cochez, M. (eds.) The Semantic Web. pp. 583–596. Lecture Notes in Computer Science, Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-49461-2_34
13. Shannon, C.E.: A mathematical theory of communication. The Bell System Technical Journal **27**(3), 379–423 (1948). https://doi.org/10.1002/j.1538-7305.1948.tb01338.x
14. Suresh, S.R.: A bayesian strategy to the 20 question game with applications to recommender systems. Ph.D. thesis, Duke University (2017)
15. Ulam, S.M.: Adventures of a Mathematician. University of California Press (1991), google-Books-ID: U2_zEZOHdU4C
16. Variani, E., Lahouel, K., Bar-Hen, A., Jedynak, B.M.: Non-adaptive policies for 20 questions target localization. 2015 IEEE International Symposium on Information Theory (ISIT) pp. 775–778 (2015)
17. Walsorth, M.T.: Twenty Questions: A Short Treatise on the Game to which are Added a Code of Rules and Specimen Games for the Use of Beginners. Holt (1882), google-Books-ID: VzoVAAAAYAAJ
18. Wu, X., Hu, H., Klyen, M., Tomita, K., Chen, Z.: Q20: Rinna Riddles Your Mind by Asking 20 Questions (2018)

A Extensive Results

	Random Noise Performance					
	N=1	**N=2**	**N=3**	**N=4**	**N=5**	**N=6**
Run 1	97.76	94.03	75.37	51.49	22.39	2.24
Run 2	94.78	92.54	73.13	47.01	21.64	2.99
Run 3	97.01	93.28	72.39	43.28	17.91	3.73
Run 4	99.25	90.30	75.37	49.25	21.64	5.97
Run 5	97.01	93.28	82.09	44.78	24.63	3.73
Run 6	100.00	97.01	82.09	47.76	20.15	2.24
Run 7	99.25	92.54	70.15	49.25	16.42	2.99
Run 8	100.00	92.54	71.64	48.51	12.69	5.97
Run 9	97.76	91.04	72.39	42.54	17.91	0.75
Run 10	96.27	88.81	79.10	38.81	16.42	1.49

Table 3. All Results Under Random Noise

	Chunk Noise Performance					
	N=1	**N=2**	**N=3**	**N=4**	**N=5**	**N=6**
X=1	97.01	82.09	46.27	12.69	2.99	0.00
X=2	92.54	80.60	46.27	24.63	11.94	0.75
X=3	97.76	82.84	46.27	26.87	12.69	3.73
X=4	96.27	79.85	63.43	38.06	20.15	12.69
X=5	99.25	94.78	74.63	54.48	35.07	11.19
X=6	100.00	92.54	83.58	69.40	47.76	19.40
X=7	99.25	89.55	81.34	65.67	45.52	19.40
X=8	97.01	88.06	79.85	64.18	37.31	17.91
X=9	98.51	89.55	82.84	59.70	44.03	19.40
X=10	98.51	94.78	86.57	67.91	47.76	20.15
X=11	100.00	97.76	92.54	83.58	60.45	31.34
X=12	99.25	97.01	94.78	85.82	69.40	43.28
X=13	97.76	97.01	91.79	85.07	76.87	55.97
X=14	99.25	97.01	94.03	90.30	76.87	50.75
X=15	97.76	97.01	94.78	89.55	75.37	51.49
X=16	100.00	99.25	97.01	88.06	70.90	70.90
X=17	100.00	98.51	96.27	82.09		
X=18	100.00	99.25	91.79			
X=19	100.00	97.01				
X=20	100.00					

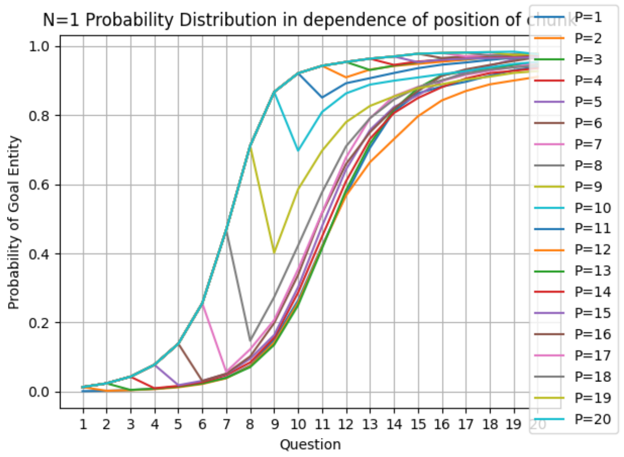
Table 4. All Results Under Chunk Noise

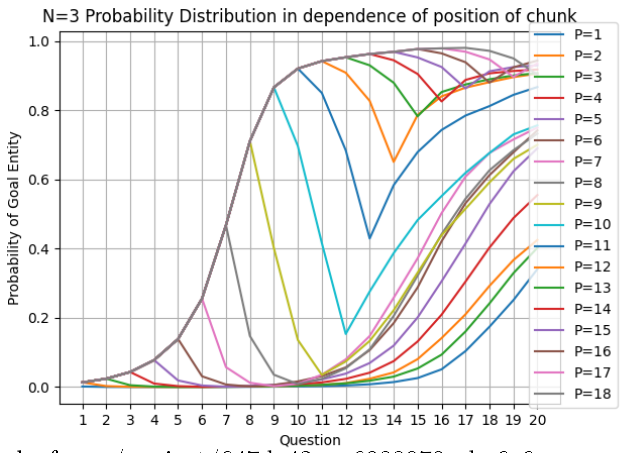
	X Plus N Performance					
	X+N=1	**X+N=2**	**X+N=3**	**X+N=4**	**X+N=5**	**X+N=6**
X=1	97.01	87.31	65.67	32.09	10.45	0.75
X=2	92.54	84.33	68.66	38.06	15.67	1.49
X=3	97.76	87.31	69.40	36.57	11.19	2.99
X=4	96.27	91.04	59.70	42.54	11.19	3.73
X=5	99.25	89.55	77.61	38.81	12.69	1.49
X=6	100.00	93.28	73.88	38.06	17.91	2.24
X=7	99.25	90.30	81.34	46.27	16.42	2.99
X=8	97.01	91.79	77.61	47.76	22.39	4.48
X=9	98.51	91.79	78.36	45.52	17.16	3.73
X=10	98.51	94.78	78.36	50.00	17.91	2.99
X=11	100.00	92.54	79.10	50.00	14.93	5.22
X=12	99.25	94.03	74.63	50.00	20.15	3.73
X=13	97.76	94.03	77.61	47.76	20.15	5.97
X=14	99.25	97.01	78.36	42.54	22.39	3.73
X=15	97.76	95.52	82.09	54.48	25.37	7.46
X=16	100.00	95.52	81.34	42.54	26.12	3.73
X=17	100.00	97.01	83.58	52.24	19.40	1.49
X=18	100.00	95.52	83.58	53.73	24.63	4.48
X=19	100.00	96.27	81.34	55.22	23.13	2.98
X=20	100.00	97.01	85.07	59.70	25.37	6.72

Table 5. All Results Under Random+Specific Noise

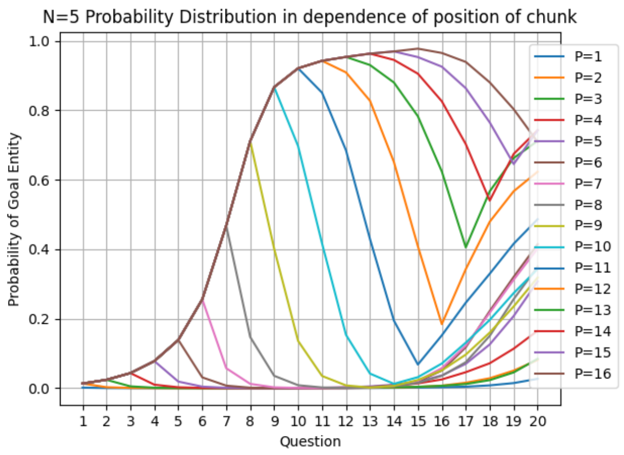
B Probability Entity Chunks

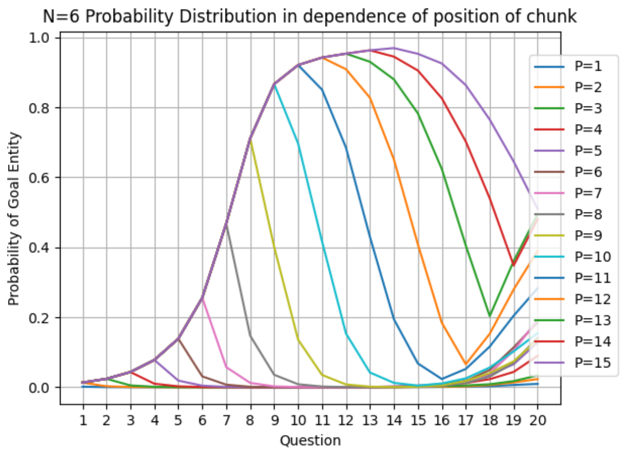
The following figures show the average probability of the goal entity under Chunk Noise. Each figure has a different value for N. P indicates the starting position of the chunk of wrong answers. The y-axis indicates the average probability of the goal entity and the x-axis represents the position in the game.





https://de.overleaf.com/project/647de43aaa6988070cebc6e6


N=5 Probability Distribution in dependence of position of chunk
Probability of Goal Entity
Question
0.0
0.2
0.4
0.6
0.8
1.0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
P=1
P=2
P=3
P=4
P=5
P=6
P=7
P=8
P=9
P=10
P=11
P=12
P=13
P=14
P=15
P=16


N=6 Probability Distribution in dependence of position of chunk
Probability of Goal Entity
Question
0.0
0.2
0.4
0.6
0.8
1.0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
P=1
P=2
P=3
P=4
P=5
P=6
P=7
P=8
P=9
P=10
P=11
P=12
P=13
P=14
P=15

C Statistical Analysis & P values

The following table depicts the p values between the performances of the algorithm under different patterns of noise. The difference is statistically significant if the p value is smaller than 0.05. The following boxplot diagram

P-Values (Random vs Chunk)						
N	1	2	3	4	5	6
P-Value	0.4	0.9294	0.4085	0.0360	0.0025	0.0021

P-Values (Random vs Specific)						
N	1	2	3	4	5	6
P-Value	0.4	0.836	0.52	0.976	0.804	0.551

P-Values (Chunk vs Specific)						
N	1	2	3	4	5	6
P-Value	1	0.791	0.437	0.004	2.9e-05	2.87e-05

Table 6. Statistical Analysis of the Experiments

shows the differences between the performance under Chunk Noise and Specific + Random Noise.

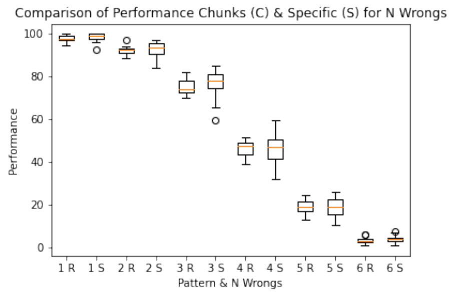


Fig. 10. Statistical Analysis results Chunks (C) & Specific (S)

D Average Probability Goal Entity under Random noise

The following figure shows the average probability of the goal entity at position X, given different values for N.

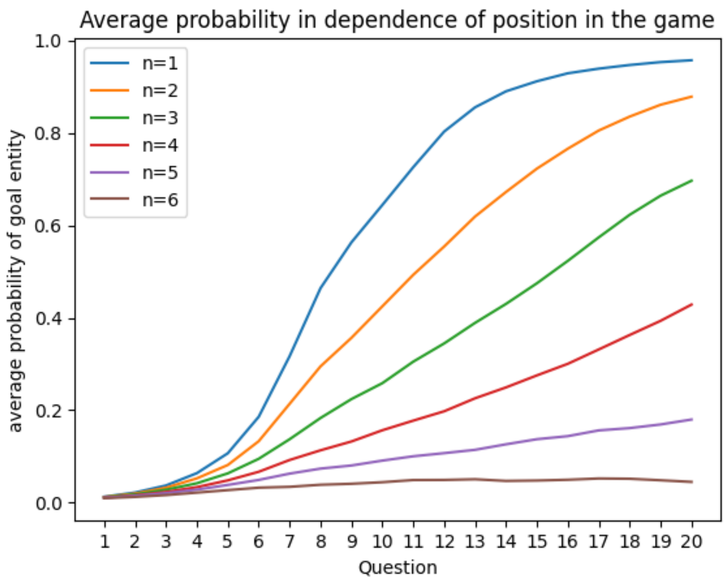


Fig. 11. probability under random noise