Constrained Optimal Querying: Huffman Coding and Beyond

Shuyuan Zhang Jichen Sun Shengkang Chen

May 2022

Abstract

Huffman coding is well known to be useful in certain decision problems involving minimizing the average number of (freely chosen) queries to determine an unknown random variable. However, in problems where the queries are more constrained, the original Huffman coding no longer works. In this paper, we proposed a general model to describe such problems and two code schemes: one is Huffman-based, and the other called GBSC (Greedy Binary Separation Coding). We proved the optimality of GBSC by induction on a binary decision tree, telling us that GBSC is at least as good as Shannon coding. We then compared the two algorithms based on these two codes, by testing them with two problems: DNA detection and 1-player Battleship, and found both to be decent approximating algorithms, with Huffman-based algorithm giving an expected length 1.1 times the true optimal in DNA detection problem, and GBSC yielding an average number of queries 1.4 times the theoretical optimal in 1-player Battleship.

Keywords: Information theory, Huffman coding, greedy algorithms, decision trees, Battleship game, DNA exon detection

1 Introduction

Our research is inspired by a problem in Cover's textbook[1] (see page 153).

Example 1 (Bad wine). *One is given six bottles of wine. It is known that precisely one bottle has gone bad (tastes terrible). From inspection of the bottles it is determined that the probability p_i that the ith bottle is bad is given by $(p_1, p_2, \ldots, p_6) = (\frac{8}{23}, \frac{6}{23}, \frac{4}{23}, \frac{2}{23}, \frac{2}{23}, \frac{1}{23})$. Tasting will determine the bad wine. You can mix some of the wines in a fresh glass and sample the mixture. You proceed,* *mixing and tasting, stopping when the bad bottle has been determined. What is the minimum expected number of tastings required to determine the bad wine?*

As an exercise in the textbook, this problem is well solved. In fact, it is equivalent to Huffman coding. The process that we determine which bottle of wine is bad can be regarded as a decision tree. At each node, we make an observation that has two possible outcomes (in this example, good or bad) and move to the left or right child node according to the outcome. This process continues until we reach a leaf node. If we represent each move to left with 0 and each move to right with 1, then a move sequence can be represented by a binary code. Minimizing the expected number of moves is equivalent to minimizing the expected code length, so we only need to use Huffman coding.

We want to generalize this problem. A natural generalization is changing the number of bad wines. It seems that the original Huffman coding strategy will still work, but unfortunately, it is wrong. This is shown by the following example.

Example 2 (More bad wine)**.** *The background is similar to Example 1, but this time there are four bottles of wine to be examined and two of them are bad. Suppose the probability p_{ij} that the ith and jth bottles are bad is given by* $(p_{12}, p_{13}, p_{14}, p_{23}, p_{24}, p_{34}) = (0.1, 0.1, 0.15, 0.15, 0.3, 0.2)$. *What is the minimum expected number of tastings required to determine the two bad wines?*

If we repeat the Huffman coding strategy, we will obtain the unique Huffman tree below.

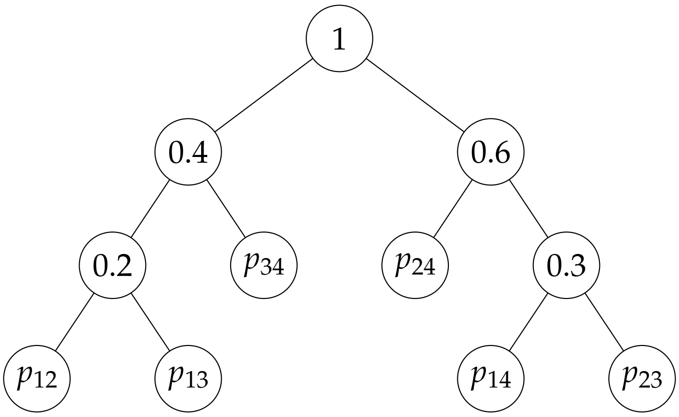


Figure 1: Huffman Tree of Example 2.

The expected Huffman code length is 2.5, but it is not the answer. This is because the decision tree we have constructed is infeasible. To see this, just look at the root of the tree. Note that in both the left and right subtrees, every bottle of wine may be bad, so there is no way that we can decide in which direction we should move by just one observation.

Example 2 shows that changing Example 1 a little bit will produce a much more difficult problem. In fact, Example 2 is just an instance of a huge family of similar problems, and finding the precise solution of these problems is usually extremely difficult.

In this report, we proposed a generalized model that formulates such type of problems, and analyzed three problems related to this model. We designed two generalized approximate algorithms for these problems. One algorithm is based on Huffman coding, while the other is based on a new code scheme (at least not being researched much). We call the new code GBSC (Greedy Binary Separation Code), and we proved an interesting property of GBSC.

2 Models

2.1 Generalized Model

We consider this generalized problem. Given a random variable X with alphabet $\mathcal{X}$ and its distribution $p(x)$. We determine the value of X by asking questions. The i-th question we ask is that "is X in set S_i?", where $S_i \in \mathscr{A}, \forall i$. Suppose we need N questions to determine the value of X. Our goal is to find out the minimum of EN (the expectation of N) and if possible, its corresponding strategy.

Definition 1. *The family of set $\mathscr{A}$ is called a decision set on $\mathcal{X}$.*

Obviously, different structures of the decision set will produce a bunch of completely different problems. In the rest of this chapter, we will introduce three specializations of this model, which are Huffman coding, the DNA detection problem and the Battleship problem.

2.2 Huffman Coding: the Easy Case

In Example 1, the decision set is $2^{\mathcal{X}}$. We have seen that in this case, the problem is equivalent to Huffman coding. However, the condition can be loosened slightly.

Definition 2. *A decision set $\mathscr{A}$ on $\mathcal{X}$ is decision-complete, if $\forall S \in 2^{\mathcal{X}} \setminus \{\emptyset, \mathcal{X}\}$, $S \in \mathscr{A}$ or $\mathcal{X} \setminus S \in \mathscr{A}$.*

Proposition 1. *If $\mathscr{A}$ is decision-complete, then any decision tree of X is feasible.*

Proof. This is easy to demonstrate. Asking "is X in set S_i?" is equivalent to asking "is X in set $\mathcal{X} \setminus S_i$?", and there is no point to ask whether X is in $\emptyset$ or $\mathcal{X}$. □

Corollary 1. *If $\mathscr{A}$ is decision-complete, then Huffman coding will produce the optimal solution.*

Proposition 1 also provides a trivial but practical necessary condition for a set to be decision-complete.

Corollary 2. *If $\mathscr{A}$ is decision-complete, then $|\mathscr{A}| \geq 2^{|\mathcal{X}|-1} - 1$.*

Using Corollary 2, it is easy to prove that the decision set of Example 2 is not decision-complete, so the Huffman coding strategy might fail.

Proposition 2. *The decision set $\mathscr{A}$ of Example 2 is not decision-complete.*

Proof. $|\mathcal{X}| = 6$, so $|\mathscr{A}| = 2^4 - 1 < 2^{|\mathcal{X}|-1} - 1$. □

2.3 DNA Detection Problem

To determine genomic sequences of several organisms, we need to assign biological meaning to particular regions of the sequence. One of important steps in this process is the identification of genes. Exon is an interval of the DNA sequence and it does not overlap with other exons and gene is a sequence of exons. In this paper, we simplified the question by assuming that the target gene only contains one exon. We can detect whether the target exon is in an interval in the DNA sequence or not at each detection. The position of the exon on DNA is fixed, so we want to minimize the expected number of detection to determine the position of the exon by choosing the intervals wisely, thus reducing the cost of DNA detection[2][3].

Definition 3. *A set S of integers is continuous, if $S = [\min S, \max S] \cap \mathbb{Z}$.*

Example 3. $\{1, 2, 3\}$ *and* $\{4\}$ *are continuous, while* $\{1, 3\}$ *is not.*

If we assume that the exon's location we want to find on the DNA is discrete and unique, we can use a random variable $X \in \{1, 2, \ldots, n\}$ to it, and in this case, the decision set is any continuous set in space $\mathcal{X}$.

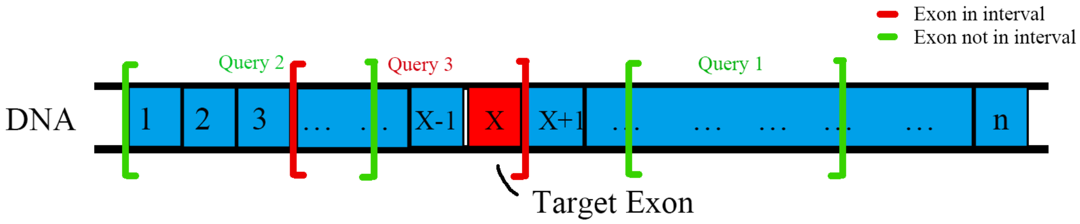


Figure 2: DNA exon detection.

2.4 Battleship Problem

"Battleship" is a popular 2-player strategy and guessing game. In the typical setting, each player places "boats" with different lengths on his 10 × 10 board, which is hidden to the opponent. Each player take turns to "bomb" a grid (i, j) on opponent's board, and the opponent must honestly report if it "hits" or "misses". The goal of the game is sink all of the opponent's ships, i.e. "hit" all the grids on the opponent's board that represents a ship, before the opponent sinks all the player's ships. To simplify analysis and computation, we instead study the 1-player Battleship. In this setting, the game randomly generates a possible layout of ships unknown to the player. The goal of the player is to sink all the ships in the fewest tries (bombs). For example, in this particular board placed by the opponent (the game), the player needs to "hit" all grids marked in gray.

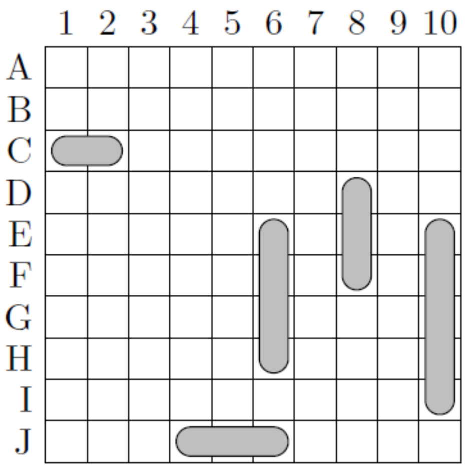


Figure 3: One possible board of one player in Battleship [4].

The 1-player Battleship problem can be formulated into the following. Suppose there are n different possible ship layouts. Let $\mathcal{X}_0 = \{X_1, X_2, X_3, \ldots, X_n\}$ be the set of all possible layouts, where $X_k, 1 \leq k \leq n$ is a 10 × 10, $0 - 1$ matrix. The game randomly chooses the target board $X^* \in \mathcal{X}_0$ (every legal board is equally possible), and the player tries to minimize the number of tries T to guess T^*.

At t-th step, the player gives a query on i, j: $Q_t = (i_t, j_t)$, which means: "Is bombing of grid (i, j) a hit?" the game answers honestly, giving the player some information to eliminate some layouts in $\mathcal{X}_{t-1}$ and get the new $\mathcal{X}_t$.

$$\mathcal{X}_t = \{X | (X \in \mathcal{X}_{t-1}) \wedge (X_{ij} = X^*_{ij})\}$$

The goal is to minimize the number of tries T to determine what the target layout is, by choosing queries wisely. That is,

$$\min_{Q_1, Q_2, \ldots, Q_T} T, Q_t = (i_t, j_t)$$

$$\mathcal{X}_t = \{X | (X \in \mathcal{X}_{t-1}) \wedge (X_{i_t j_t} = X^*_{i_t j_t})\}, 1 \leq t \leq T$$

$$\mathcal{X}_0 = \{X_1, X_2, X_3, \ldots, X_n\}, \mathcal{X}_T = \{X^*\}, \text{ random variable } X^* \in \mathcal{X}_0$$

More precisely, we would like to minimize the average number of tries $\bar{T}$ over all possible targets X^*.

Previous studies use diagonal searching[5], tree search[4], or RL-based[6] approaches. In our study, we adopt the idea of greedy information gain per step, which will be demonstrated in Chapter 4.

3 Greedy Algorithm Based on Huffman coding

3.1 Description

The most straightforward algorithm is search by brute force. This algorithm is promised to find the precise result but is too slow for large-scale problems. Therefore, we try to use the same method of Huffman coding. In every iteration we still try to merge the two nodes with minimum sum of probabilities, if possible. This gives us the prototype of a greedy algorithm based on Huffman coding.

The critical part of the algorithm is how we judge whether two nodes can be merged (line 8). The following proposition describes this process mathematically, but it is not practical in practice. The algorithm of the process needs to be designed specifically to get a better performance.

Definition 4. *The set of possible values of X at a certain node of a decision tree is called the candidates of the node.*

Algorithm 1 Greedy algorithm based on Huffman coding

```
procedure GreedyHuffman(nodes)
    if $nodes$ contains only one node then
        return $nodes$
    end if
    $pairs \leftarrow \{(nodes[i], nodes[j]) : i < j\}$
    sort $pairs$ by the sum of probabilities in ascending order
    for every $(i, j) \in pairs$ do
        if $i$ and $j$ can merge then
            $newNodes \leftarrow$ merge $i$ and $j$
            $tree \leftarrow$ GreedyHuffman($newNodes$)
            if $tree \neq \emptyset$ then
                return $tree$
            end if
        end if
    end for
    return $\emptyset$
end procedure
read $p(x)$ and initialize corresponding $nodes$
$tree \leftarrow$ GreedyHuffman($nodes$)
```

Proposition 3. *Assume that A and B are the candidates of two nodes. Then the two nodes can be merged if and only if $\exists C \in \mathscr{A}$, $(A \cup B) \setminus C \in \{A, B\}$.*

3.2 First Try to Solving the DNA Detection Problem

The following proposition instruct us how to judge whether two nodes can be merged for this problem.

Proposition 4. *Assume that A and B are the candidates of two nodes. Then the two nodes can be merged if and only if any of the two conditions holds:*

1. *A or B is continuous.*

2. $\min A > \max B$ *or* $\min B > \max A$.

Thus, we implemented our first algorithm that deals with the DNA detection problem. We carried out a experiment by generating 10,000 random $p(x)$ with $n = 6$ and compare the result of the brute force algorithm and the Huffman-based algorithm. The results of the experiment indicate that the result of the greedy algorithm is rather close to the optimal value. The result of each data is represented by a point in Figure 4. The red line in Figure 4 is the optimal bound, because the brute force algorithm is promised to find the precise optimal value. We can see that most of the points are close to the red line, and many of them exactly lie on the red line, which means they reach the optimal bound. In fact, around 60% of the data reach the optimal bound.

This intuition is further confirmed by Figure 5. Let L_b and L_g be the expected length of the brute force algorithm and the Huffman-based algorithm, respectively. We define that gap $= \frac{L_g - L_b}{L_b}$. Although the maximum of gap is around 35%, in most cases, gap is less than 10%. Therefore, the result of the Huffman-based algorithm is very satisfactory.

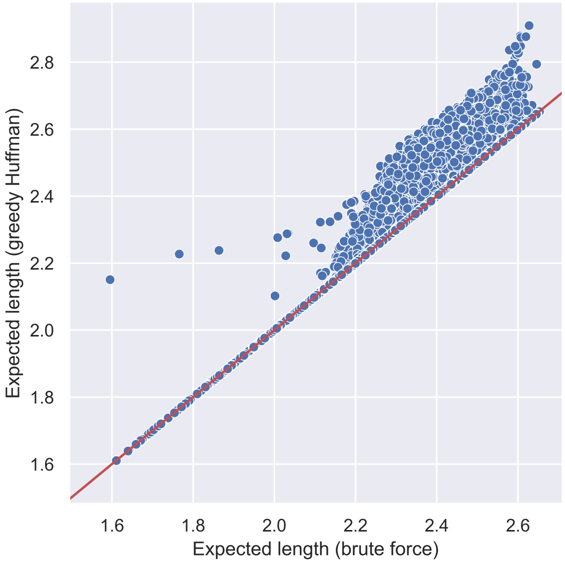


Figure 4: Expected lengths of the brute force algorithm and the Huffman-based algorithm.

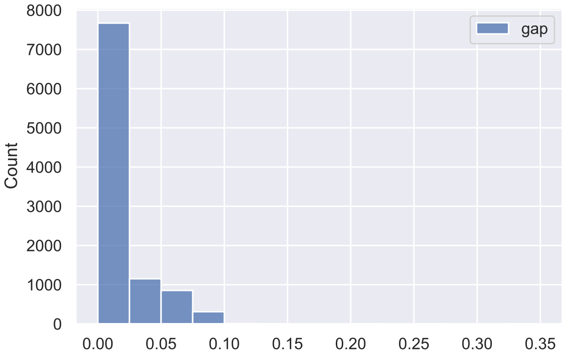


Figure 5: Distribution of the gap between the optimal value and the output of the Huffman-based algorithm.

4 Greedy Binary Separation Coding

4.1 Motivation

The greedy algorithm based on Huffman coding has a vital drawback: when $|\mathcal{X}|$ is large and $|\mathscr{A}|$ is relatively small, the algorithm becomes very slow. In most cases, $|\mathscr{A}|$ is small compared to $|\mathcal{X}|$. Therefore, instead of building the decision tree from bottom to top, we consider another way of building the tree from top to bottom, by choosing the best question. We call this coding Greedy Binary Separation Coding (GBSC).

4.2 Definition

We now formally describe GBSC.

Definition 5. *$\{A, B\}$ is a (binary) partition of S, if $A \cup B = S$ and $A \cap B = \emptyset$.*

Definition 6. *A partition $\{A, B\}$ of S is optimal, if for any partition $\{C, D\}$ of S, $|p(A) - p(B)| \leq |p(C) - p(D)|$, where $p(X)$ is the sum of all the numbers in X.*

The idea of GBSC is simple. We construct the decision tree from top to bottom. Let $S = \{p(x) : x \in \mathcal{X}\}$. At root, we choose any optimal partition of S, and split the tree according to the partition. The process is repeated recursively at each node, and each time we try to find the best partition of the candidates of the node.

Example 4. *If $X \in \{1, 2, 3, 4\}$ and $(p_1, p_2, p_3, p_4) = (0.1, 0.2, 0.3, 0.4)$, then the decision tree of X using GBSC is as follows.*

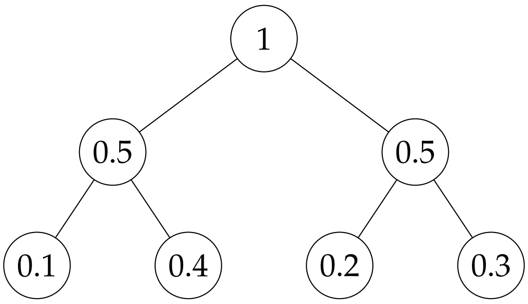


Figure 6: Decision tree of Example 4.

4.3 Intuition

The intuition of GBSC is simple: greedily maximize the information gain (mutual information) at each query. In binary decision trees, each node corresponds to a possible random variable sampled from some distribution. For a query Q on node $X \in \mathcal{X} = \{X_1, X_2, \ldots, X_n\}$, each X_i gives a binary answer. The query is designed such that

$$p = \frac{|\{X_i | X_i \in \mathcal{X} \land \text{ query on } X_i \text{ gives result } 1\}|}{n}$$

so $\Pr(Q = 1) = p$. The information gain of a query Q with respect to node X is

$$\begin{aligned} I(X;Q) &= H(X) - H(X|Q) \\ &= H(X) - \Pr(Q = 0)H(X|Q = 0) - \Pr(Q = 1)H(X|Q = 1) \end{aligned}$$

Suppose X follows a uniform distribution, then X is also conditionally uniformly distributed given Q. Then

$$\begin{aligned} I(X;Q) &= \log n - (1-p)\log n(1-p) - p \log np \\ &= -(1-p)\log(1-p) - p\log p \\ &= H(p) \end{aligned}$$

Therefore, $p = \frac{1}{2}$ maximizes the information gain at each node. For more general distributions, we will prove its optimality by the analysis below.

4.4 Analysis

Recall that Huffman coding is the best we can do. We hope that GBSC can be as good as Huffman coding. However, sometimes GBSC cannot reach the optimal bound. For X in Example 4, the expected code length is 1.9 for Huffman coding and 2 for GBSC. Although GBSC is not the best coding, from experiments we observe that most of the time it is quite good, so we believe that the average code length of GBSC will not be too far away from $H(X)$. This idea inspire us to prove the following theorem (we will prove this theorem later).

Theorem 1. *Assume that the expected code length is L_g for GBSC and L_s for Shannon coding. Then $L_g \leq L_s$, i.e., GBSC is at least as good as Shannon coding.*

Recall that L_s also satisfies $L_s < H(X) + 1$, so we immediately obtain a good upper bound for L_g.

Corollary 3. *Assume that the expected code length is L_g for GBSC. Then $L_g < H(X) + 1$.*

This gives us confidence that GBSC can be applied to obtain a quite good approximation of the optimal coding. Also, it shows that if we use the same technique mentioned in Cover's textbook (see page 114), we can use GBSC to approach the Shannon bound.

Now we focus on proving Theorem 1. Recall that by definition, the code length for a symbol with probability p in Shannon coding is $\lceil -\log p \rceil$, so we only need to prove this proposition.

Proposition 5. *Assume that the code length for a symbol with probability p is L for GBSC. Then $L \leq n$ if $p \geq 2^{-n}(n \in \mathbb{Z}^+)$, or equivalently, $L \leq \lceil -\log p \rceil$.*

Proof. In Figure 7, a circle represents a single node and a rectangle represents a leaf or a subtree. The content in the nodes are indices of the nodes. Node $i(1)$ and node $i(2)$ are in the ith layer. $p_{i(j)}$ is the sum of probabilities of all the children nodes of node $i(j)$.

We prove by contradiction. WLOG, assume that $p_{k+2(1)} \geq 2^{-(k+1)}$. If $p_{k+2(2)} < p_{k+1(1)} - p_{k+1(2)}$, then moving node $k+2(2)$ to node $k+1(2)$ will make $|p_{k+1(1)} - p_{k+1(2)}|$ smaller[1], so $p_{k+2(2)} \geq p_{k+1(1)} - p_{k+1(2)}$. Thus, $p_{k+1(1)} + p_{k+1(2)} \geq 2p_{k+2(1)} + p_{k+2(2)}$. Similarly, $p_{k+2(2)} \geq p_{k(1)} - p_{k(2)}$. Thus, $p_{k(1)} + p_{k(2)} \geq 2p_{k(1)} + p_{k+2(2)} \geq 4p_{k+2(1)} + p_{k+2(2)}$. The same argument shows that $p_{1(1)} + p_{1(2)} \geq 2^{k+1}p_{k+2(1)} + p_{k+2(2)} > 1$, which is a contradiction.

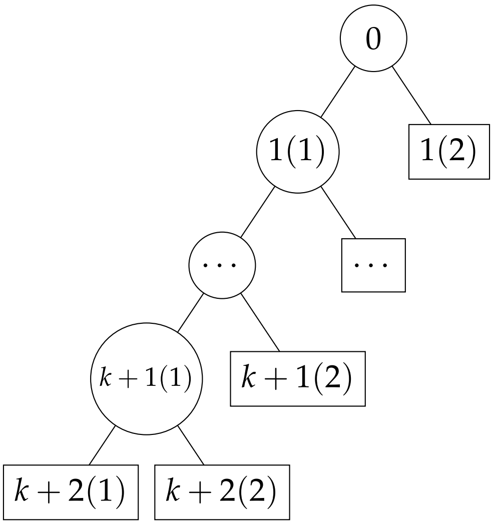


Figure 7: Illustration for the proof of Proposition 5.

□

5 Greedy Algorithms Based on GBSC

5.1 Basic Structure

The basic idea of the algorithm is shown below. In practice, the most difficult part is how to find the best question (line 2). However, since usually $|\mathscr{A}|$ is small and $\mathscr{A}$ has a simple

[1] We do not want to describe the process of "moving" mathematically, because it will just make things harder to understand. In case that some readers may feel confused, we explain the idea more clearly. Simply speaking, moving node i to node j means moving all the leaves in node i to node j so that they become the leaves of node j. If there are many leaves to be moved, the process of moving is not unique.

structure, this process is often very simple. This is especially the case when we deal with the Battleship problem later.

Algorithm 2 Greedy algorithm based on GBSC

```
procedure GBSC(node, p)
    choose question $X \in \mathscr{A}$ that makes the left and right subtrees most average
    $p_1 \leftarrow$ the possibilities in condition that the question is true
    GBSC($root.left, p_1$)
    $p_2 \leftarrow$ the possibilities in condition that the question is false
    GBSC($root.right, p_2$)
end procedure
read $p$, where $p[i]$ is the probability of $i$
initialize $root$
GBSC($root, p$)
```

5.2 Dealing with 1-player Battleship Problem Using GBSC

Recall the 1-player Battleship problem:

$$\min_{Q_1, Q_2, \ldots, Q_T} T, Q_t = (i_t, j_t)$$

$$\mathcal{X}_t = \{X | (X \in \mathcal{X}_{t-1}) \wedge (X_{i_t j_t} = X^*_{i_t j_t})\}, 1 \leq t \leq T$$

$$\mathcal{X}_0 = \{X_1, X_2, X_3, \ldots, X_n\}, \mathcal{X}_T = \{X^*\}, \text{ random variable } X^* \in \mathcal{X}_0$$

$\mathcal{X}_0$ is a set of $0 - 1$ matrices representing the entire possible board space without any prior knowledge, and X^* is the target board chosen by the opponent (randomly sampled from $\mathcal{X}_0$ in our case).

More precisely, we would like to minimize the average number of tries T over all possible targets X.

By assumption, the target board X^* is randomly sampled from $\mathcal{X}_0$. The key to minimizing the number of queries t is to choose the queries $Q_1, Q_2, \ldots, Q_T$ such that the size of remaining possible boards, $|\mathcal{X}_t|$ converges to 1 quickly.

Using the conclusions from GBSC coding, we can devise a way to minimize the average number of queries. At t-th query, the hit probability matrix is

$$P_{ij}^{(t)} = \frac{\sum_{X \in \mathcal{X}_{t-1}} X_{ij}}{|\mathcal{X}_{t-1}|}$$

Obviously, the hit probability of some previously queried (i, j) is either 0-missed or 1-hit. So there is no need to query the same grid twice.

we choose Q_t by this greedy strategy, deducted from GBSC:

$$Q_t^* = (i_t^*, j_t^*) = \arg\min_{(i,j)} |P_{ij}^{(t)} - \frac{1}{2}|$$

.

Ideally, each query (i_t^*, j_t^*) will half the size of the remaining boards space, giving us the average number of queries $\bar{T} = \lceil \log n \rceil$, where $n = |\mathcal{X}_0|$. Of course, this is not always possible in the real world, since there may not be exist a (i, j) with $p_{ij}^t = \frac{1}{2}$. Nonetheless, we can always choose the grid with hit probability closest to $\frac{1}{2}$ and get a sub-optimal algorithm.

Algorithm 3 Solving 1-player Battleship using GBSC.

```
calculate initial possible board space $\mathcal{X}_0$
$t \leftarrow 0$
while $|\mathcal{X}_t| > 1$ do
    $t \leftarrow t + 1$
    $P_{ij}^{(t)} \leftarrow \frac{\sum_{X \in \mathcal{X}_{t-1}} X_{ij}}{|\mathcal{X}_{t-1}|}$            ▷ calculate hit probability matrix
    $Q_t^* \leftarrow \arg\min_{(i,j)} |P_{ij}^t - \frac{1}{2}|$            ▷ choose locally optimal query
    $\mathcal{X}_t \leftarrow \{X | (X \in \mathcal{X}_{t-1}) \wedge (X_{i^* j^*} = X_{i^* j^*}^*)\}$            ▷ eliminate some boards in $\mathcal{X}_{t-1}$
end while
$\mathcal{X}_t = \{X^*\}$
```

We did some tests to show the effectiveness of the GBSC-based algorithm on dealing with 1-player Battleship problem. To speed up computation, we consider 3 ships of length 5, 4 and 3 placed on a 10 × 10 board. In total, there are $n = 1{,}850{,}736$ possible board layouts, which is just under 2^{21}. Here we define the number of tries t as the number of queries to determine the target board X^*.[2]Since each query (bombing) in average gives at most 1 bit of information, the theoretical minimal average number of tries is $\bar{T}^* = \lceil \log n \rceil = 21$ queries.

[2] Sometimes two or more boards may be different yet indistinguishable, and the terminal entropy is nonzero, for example if two boats of length 3 and 4 are adjacent and from a L-shape pattern. Since this does not affect the outcomes, we consider these boards to be identical.

Of course, one can sometimes get lucky and determine X^* in less than 21 tries.

Here we have 10 randomly chosen target boards for our algorithm to play against. The vertical axis is calculated by $H(X) = \log |\mathcal{X}_t|$. Again, we assume that every target board has the same probability of being chosen.

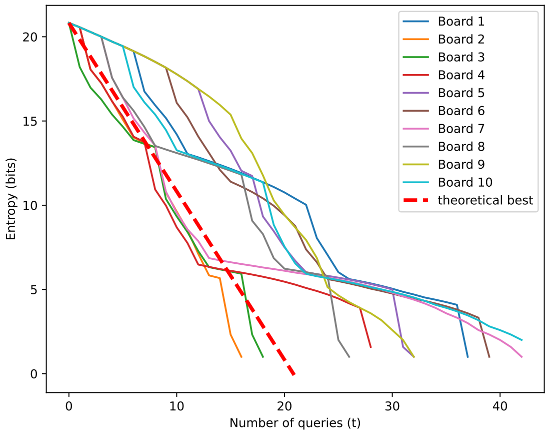


Figure 8: Algorithm against 10 random boards v.s. theoretical best average

We can observe two types of stages in the declination of entropy: at first, the entropy declination is usually slow, as the algorithm knows very little about the board and is exploring the board. When a "hit" occurs, the entropy drops rapidly, before the information provided by this hit is fully exploited, and this process repeats until entropy drops to 0.

For this small scale test, the results are very diverged. Some were very lucky and better than the theoretical best average 21, and some are almost 2 times of theoretical best average. We ran the same test on a larger scale, on 1000 randomly chosen target boards and got some interesting results.

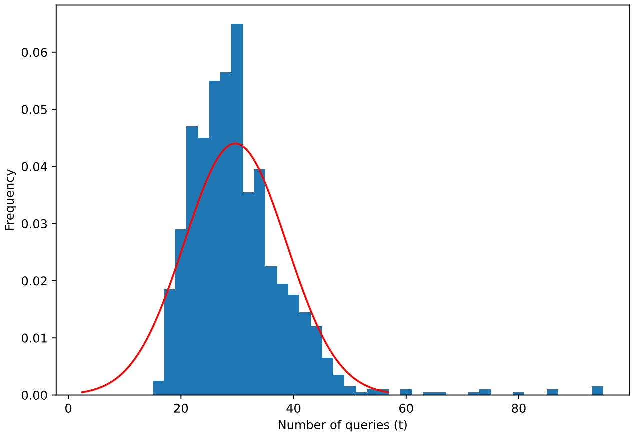
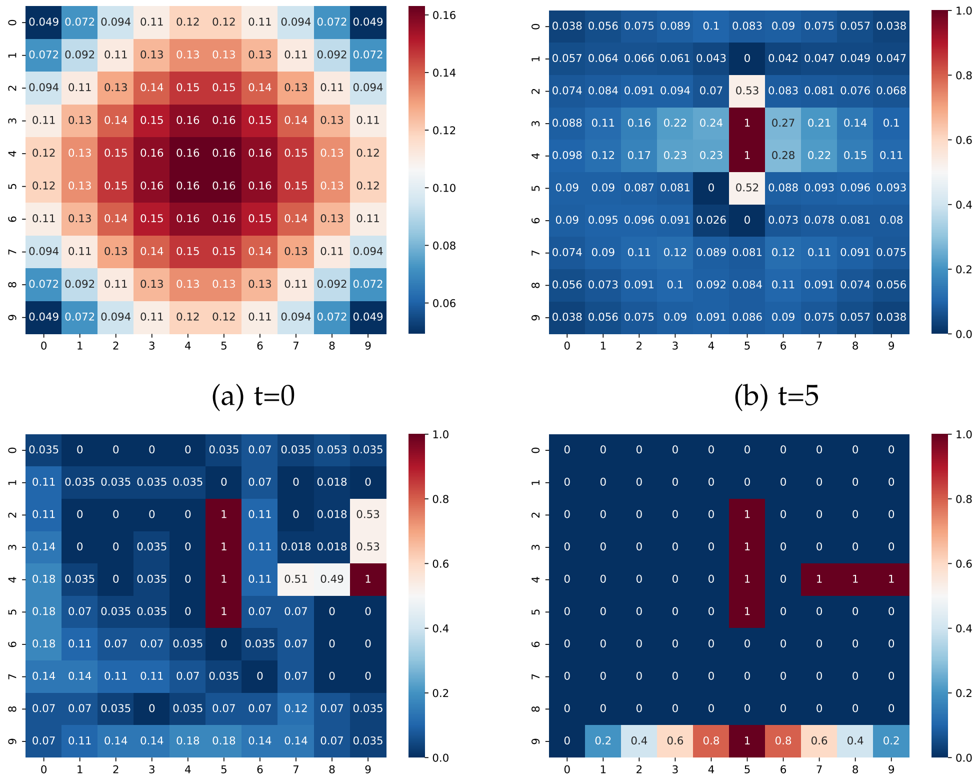


Figure 9: Distribution of queries to determine target board X^*

The mean number of queries $\bar{T} = 29.651$ and the standard deviation $\sigma = 9.062$. From this



(a) t=0

(b) t=5

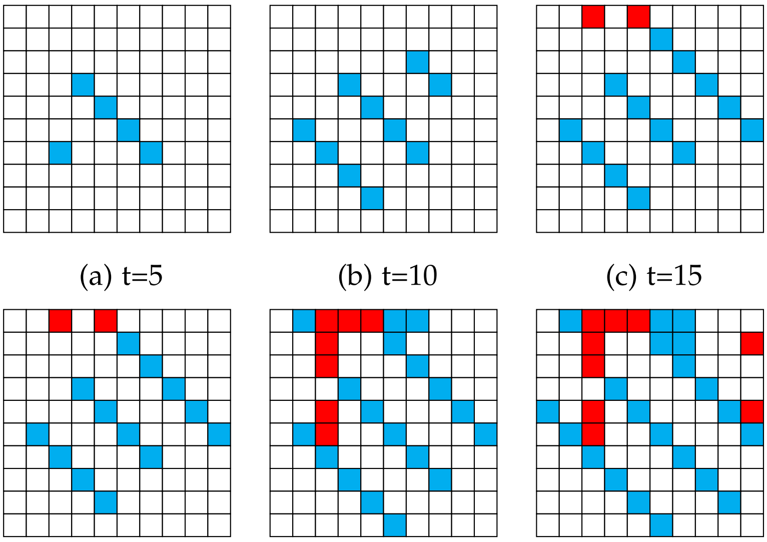
(c) t=19

(d) t=22

Figure 10: Hit probability matrix visualized. p_{ij} : 0=No ship, 1=Has ship, (0,1)=Uncertain

we can conclude that our GBSC-based algorithm performs reasonably well for Battleship, since $\bar{T} \approx 1.4\bar{T}^*$. We can also take a look at the actual strategy of our algorithm for one game. This visualization (Fig.10) shows how the algorithm prefers hit probability closest to 0.5.

In another game, we demonstrate the hit pattern of our algorithm. (Fig.11) Interestingly, our algorithm displays a diagonal search strategy, which is not intentionally designed in our GBSC based algorithm, but rather the result of computing the hit probability matrix, because a miss at (i,j) reduces the conditional probability at $(i+1,j),(i-1,j),(i,j+1),(i,j-1)$. Also notice that the diagonal bombing lines are 3 grids apart, making a 4-long ship impossible to fit in the center area. Therefore, a diagonal searching pattern is great for quickly reducing entropy. This increases our confidence about this GBSC-based algorithm, since trying to hit the ships by bombing a diagonal pattern is a well-known strategy in this classic game, which also coincides with some previous deterministic approaches[5].



(a) t=5

(b) t=10

(c) t=15

(d) t=20

(e) t=25

(f) t=30 (Terminal)

Figure 11: Hit pattern for a typical Battleship game played by GBSC-based algorithm

5.3 Another Try to Solving the DNA Detection Problem

Besides the greedy algorithm based on huffman coding, we can also use the GBSC to solve the DNA detection problem.

We compared the output and time cost of the two algorithms by using the same random seed to generate random sequences (assuming probability at every point is iid) of different lengths and using the two algorithms to compute the expected number of detections respectively. We generated 100 sequences for each length to reduce random error. Here are the results.

The average time cost for two algorithm is shown in Fig.12.

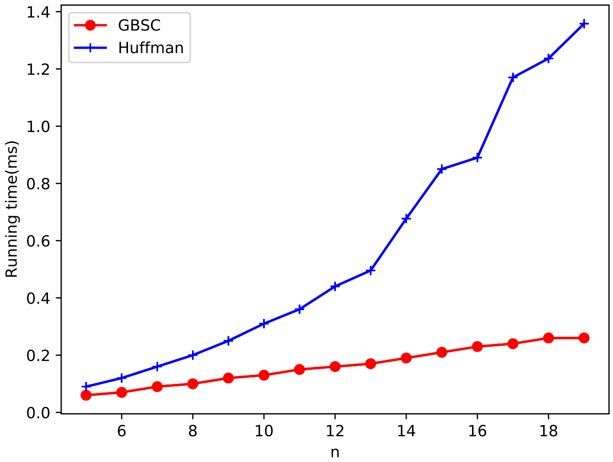


Figure 12: Running time of the two algorithms.

It can be seen from the figures that the time complexity of GBSC is nearly linear, while the time complexity of the Huffman-based algorithm is not. The GBSC-based algorithm runs much faster than the Huffman-based algorithm algorithm. Besides, the the Huffman-based algorithm is not robust, and would take quite a long time to give an output when initial data is not very ideal. For example, one of the randomly generated distributions with size 36 took it took about 30 seconds for the Huffman-based algorithm to calculate the result, while most other distributions usually took about 10ms. In comparison, the GBSC reliably gave results in a relatively short time.

The expected lengths calculated by two algorithm are shown in Fig.13.

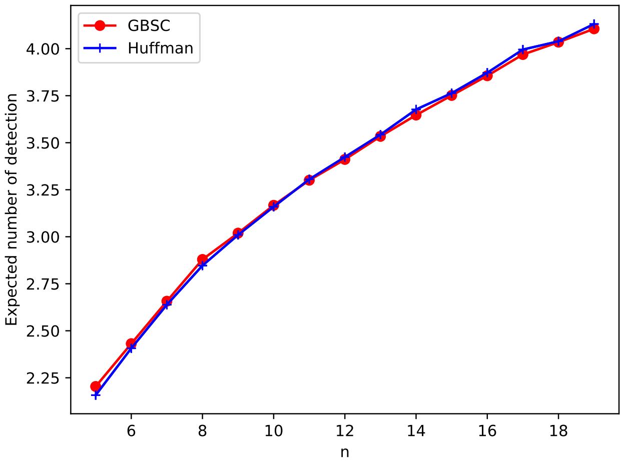


Figure 13: Expected number of detections calculated by the two algorithms.

It is easy to see that the two figures are very similar, which means that the outputs of two algorithm in our experiment are almost the same.

6 Future Work

Here we propose some aspects that can be further researched.

- For the DNA detection problem, our algorithm only works the simplified problem where there is only one target exon. A more general efficient multi-exon detection algorithm remains to be discovered.
- Finding the true optimal solution of the DNA detection problem and the Battleship problem is probably a NP-hard problem. Can we prove it?

- In this report, we implicitly assume that the distribution $p(x)$ is independent, i.e., $\forall i, j \in \mathcal{X}$, $p(i)$ and $p(j)$ are independent. If this condition is canceled, the shape of possible decision trees will not be arbitrary, making a better algorithm possible.

- Currently the our GBSC algorithm runs brute force to find the separation with probability closest to $\frac{1}{2}$. This can be time-costly for larger problems, for example, Battleship with a larger board or more ships, so there may be room for optimizations, for example maybe use Monte-Carlo method to get a good-enough approximation.

7 Conclusion

In this paper, we purposed a generalized model to similar querying problems where the questions we ask are limited to a subset $\mathscr{A} \subsetneq 2^{\mathcal{X}}$, by analyzing 3 different problems: Huffman-coding-equivalent, the 1-player Battleship problem, and the DNA detection problem.

Inspired by Huffman coding and greedy decision trees, we proposed two coding schemes: one based on Huffman coding that merges two nodes greedily if possible, and another called GBSC (Greedy Binary Separation Coding). We then proved that the expected code length of GBSC achieves the Shannon Code bound, meaning GBSC is at least as good as Shannon Coding.

To see the effectiveness of these two algorithms in real world applications, we applied the Huffman-based coding to the DNA detection problem, and GBSC to both the 1-player Battleship and the DNA detection problem to see how well they perform in terms of average coding length, or equivalently, the average number of queries to determine the target random variable. For the DNA detection problem, out of 10,000 tests, the Huffman-based algorithm gives expected length L less than $1.1L^*$ in most cases, where L^* is the true optimal calculated by brute force. We also compared the Huffman-based algorithm and GBSC algorithm, and found that while the two algorithm yielded very similar expected lengths, GBSC was much more time-efficient and therefore suitable for larger problems. In 1-player Battleship with a 10×10 board and 3 ships, out of the 1000 tests, the GBSC-based algorithm achieves an average of $\bar{T} \approx 1.4\bar{T}^* = 29.651$ queries, where $\bar{T}^*$ is the theoretical best average number of queries. Overall, we can conclude that GBSC and the Huffman-based greedy code performs decently well in solving query-based decision problems that cannot be solved by the original Huffman coding algorithm.

References

[1] T. M. Cover, *Elements of information theory*. John Wiley & Sons, 1999.

[2] T. Biedl, B. Brejová, E. D. Demaine, A. M. Hamel, A. López-Ortiz, and T. Vinař, “Finding hidden independent sets in interval graphs,” *Theoretical Computer Science*, vol. 310, no. 1-3, pp. 287–307, 2004.

[3] G. Xu, S.-H. Sze, C.-P. Liu, P. A. Pevzner, and N. Arnheim, “Gene hunting without sequencing genomic clones: finding exon boundaries in cdnas,” *Genomics*, vol. 47, no. 2, pp. 171–179, 1998.

[4] M. Audinot, F. Bonnet, and S. Viennot, “Optimal strategies against a random opponent in battleship.”

[5] E.Y.Rodin, “Developing a strategy for battleship,” *Mathematical and Computer Modelling*, pp. 145–153, 1988.

[6] L. Clementis, “Supervised and reinforcement learning in neural network based approach to the battleship game strategy,” pp. 191–200, 2013.

Appendix

The code of the experiments in the paper is available at `https://github.com/MadCreeper/Constrained-Optimal-Querying-Huffman-Coding-and-Beyond`.