Binary Matrix Guessing Problem
-Draft-

Çağrı Latifoğlu, PhD
cagri.latifoglu@tedu.edu.tr
TED University, Industrial Engineering Department,
Ziya Gökalp Caddesi No:48 06420, Kolej, Çankaya, ANKARA, TURKEY

January 24, 2017

Abstract

We introduce the Binary Matrix Guessing Problem and provide two algorithms to solve this problem. The first algorithm we introduce is Elementwise Probing Algorithm (EPA) which is very fast under a score which utilizes Frobenius Distance. The second algorithm is Additive Reinforcement Learning Algorithm which combines ideas from perceptron algorithm and reinforcement learning algorithm. This algorithm is significantly slower compared to first one, but less restrictive and generalizes better. We compare computational performance of both algorithms and provide numerical results.

1 Introduction

Consider the following game: Alice creates an $n \times n$ square binary matrix $\mathcal{A}$ with elements $[a_{ij}] \in \{0,1\} \quad \forall i,j \in \{1,\dots,n\}$. $\mathcal{A}$ is hidden from Bob. Bob is trying to guess $\mathcal{A}$. Alice never reveals $\mathcal{A}$ however returns a score if Bob submits a guess $\mathcal{B}$ where $[B_{ij}] \in \{0,1\} \quad \forall i,j \in \{1,\dots,n\}$. The scoring function f takes $\mathcal{A}$ and $\mathcal{B}$ as inputs and returns score $s \in \mathbb{R}$. Note that Bob does not know scoring function f or $\mathcal{A}$ and can only observe s values for his guess $\mathcal{B}$[1]. We will assume the maximum value of score s, s^{max}, is known by both parties, and is attained when $\mathcal{A} = \mathcal{B}$. Furthermore we will assume the function f provides "informative" values of s for partially correct guesses. For example a scoring function which returns 1 if $\mathcal{A} == \mathcal{B}$ and 0 otherwise is not really informative as the Bob can not infer the correct positions of 0s and 1s from partial matches. However, a scoring function that utilizes the Frobenius Distance (FD) between $\mathcal{A}$ and $\mathcal{B}$ can be considered informative as one can measure a sense of proximity. For the rest of the paper, we will assume the scoring function, f, is defined as in the following[2]:

$$FD(\mathcal{A},\mathcal{B}) = \sqrt{trace((\mathcal{A}-\mathcal{B}) * (\mathcal{A}-\mathcal{B})^\top)} \tag{1}$$

$$s^{max} = FD(\mathcal{C},\mathcal{D}) \tag{2}$$

$$f(\mathcal{A},\mathcal{B}) = s^{max} - FD(\mathcal{A},\mathcal{B}) = s \tag{3}$$

where $(\mathcal{A}-\mathcal{B})^\top$ is the conjugate transpose of $(\mathcal{A}-\mathcal{B})$, and,

$$\mathcal{C}_{n\times n} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{pmatrix}, \quad \mathcal{D}_{n\times n} = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}.$$

[1] One might consider brute force enumeration to guess the target matrix. For matrices of small dimensions, this approach might be viable, however the number of possibilities increases exponentially as dimension of the matrix increase. For example for a 10×10 matrix, there are 2^{100} possibilities

[2] One could have approached this problem as a matrix distance minimization problem, i.e. smaller distances correspond to closer matches between the guess matrix, and the target matrix, however in our setting we would like use the term "score" instead of "distance" so we subtract the distance between target matrix and guess matrix from the maximum possible distance,s^{max}.

FD between two binary matrices of equal dimensions, is simply square root of sum of squares of elementwise differences. See Appendix A. Next we present our first solution approach, the Elementwise Probing Algorithm.

2 Elementwise Probing

Consider the following: Let $\mathcal{B}^0$ and $\mathcal{B}^1$ be identical $n \times n$ binary matrices except at index $\{l, m\}$, therefore $\mathcal{B}^0_{i,j} = \mathcal{B}^1_{i,j}\ \forall (i,j) \in \{(i,j) | \{1,\ldots,n\} \times \{1,\ldots,n\} \setminus \{l,m\}\}$. Without loss of generality, assume $\mathcal{B}^0_{l,m} = 0$ and $\mathcal{B}^1_{l,m} = 1$. At the other indices these two matrices are exactly the same. Now consider the scores they would get when evaluated against an arbitrary target binary matrix, $\mathcal{A}$: $s_0 = f(\mathcal{A}, \mathcal{B}^0)$ and $s_1 = f(\mathcal{A}, \mathcal{B}^1)$. It is trivial to see that $s_0 \neq s_1$ and either $s_0 > s_1$ or $s_0 < s_1$. Furthermore, $\mathcal{A}_{l,m} = 0 \implies s_0 > s_1$ and $\mathcal{A}_{l,m} = 1 \implies s_0 < s_1$ since scores increase as corresponding entries in matrices $\mathcal{A}$ (target) and $\mathcal{B}$ (guess) match. See Appendix B.
Utilizing this observation, we can construct the following algorithm:

Algorithm 1 Elementwise Probing Algorithm (EPA)

```
Randomly create a binary (guess) matrix B
for all i ∈ {1,...,n} do
  for all j ∈ {1,...,n} do
    Set B_{i,j} = 0, get score s_0
    Set B_{i,j} = 1, get score s_1
    (Note that s_0 ≠ s_1)
    if s_0 > s_1 then
      Guess A_{i,j} as 0
      Set B_{i,j} as 0
    else
      Guess A_{i,j} as 1
      Set B_{i,j} as 1
    end if
  end for
end for
```

With Algorithm 1, after a maximum of $2 \times n^2$ FD evaluations, we can extract the target matrix $\mathcal{A}$. We've implemented a serial[3] version of this algorithm in Python, run the code on a computer with Intel Core i7-920 processor with base frequency 2.66 GHz and 18 GB RAM, with Ubuntu 16.04 as the operating system. The computational results are presented in Table 1.
In Table 1, GridSize is simply equal to n, ACC is the percentage of times the Algorithm 1 completely extracted the target in 100 trials (1.00 means 100% success), timeAVR is average solution time (in seconds) of 100 trials, and timeSD is the standard deviation of the solution times (in seconds) of 100 trials.
In the next section we present our second solution algorithm: Additive Reinforcement Learning Algorithm.

3 Additive Reinforcement Learning Algorithm

For solving the binary matrix guessing problem, we have developed a perceptron-like [1] learning algorithm, called Additive Reinforcement Learning Algorithm (ARLA) which learns the target matrix after making a certain number of queries. ARLA, in spirit, is close to perceptron and reinforcement learning [2], as close matches between the guess and the target reinforces ARLA's current guess of the positions of 0s and 1s in the target matrix.
Perceptron algorithm is a supervised learning algorithm. It solves a binary classification problem, uses vector inputs, and updates weights of the elements in the vector in an additive manner. In

[3]Note that Algorithm 1 is easily parallelizable, as each index pair (i, j) can be tested independently from other pairs.

GridSize	Acc	timeAVR	timeSD
10	1.00	0.00	0.00
20	1.00	0.01	0.00
30	1.00	0.04	0.00
40	1.00	0.07	0.00
50	1.00	0.14	0.00
60	1.00	0.22	0.00
70	1.00	0.35	0.00
80	1.00	0.52	0.01
90	1.00	0.74	0.01
100	1.00	1.11	0.11
110	1.00	1.56	0.17
120	1.00	1.96	0.01
130	1.00	4.66	0.03
140	1.00	3.40	0.01
150	1.00	4.46	0.02
160	1.00	5.73	0.09
170	1.00	7.14	0.03
180	1.00	8.83	0.66
190	1.00	10.86	0.36
200	1.00	13.40	0.93

Table 1: Elementwise Probing Results

reinforcement learning there are multiple components which are states, actions, rewards, and various rules that capture state transitions, and reflect the constraints on learning agent's perception of the environment. The objective is to learn a policy that would map the states to actions so that the reward the agent obtains is maximized.

From a reinforcement learning perspective, the scores in ARLA can be classified as rewards, and the guesses can be classified as states. However there are no constraints on state transition, states do not depend on each other or past history, as each guess is created randomly and independently from other guesses. Due to this independence property, ARLA training is embarrassingly parallelizable.

ARLA algorithm combines the additive updates feature of the perceptron algorithm and the reward for state idea of reinforcement learning. Unlike perceptron algorithm, in ARLA the inputs are matrices (i.e. guesses) instead of vectors. Rewards of the guesses (i.e. scores in our case) are multiplied with the guess matrix and added on top of each other during the training period.

In its current form, ARLA requires many samples (i.e. a guess, $\mathcal{B}$, and its score, s) for training to deduce the correct target, $\mathcal{A}$. The number of samples required increases as the matrix dimensions increase. To model this dependence, we will first define the number of queries ARLA makes (i.e. the number of samples it will collect) during the training period as $sc \times n$ where n is the dimension of the matrix, and sample coefficient, sc, is a multiplier that is selected empirically. In our numerical experiments, we observed that setting sc to 10000 for a 10×10 matrix is enough. This means during the training period, ARLA will use 10×10000 samples.

To illustrate the algorithm we will first define an all zeros matrix:

$$\mathcal{E}_{n\times n} = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

During the training period, we will capture the information about the guess and its score in $\mathcal{E}$.

We will randomly create a binary matrix $\mathcal{B}$, obtain $s = f(\mathcal{A}, \mathcal{B})$, update $\mathcal{E}$ and repeat this process $sc \times n$ times. $\mathcal{E}$ is updated during the sampling period in the following manner:

Algorithm 2 Additive Learning Algorithm (ARLA)

1: **for all** $i \in \{1, \ldots, n * sc\}$ **do**
2: Randomly create $n \times n$ binary guess matrix $\mathcal{B}$
3: Get $s = f(\mathcal{A}, \mathcal{B})$
4: $\mathcal{E} \leftarrow \mathcal{E} + s * \mathcal{B}$
5: **end for**

In Algorithm 2, the key idea is good guess layouts which give high scores give two key pieces of information: the layout itself, namely the current guess $\mathcal{B}$ and the score of this layout, namely s. We combine these two pieces by multiplying them and adding the resulting matrix to $\mathcal{E}$, so that after learning period terminates, the accumulated score and layout information will be captured in $\mathcal{E}$ in a cumulative manner.
If the learning period was long enough, the positions of 1s and 0s in the target matrix, $\mathcal{A}$, can be deduced from $\mathcal{E}$. The key observation here is, the scores accumulated in $\mathcal{E}$ at positions that correspond to positions of 1s in $\mathcal{A}$ will be greater than the scores accumulated in $\mathcal{E}$ at positions that correspond to positions of 0s in $\mathcal{A}$.
Following this idea, first we find the minimum of $\mathcal{E}$, and call it $m = min(\mathcal{E})$. Now subtract m from all elements of $\mathcal{E}$.

$$m = min(\mathcal{E}) \tag{4}$$

$$\mathcal{E} := \mathcal{E} - m \tag{5}$$

Then we take the average of all elements of $\mathcal{E}$, and call it avg. Then we create a matrix of all zeros,

$$\mathcal{F}_{n\times n} = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

then we do the following operation:

Algorithm 3 Additive Learning Algorithm: Final Inference

1: **for all** $i \in \{1, \ldots, n\}$ **do**
2: **for all** $j \in \{1, \ldots, n\}$ **do**
3: **if** $\mathcal{E}_{i,j} \geq avg$ **then**
4: $\mathcal{F}_{i,j} = 1$
5: **else**
6: $\mathcal{F}_{i,j} = 0$
7: **end if**
8: **end for**
9: **end for**

The output of Algorithm 3, $\mathcal{F}$, will be equal to $\mathcal{A}$ if the learning period was long enough. The probability of $\mathcal{F}$ converging to $\mathcal{A}$ is dependent on the length of the sampling process. We have implemented ARLA on Python, the computational results for different dimension sizes, and different sampling coefficients, can be seen in Table C2, Table C3, and Table C4. [4] GridSize is simply the dimension of the matrices, n, SampleCoefficient is the multiplier we use to control the number of training samples, Accuracy the percentage of times Algorithm 2 completely extracted the target in 100 trials (1.00 means 100% success), timeAVR is average solution time (in seconds) of 100 trials, and timeSD is the standard deviation of the solution times (in seconds) of 100 trials.
Our results on performance of ARLA for this problem are empirical. We are currently working on the mathematical proof of ARLA, and will present the results along with convergence properties and

[4] One can easily parallelize the training period by distributing the workload of the for loop in Algorithm 2 to multiple computers.

characterization of the optimal number of samples required for convergence to the target matrix in a future publication.
Compared to EPA, ARLA is orders of magnitude slower. For successful target matrix recovery, it needs a lot of samples. The largest instance reported in Table C4 has $n = 15$ and the average solution time is 1145.78 seconds. ARLA is embarrassingly parallelizable and the solution time decreases linearly with the number of compute instances allocated to training period. With 10 computers working in parallel, it would take approximately 113 seconds to solve the 15×15 instance with ARLA which is still significantly slower than EPA.

4 Conclusion

We have introduced the Binary Matrix Guessing Problem and provided two algorithms to solve this problem: Elementwise Probing Algorithm (EPA) and Additive Learning Algorithm (ARLA). EPA is very fast and can be made even faster if parallelized however it requires a scoring metric that is responsive to elementwise changes. ARLA can also be used to solve the same problem and compared to EPA it generalizes better, however it is slower compared to EPA but can be improved speedwise since its embarrassingly parallelizable. We have used ARLA to design a photonic crystal with very good focusing/coupling properties [citation goes here] where the scoring function doesn't have necessarily good qualities, such as elementwise sensitivity, as we have in Frobenius Distance.
As future work, ARLA will be extended to consider the rewards from complements. Consider a guess matrix $\mathcal{B}$ which is the complement of the target matrix $\mathcal{A}$. This means wherever $\mathcal{A}$ has zeros $\mathcal{B}$ has ones, and $\mathcal{A}$ has ones $\mathcal{B}$ has zeros. The score of $f(\mathcal{A}, \mathcal{B}) = 0$ since $FD(\mathcal{A}, \mathcal{B}) = n$. So in its current form of ARLA, during the training period, when we do the following update $\mathcal{E} \leftarrow \mathcal{E} + 0 * \mathcal{B}$, $\mathcal{E}$ doesn't change. However a smarter algorithm might recognize having a zero score would mean complementing current guess could yield the target, and act accordingly.
The other research direction is evaluating the performance of a multiplicative algorithm such as Winnow Algorithm [3] for Binary Matrix Guessing Problem.

5 Acknowledgements

The author thanks Dr. Tayfun Küçükyılmaz, Dr. Mirbek Turduev, Dr. Utku İnan Türkmen and Dr. Sinan Hanay from TED University for the helpful discussions. We have designed these algorithms to solve a problem posed by Dr. Mirbek Turduev. Dr. Tayfun Küçükyılmaz provided keen observations during the design process of EPA. Dr. Utku İnan Türkmen provided helpful comments on ARLA algorithm. Dr. Sinan Hanay provided helpful comments throughout the development process.

References

[1] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psych. Rev.*, 65:386–407, 1958. (Reprinted in *Neurocomputing* (MIT Press, 1988).).

[2] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[3] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine learning*, 2(4):285–318, 1988.

Appendix A FD and s^{max}

Consider two 2×2 binary matrices $\mathcal{Y}$ and $\mathcal{Z}$:

$$\mathcal{Y}_{2\times 2} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \quad \mathcal{Z}_{2\times 2} = \begin{pmatrix} e & f \\ g & h \end{pmatrix}.$$

$FD(\mathcal{Y}, \mathcal{Z})$ is simply

$$\sqrt{(a-e)^2 + (b-f)^2 + (c-g)^2 + (d-h)^2}.$$

If $\mathcal{Y}$ is all zeros matrix, and $\mathcal{Z}$ is all ones matrix, $FD(\mathcal{Y}, \mathcal{Z}) = \sqrt{1+1+1+1} = 2$. In general for $n \times n$ binary matrices, the maximum FD that can be attained (when one of the matrices is all zeros, and the other one all ones) is $\sqrt{n^2} = n$. So under FD assumption, maximum value of s^{max} is n for $n \times n$ binary matrix inputs.

Appendix B Matrix Proof

Consider two 2×2 binary matrices $\mathcal{K}$ and $\mathcal{L}$ which will be used as our guess matrices:

$$\mathcal{K}_{2\times 2} = \begin{pmatrix} a & b \\ 0 & d \end{pmatrix}, \quad \mathcal{L}_{2\times 2} = \begin{pmatrix} a & b \\ 1 & d \end{pmatrix}.$$

Note that $\mathcal{K}$ and $\mathcal{L}$ are identical except at index $(2,1)$. $\mathcal{K}_(2,1) = 0$ and $\mathcal{L}_(2,1) = 1$. Also note that, $s^{max} = 2$ since $n = 2$.
Consider the 2×2 binary matrix $\mathcal{M}$ which will be used as our target matrix:

$$\mathcal{M}_{2\times 2} = \begin{pmatrix} e & f \\ g & h \end{pmatrix}.$$

$$FD(\mathcal{K}, \mathcal{M}) = \sqrt{(a-e)^2 + (b-f)^2 + (0-g)^2 + (d-h)^2}.$$
$$FD(\mathcal{L}, \mathcal{M}) = \sqrt{(a-e)^2 + (b-f)^2 + (1-g)^2 + (d-h)^2}.$$

In above equations, note that except the terms involving g, all the other squared difference terms are identical. This helps us compare $FD(\mathcal{K}, \mathcal{M})$ and $FD(\mathcal{L}, \mathcal{M})$ without knowing the values of a, e, b, f, d, h.
Let

$$s_0 = f(\mathcal{K}, \mathcal{M}) = 2 - FD(\mathcal{K}, \mathcal{M})$$

and

$$s_1 = f(\mathcal{L}, \mathcal{M}) = 2 - FD(\mathcal{L}, \mathcal{M})$$

Now g, which is the element at $(2,1)$ position in the target binary matrix $\mathcal{M}$, is either 0 or 1.
If $g = 0$ then $FD(\mathcal{K}, \mathcal{M}) < FD(\mathcal{L}, \mathcal{M})$, and consequently $s_0 > s_1$. If $g = 1$ then $FD(\mathcal{K}, \mathcal{M}) > FD(\mathcal{L}, \mathcal{M})$, and consequently $s_0 < s_1$.

Appendix C ARLA Computational Results

GridSize	SampleCoefficient	Accuracy	timeAVR (secs)	timeSD (secs)
4	10000	1.00	2.50	0.14
4	20000	1.00	5.01	0.29
4	30000	1.00	7.51	0.43
4	40000	1.00	10.00	0.61
4	50000	1.00	12.52	0.71
4	60000	1.00	15.05	0.91
4	70000	1.00	17.61	1.25
4	80000	1.00	20.03	1.22
4	90000	1.00	22.52	1.30
4	100000	1.00	25.03	1.45
5	10000	1.00	3.87	0.26
5	20000	1.00	7.72	0.52
5	30000	1.00	11.58	0.78
5	40000	1.00	15.43	1.04
5	50000	1.00	19.32	1.33
5	60000	1.00	23.22	1.62
5	70000	1.00	27.16	2.11
5	80000	1.00	30.85	2.09
5	90000	1.00	34.71	2.35
5	100000	1.00	38.59	2.61
6	10000	1.00	5.66	0.44
6	20000	1.00	11.29	0.68
6	30000	1.00	16.95	1.04
6	40000	1.00	22.57	1.37
6	50000	1.00	28.23	1.73
6	60000	1.00	33.92	2.07
6	70000	1.00	39.57	2.44
6	80000	1.00	45.18	2.72
6	90000	1.00	50.78	2.91
6	100000	1.00	56.26	2.77

Table C2: Additive Reinforcement Learning Computational Results 1

GridSize	SampleCoefficient	Accuracy	timeAVR (secs)	timeSD (secs)
7	10000	1.00	8.08	0.57
7	20000	1.00	16.14	1.11
7	30000	1.00	24.24	1.70
7	40000	1.00	32.32	2.24
7	50000	1.00	40.35	2.79
7	60000	1.00	48.42	3.35
7	70000	1.00	56.44	3.96
7	80000	1.00	64.55	4.25
7	90000	1.00	72.82	5.75
7	100000	1.00	80.81	5.28
8	10000	1.00	11.05	0.74
8	20000	1.00	22.18	1.47
8	30000	1.00	33.35	2.42
8	40000	1.00	44.45	3.01
8	50000	1.00	55.93	4.47
8	60000	1.00	67.20	5.68
8	70000	1.00	78.49	6.44
8	80000	1.00	89.29	6.95
8	90000	1.00	100.11	6.56
8	100000	1.00	110.73	5.72
9	10000	1.00	14.79	0.98
9	20000	1.00	29.53	1.96
9	30000	1.00	44.34	2.88
9	40000	1.00	59.05	3.43
9	50000	1.00	73.88	4.89
9	60000	1.00	88.45	5.05
9	70000	1.00	103.35	6.07
9	80000	1.00	118.30	7.87
9	90000	1.00	133.38	9.18
9	100000	1.00	148.22	10.25
10	10000	0.99	19.40	1.53
10	20000	1.00	38.81	3.02
10	30000	1.00	58.13	4.21
10	40000	1.00	77.29	5.23
10	50000	1.00	96.43	5.55
10	60000	1.00	115.80	6.77
10	70000	1.00	135.03	7.83
10	80000	1.00	154.55	10.13
10	90000	1.00	173.59	9.64
10	100000	1.00	193.07	11.72

Table C3: Additive Reinforcement Learning Computational Results 2

GridSize	SampleCoefficient	Accuracy	timeAVR (secs)	timeSD (secs)
11	100	0.00	0.50	0.00
11	1000	0.00	4.99	0.01
11	10000	0.89	49.85	0.22
11	100000	1.00	497.92	1.18
12	100	0.00	0.65	0.06
12	1000	0.00	6.49	0.24
12	10000	0.72	64.72	2.14
12	100000	1.00	647.08	20.84
13	100	0.00	0.82	0.15
13	1000	0.00	7.92	0.28
13	10000	0.41	80.08	7.90
13	100000	1.00	798.90	70.81
14	100	0.00	0.96	0.16
14	1000	0.00	9.56	1.59
14	10000	0.15	94.81	7.17
14	100000	1.00	945.17	49.22
15	100	0.00	1.15	0.05
15	1000	0.00	11.48	0.22
15	10000	0.01	114.62	2.17
15	100000	1.00	1145.78	21.49

Table C4: Additive Reinforcement Learning Computational Results 3