Randomized Algorithms (RA-MIRI): Assignment #2

Marc Díaz (marc.diaz.calderon@estudiantat.upc.edu)

November 2024

1 Code Solution

The class `GaltonBoard` will be in charge of performing the simulations. In its constructor we can specify the following parameters:

1. The number of bins to use. This sets the `number_of_bins_` variable and initializes the vector `bins_`, which will contain the number of balls in each bin.
2. The number of balls to throw, which sets the `balls_` variable.
3. The allocation scheme to use. This can accept either a real value $\beta \in (0, 1)$ for the $(1 + \beta)$-choice scheme or an integer $d \geq 1$ for the d-choice scheme.
4. Lastly, an integer $k \in \{0, 1, 2\}$ to model the uncertainty. A value of 0 indicates no uncertainty, while 1 or 2 represents the number of questions asked following the assignment.

The main function creates a simulation environment for various values of $n \in [0, m^2]$, where m is the number of bins in the simulation. Each simulation is repeated T times, with T defaulting to 100 trials. For each simulation, we instantiate the `GaltonBoard` class, call the `Run()` method, and gather the maximum gap using `CalculateMaxGap()`. The average maximum gap is computed over the 100 trials and saved in a CSV file.

The `Run()` method can be found in the following pseudocode 1. This function accepts a parameter `batch_size`, specifying the number of balls to process per batch without immediately updating the `bins_` vector.

Algorithm 1: Simulation - Run Function

Input : *batch_size* – number of balls processed per batch; default is 1 ball per batch
Output : Updated bin counts in array *bins_*

```
full_batches ← number_of_balls/ batch_size;
remaining_balls ← number_of_balls % batch_size;
for b = 0 to full_batches do
    Call SimulateBatch with batch_size;
end
if remaining_balls > 0 then
    Call SimulateBatch with remaining_balls;
end
```

The `SimulateBatch()` method is the core function. It first selects the number of randomly chosen bins based on the scheme specified in the constructor. If uncertainty $k > 0$ is used, the function `ChooseBinUncertainty()` selects the bin for each ball; otherwise, `ChooseBin()` is called. After processing each batch, the `bins_` vector is updated for the next batch.

Algorithm 2: Galton Board Simulation - Simulate Batch

Input : *batch_size* – number of balls in this batch
Output : Updated bin counts after simulating one batch

```
for i ← 0 to batch_size do
    number_possible_bins ← Result of ChooseScheme();
    chosen_bins ← Randomly chosen bins via rejection method (total count is
     number_possible_bins);
    // Select appropriate bin based on k_choices
    batch_counts [i] ← if k_choices = 0 then
        ChooseBin from chosen_bins;
    else
        ChooseBinUncertainty from chosen_bins;
    end
end
// Update bin counts
foreach chosen_bin in batch_counts do
    Increment bins[chosen_bin];
end
```

Finally, to analyse the evolution of the Gap, for each number N of balls, we calculate the maximum gap using `CalculateMaxGap()` as previously described. The formula for the gap is:

$$G_n = \max_{1 \le i \le n} \left\{ X_i(n) - \frac{n}{m} \right\} \tag{1}$$

Algorithm 3: Galton Board Simulation - Calculate Max Gap

Input : *load_factor* – average expected bin load
Output : Maximum gap between bin load and load factor

```
max_gap ← 0;
foreach bin_count in bins do
    gap ← Difference between bin_count and load_factor;
    if gap > max_gap then
        max_gap ← gap;
    end
end
return max_gap;
```

The code solution has been implemented in C++ and can be found in this GitHub repo - (https://github.com/MDCmarc/RA-Assigment2). For random number generation, we utilized the `mt19937` random engine, which is based on the classic Mersenne Twister algorithm [1].

In the experimentation, to run each experiment and trial we used OpenMP for parallelization, as they are independent. However, parallelization was not applied within each simulation, as the probability of a ball landing in a bin depends on the outcomes of prior iterations.

2 Experimentation

2.1 Different Schemes Comparison

Figure 1 and 2 show the evolution of the average Gap for all the number of balls from $n = 0$ to $n = m^2$ for the d-choice schema with values 1 and 2 respectively. We are using 25 bins for the experimentation hence the 625 balls. As we can clearly see, only adding one more choice brings down the factor when $n = m$ to half of what we had and almost 10 times less when $n = m^2$.

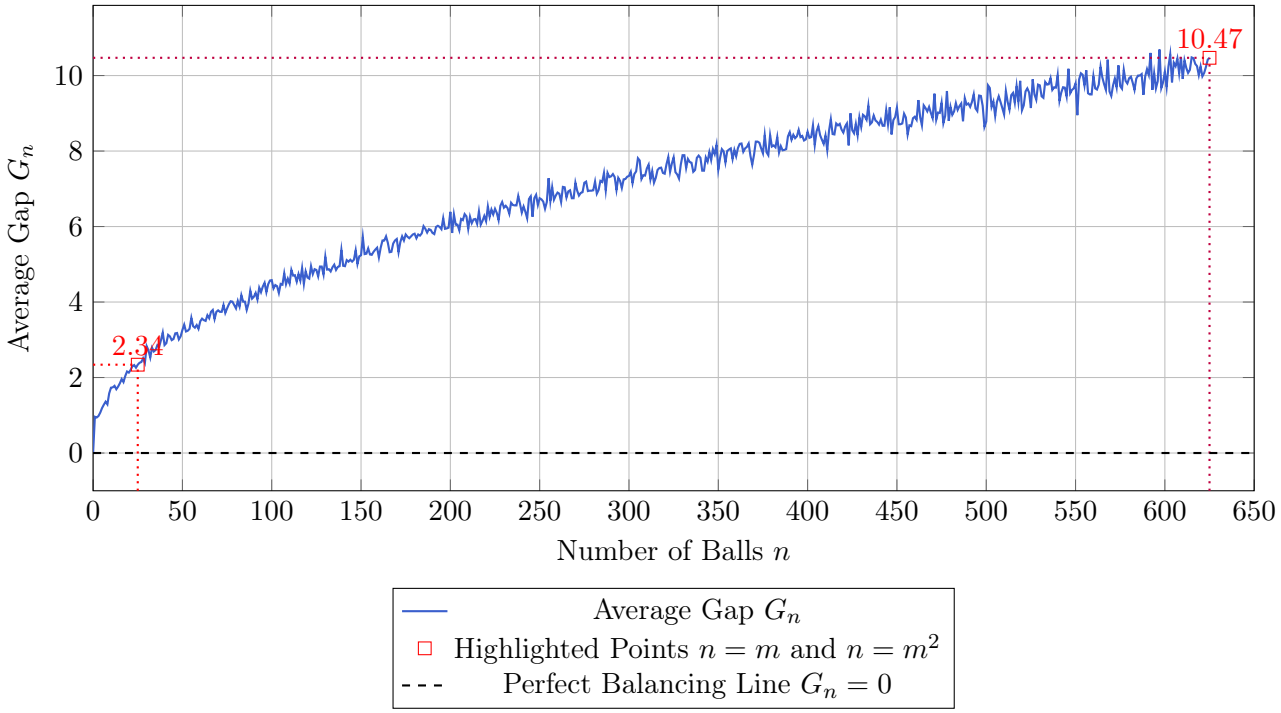

Figure 1: Average Gap G_n for 100 Trials and 25 Bins, d-Choice Scheme (d=1)

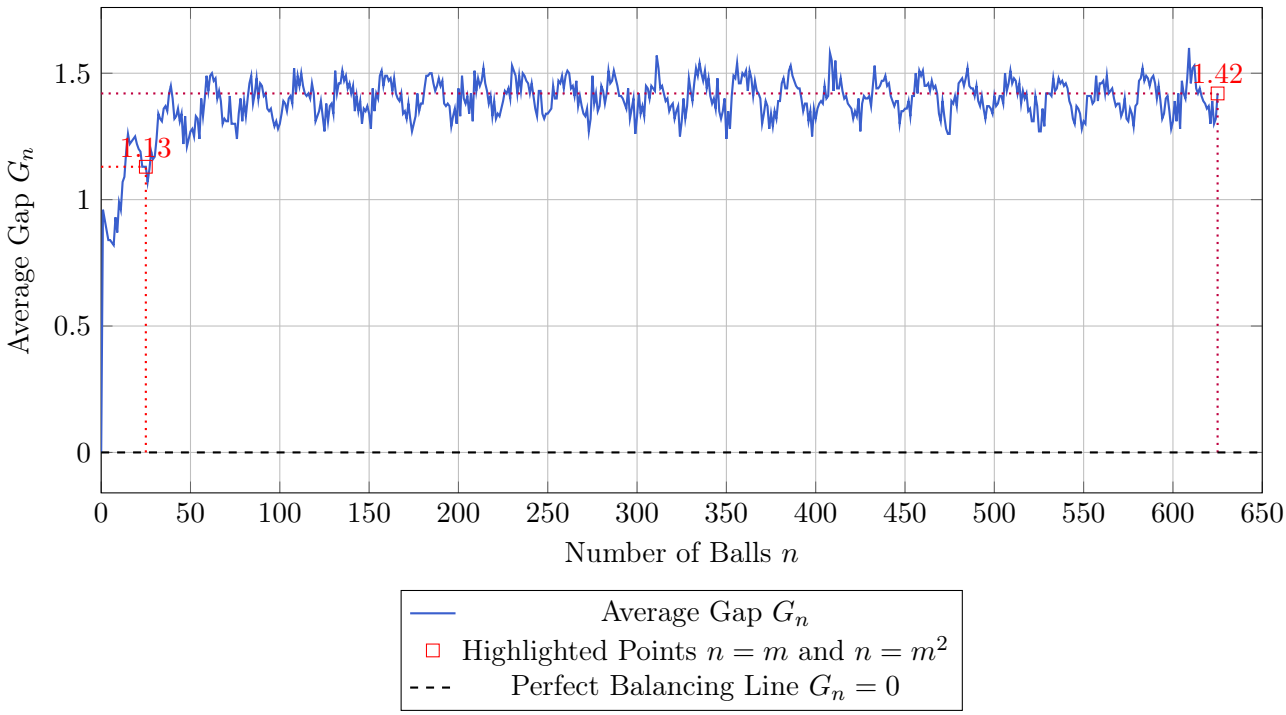

Figure 2: Average Gap G_n for 100 Trials and 25 Bins, d-Choice Scheme (d=2)

If instead of the d-choice scheme we use the $(1+\beta)$-choice scheme the average Gap will vary, depending on the value of β, within the results obtained with d=1 and d=2 we showed before. Figure 3 shows the evolution of the average Gap for all the number of balls from $n = 0$ to $n = m^2$ again for the diferent β values from 0.1 to 0.9. As expected, the higher the β, the closer we get to d=1-scheme hence the worst we will perform.

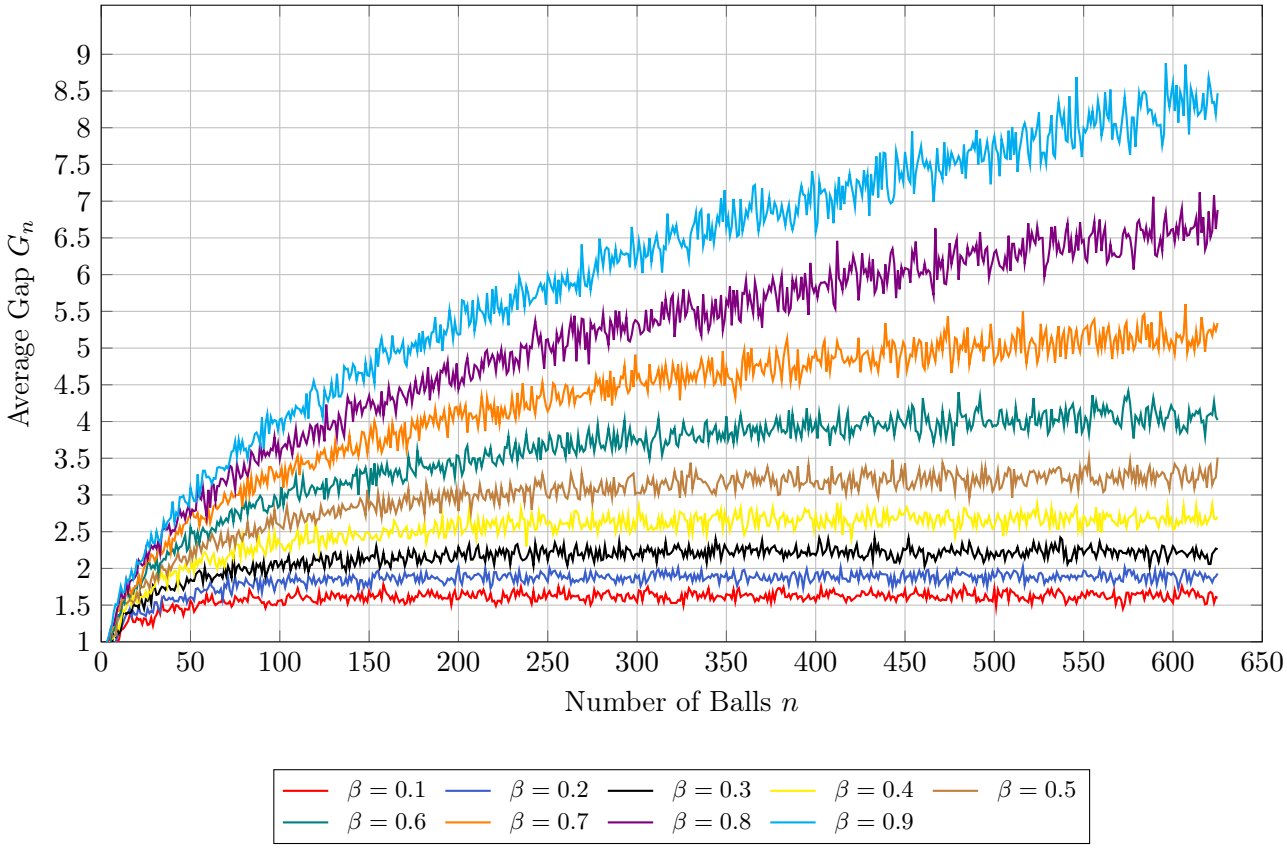

Figure 3: Average Gap G_n for 100 Trials and 25 Bins for different β values

Lastly, Table 1 shows the average Gap value of the 100 trials for the critical values $n = m$ and $n = m^2$ for different delta values. As expected, the higher the delta, i.e. the number of bins to choose from, the better we will do spreading the load equally among the bins.

d **Value**	$n = m$	$n = m^2$
3	1.0	1.01
4	1.0	1.0
5	1.0	1.0
6	1.0	0.98
7	0.98	0.96
8	0.92	0.91
9	0.87	0.87
10	0.78	0.81

Table 1: Comparison of maximum gap values for greater d-values with $n = m$ and $n = m^2$.

2.2 Adding Uncertainty: B-batching

For the second part of the experiment we performed the same simulation of 100 trials with 25 bins for all balls from $n = 0$ to $n = m^2$ but using batches. That means, we will throw the amount of balls specified in the batch size *without updating the counts in the bins*. This will then make the experiment more erratic.

Table 2 summarizes the critical points $n = m$ and $n = m^2$ for three different β values for different batches sizes that increase with the form $n = m, n = 2m, \ldots, n = \lambda m$ being $\lambda = 25$. As we can see, the more decisions we make without having updated values, the worse we perform.

Batch Size	$\beta = \mathbf{0.25}$		$\beta = \mathbf{0.5}$		$\beta = \mathbf{0.75}$	
	$n = m$	$n = m^2$	$n = m$	$n = m^2$	$n = m$	$n = m^2$
25	2.42	2.75	2.31	3.58	2.45	5.92
50	2.38	3.08	2.33	3.94	2.44	6.36
75	2.38	3.54	2.37	3.91	2.32	6.45
100	2.4	3.9	2.4	4.56	2.4	6.79
125	2.38	5.52	2.38	5.17	2.4	6.99
150	2.42	5.23	2.4	5.33	2.46	7.2
175	2.45	5.06	2.32	5.08	2.31	7.14
200	2.42	6.18	2.41	5.7	2.38	7.12
225	2.44	6.1	2.31	6.25	2.41	6.96
250	2.3	5.75	2.36	6.01	2.31	7.61
275	2.3	7.33	2.31	6.66	2.25	7.73
300	2.38	9.19	2.5	7.68	2.27	7.75
325	2.35	9.31	2.39	7.57	2.31	7.94
350	2.32	8.55	2.24	7.38	2.45	8.42
375	2.39	8.0	2.33	6.73	2.28	8.29
400	2.42	7.15	2.34	6.92	2.32	8.71
425	2.35	6.64	2.47	7.13	2.46	8.94
450	2.29	6.41	2.34	7.07	2.4	8.5
475	2.37	6.44	2.33	7.33	2.28	8.75
500	2.51	6.36	2.38	7.8	2.42	8.69
525	2.32	7.08	2.38	8.23	2.37	9.32
550	2.45	7.58	2.42	8.07	2.38	9.2
575	2.46	8.47	2.31	9.13	2.38	9.77
600	2.34	9.25	2.47	10.04	2.37	9.79
625	2.48	10.42	2.44	10.56	2.38	10.02

Table 2: Comparison of maximum gap values for different batch sizes and beta values with $n = m$ and $n = m^2$.

The exact same process has been used to generate Table 3, now for different values of d for the d-scheme. As we can see, the more decisions we make without updating the values, the worse the performance becomes. However, another interesting observation is that the worst values are not seen in the largest batch sizes, but rather in the middle of the table.

This can be explained by the fact that we are making only 2 choices. The first choice allocates the balls randomly. If we are lucky, the allocation will follow something close to a uniform distribution across all bins. However, during the second allocation, the algorithm prioritizes placing the balls into the least-filled bins. Since we still need to add around 300 balls, all of them will be allocated to a small subset of bins that were less populated in the first allocation.

Following this reasoning, increasing the number d of choices amplifies the chances of selecting the **lowest** bin from all available bins, which leads to placing all the balls there. This explains why, for

batch sizes between 250 and 450, particularly around batch sizes 300-325, we see worse values, with gap values reaching as high as 42.14 in batch size 325 with $d = 5$.

Batch Size	**d = 2**		**d = 3**		**d = 4**		**d = 5**	
	$n = m$	$n = m^2$	$n = m$	$n = m^2$	$n = m$	$n = m^2$	$n = m$	$n = m^2$
25	2.36	2.43	2.27	2.53	2.49	3.08	2.42	3.51
50	2.45	2.82	2.32	3.39	2.38	4.41	2.32	5.59
75	2.52	3.46	2.38	4.81	2.48	6.45	2.35	8.21
100	2.4	4.22	2.41	6.44	2.27	8.46	2.39	10.85
125	2.34	5.51	2.43	8.56	2.29	11.19	2.23	13.35
150	2.41	5.48	2.29	8.63	2.39	11.42	2.35	15.57
175	2.36	4.87	2.45	7.86	2.28	12.07	2.37	17.32
200	2.41	7.02	2.48	11.79	2.29	17.19	2.43	22.81
225	2.28	6.59	2.46	11.49	2.3	17.81	2.49	25.27
250	2.3	6.01	2.42	12.88	2.33	20.98	2.38	29.59
275	2.46	8.16	2.43	16.67	2.3	25.47	2.31	35.9
300	2.3	10.95	2.3	20.13	2.49	30.72	2.37	41.25
325	2.28	12.27	2.35	20.42	2.31	31.0	2.38	42.14
350	2.39	10.45	2.4	19.27	2.42	28.17	2.47	39.16
375	2.34	9.29	2.37	17.43	2.36	24.45	2.36	33.09
400	2.33	8.37	2.28	14.14	2.43	21.41	2.37	29.76
425	2.35	7.28	2.36	12.67	2.23	18.7	2.37	25.26
450	2.43	6.51	2.39	10.62	2.31	15.93	2.34	22.4
475	2.37	5.98	2.27	8.63	2.33	12.55	2.21	16.77
500	2.34	5.83	2.37	7.02	2.33	10.39	2.41	13.01
525	2.38	6.15	2.49	6.27	2.41	7.89	2.33	10.12
550	2.36	6.76	2.39	6.52	2.36	6.84	2.35	7.55
575	2.31	7.89	2.53	7.69	2.33	8.05	2.47	7.82
600	2.46	9.43	2.38	9.06	2.37	9.64	2.47	8.74
625	2.28	10.07	2.46	10.59	2.47	9.89	2.39	10.33

Table 3: Comparison of maximum gap values for different batch sizes and d values with $n = m$ and $n = m^2$.

2.3 Another Way of Adding Uncertainty: Partial Information

In the final part of the experiment, we will introduce another method for adding uncertainty to the allocation process. Unlike the previous approach, we will have all the bins updated without delay but, but instead of choosing in which bin to allocate the ball based on the exact number of balls already present in the bins, we will only be able to ask one or two questions (depending on the choice of k-value, either 1 or 2, respectively). The questions will be:

1. Is the number of balls in this bin higher or lower than the median of all bins? If only one question is allowed, we will select randomly from the bins that have fewer balls than the median.
2. Is the number of balls in this bin higher or lower than the first or third quartile (depending on the outcome of the previous question)?

Algorithm 4 shows the pseudocode for selecting a bin according to these rules.

To start, we will use the `QuickSelect` algorithm to find the median, first quartile (Q1), and third quartile (Q3) in$mathcal{O}(n)$. Then, we will call the function `FilterAndReturn`, which updates the provided vector of bins by removing any bins that have a load *equal to or greater* than the given filter value. Only the bins that have fewer balls than the filter value will remain in the vector. If no bins with fewer balls than the filter value are found, the vector remains unchanged. The function returns

`true` if it found any bins below the filter value, or `false` if all bins have loads greater than or equal to the filter value.

If only one question is allowed, we will return one randomly selected bin from the filtered set of bins. If two questions are allowed, we will first filter based on the quartile values and then return one bin selected randomly from the further filtered set.

Algorithm 4: Choose Bin with Uncertainty

Input : Set of chosen bins, *chosen_bins*
Output : Selected bin with uncertainty handling

```
// Convert chosen bins to a vector for the FilterAndReturn function
candidate_bins ← Convert chosen_bins to a vector;
// Calculate percentiles using QuickSelect
total_bins ← Size of all_loads;
median_load ← quickSelect(all_loads, 0, total_bins- 1, total_bins/ 2);
q1_load ← quickSelect(all_loads, 0, total_bins- 1, total_bins/ 4);
q3_load ← quickSelect(all_loads, 0, total_bins- 1, (3 * total_bins) / 4);
// First filter based on median load
is_below_median ← FilterAndReturn(candidate_bins, median_load);
if is_below_median and size of candidate_bins is 1 then
    return the only element in candidate_bins;
end
if k_choices = 1 then
    return SelectRandom(candidate_bins);
end
// Second filter based on quartiles
if is_below_median then
    FilterAndReturn(candidate_bins, q1_load);
end
else
    FilterAndReturn(candidate_bins, q3_load);
end
return SelectRandom(candidate_bins);
```

As we did in the first experiment, Figures 4 and 5 show the evolution of the average Gap over 100 trials for different ball counts, ranging from $n = 0$ to $n = m^2$, with 25 bins under the d-choice scheme for $d = 2$. The first figure represents the case with one available question ($K = 1$), while the second shows the results with two questions ($K = 2$).

As observed, using two questions significantly reduces the Gap, nearly halving it when $n = m^2$. Comparing these results with those from the first experiment (without uncertainty), we get the following values for $n = m$ in order of no uncertainty, $K = 2$, $K = 1$: 1.13, 1.83, 2.21 and for $n = m^2$: 1.42, 2.36, 4.44. Then as expected, the introduction of uncertainty leads to worst results.

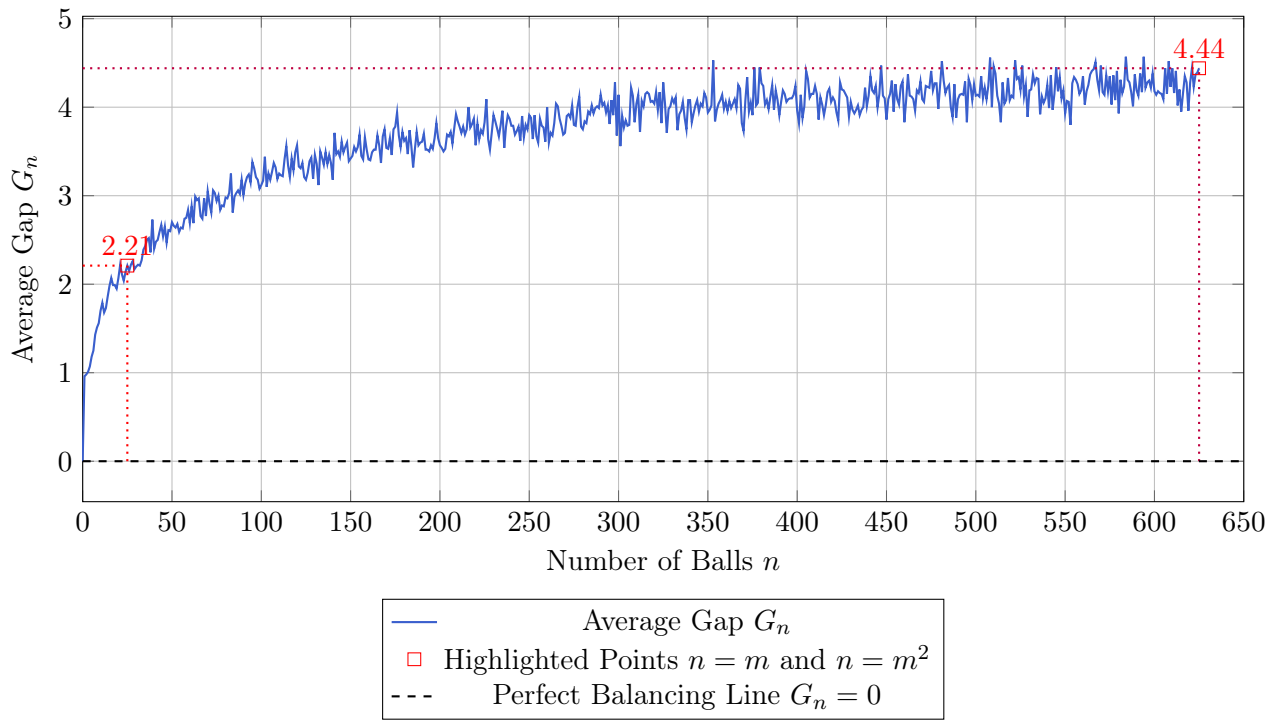

Figure 4: Average Gap G_n for the 2-Choice Scheme Modelling Uncertainty with One Question (K=1)

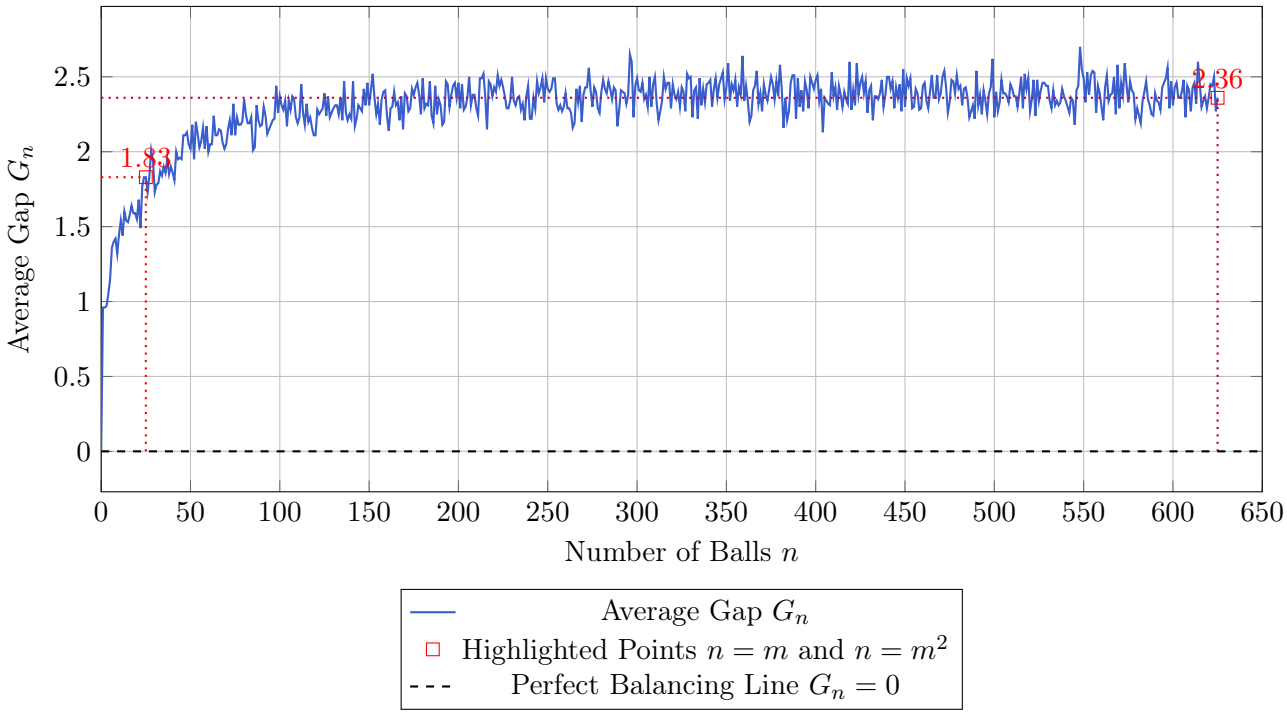

Figure 5: Average Gap G_n for the 2-Choice Scheme Modelling Uncertainty with Two Question (K=2)

Completing the table we provided in the first experiment (Table 1), Table 4 shows the side by side comparison of $K = 0$ (no uncertainty), $K = 2$ and $K = 1$ for more d Values.

d Value	**K = 0**		**K = 2**		**K = 1**	
	$n = m$	$n = m^2$	$n = m$	$n = m^2$	$n = m$	$n = m^2$
3	1.0	1.01	1.47	1.67	2.0	3.19
4	1.0	1.0	1.35	1.41	1.91	2.95
5	1.0	1.0	1.25	1.36	1.75	2.66
6	1.0	0.98	1.2	1.29	1.76	2.51
7	0.98	0.96	1.18	1.27	1.82	2.41
8	0.92	0.91	1.13	1.11	1.79	2.61
9	0.87	0.87	1.13	1.08	1.85	2.27
10	0.78	0.81	0.97	1.13	1.9	2.32

Table 4: Comparison of maximum gap values for different d values for different levels of uncertainty at $n = m$ and $n = m^2$.

To end our experimentation we provide Figure 6 which shows the evolution of the average Gap for all the number of balls from $n = 0$ to $n = m^2$ again for the different β values from $0.2, 0.4, 0.6, 0.8$ for both $K = 1$ and $K = 2$. As expected $K = 1$ performs worse making $\beta = 0.4, K = 2$ performing better than $\beta = 0.2, K = 1$ altough having a lower β value.

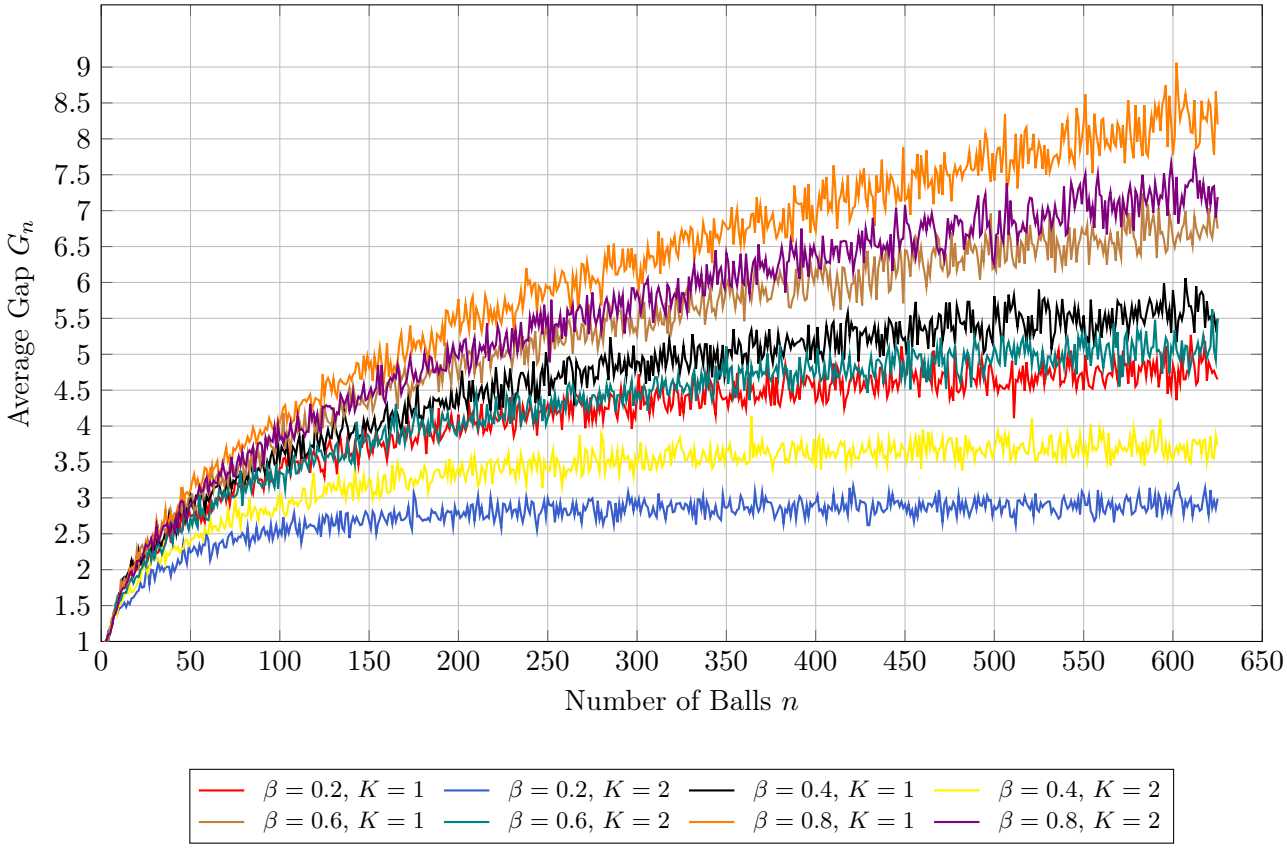

Figure 6: Average Gap G_n for 100 Trials and 25 Bins for different β and K values

3 Used Tools

All the plots have been created using TikZ+PGF. The data has been generated from the C++ code and, for some tables, the python notebook `data.ipynb` has been used to parse the csv.

References

[1] *Mersenne Twister - Wikipedia.* [Online]. Available: `https://en.wikipedia.org/wiki/Mersenne_Twister`.