Randomized Algorithms (RA-MIRI): Assignment #2

Izan Beltran Ferreiro

November 2024

Contents

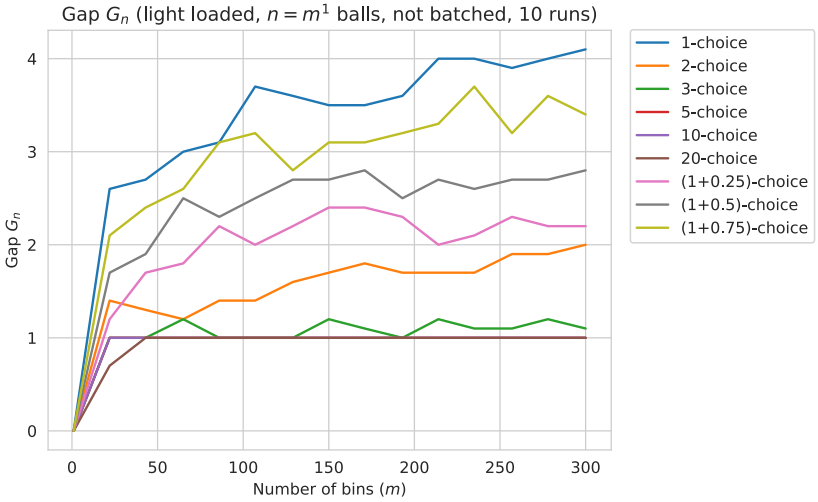
1 Introduction

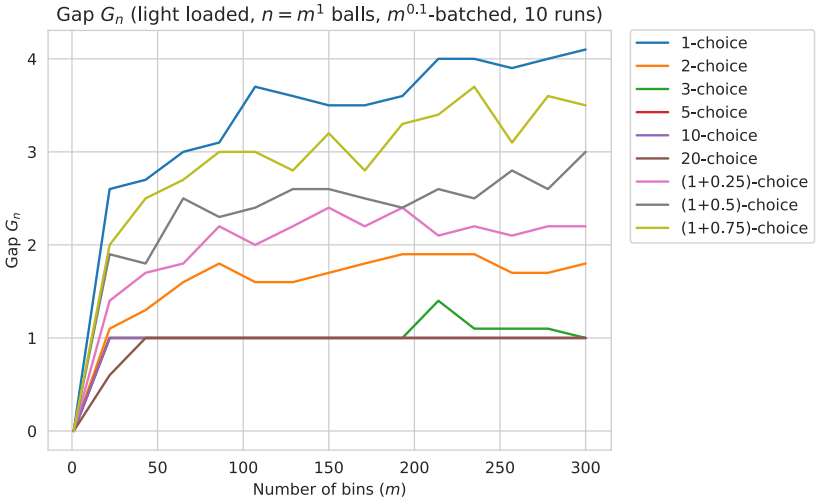
All the source code and additional files can be found on the following GitHub link: github.com/izanbf1803/balancedalloc

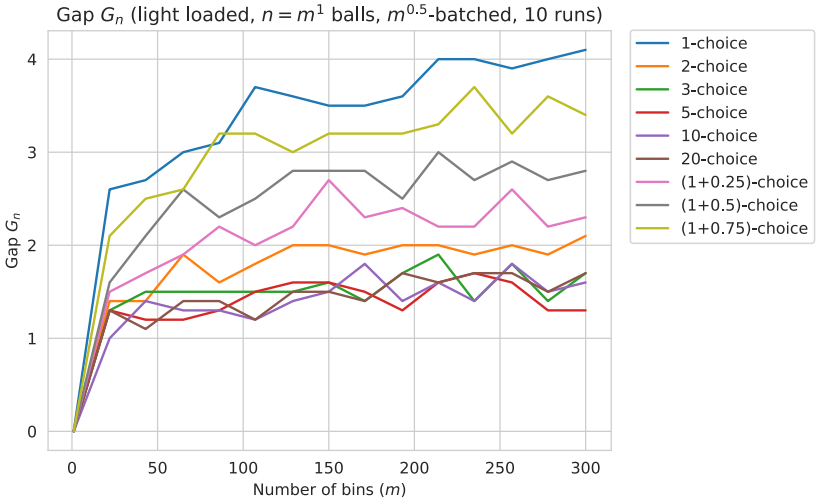
We present our results in 3 scenarios: light loaded ($n \sim m$), medium loaded ($n \sim m^{1.5}$) and heavy loaded ($n \sim m^2$). We compare not batched with different batch sizes ($m^{0.1}, m^{0.5}$ and $m^{0.9}$) and plot everything with multiple choice strategies ($1, 2, 3, 5, 10, 1.25, 1.5$ and 1.75).

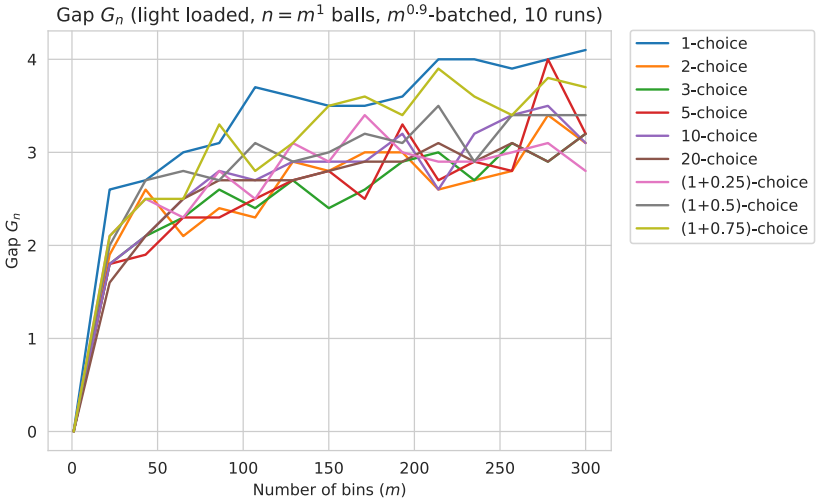
2 Results

2.1 Light loaded scenario





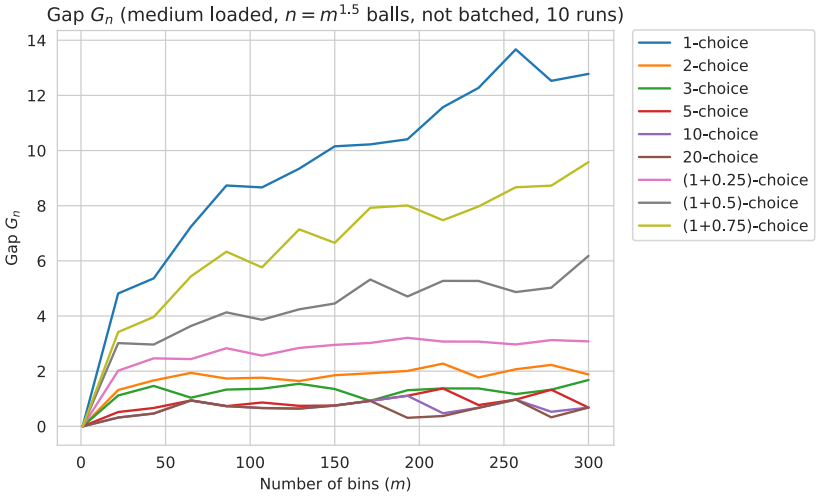


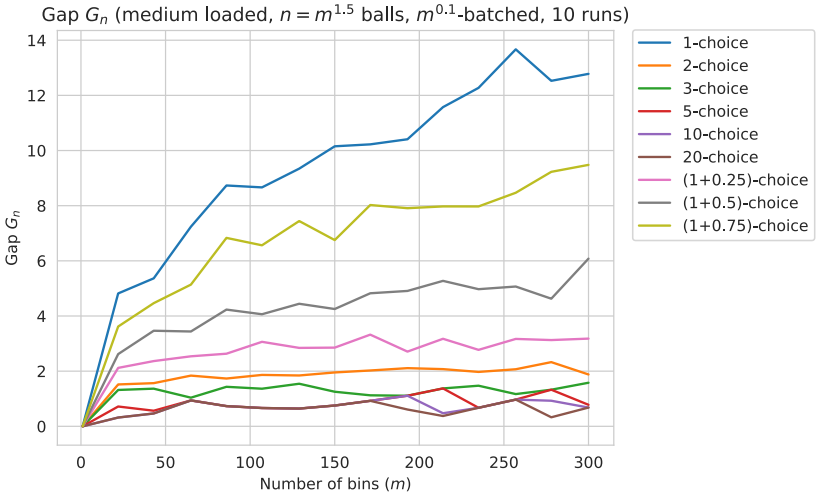


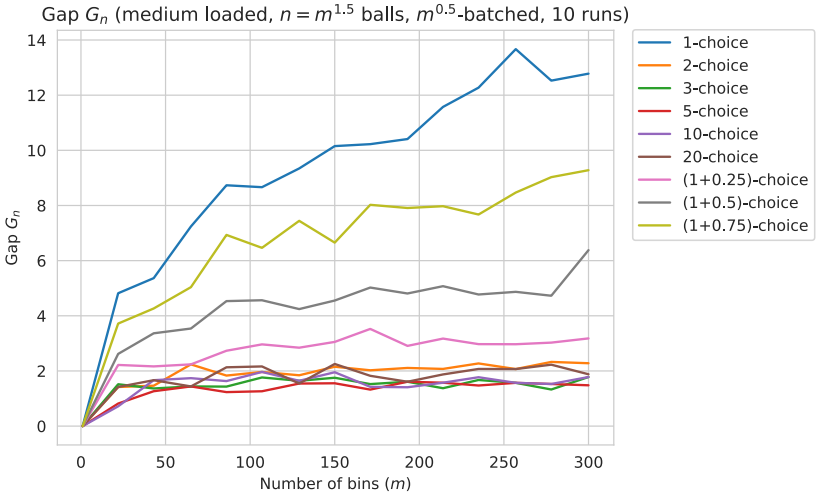
For the non-batched case we observe that d-choices are better as d increases. This is logical since it is choosing from more bins and always taking the smallest one. We observe that small batches (of size $\sim m^{0.1}$) do not degrade performance, while the bigger batch sizes start to degrade the performance, slowly making all methods converge to a similar value (having no info on the bins make all methods useless, this is why really big batch sizes make performance similar for all strategies).

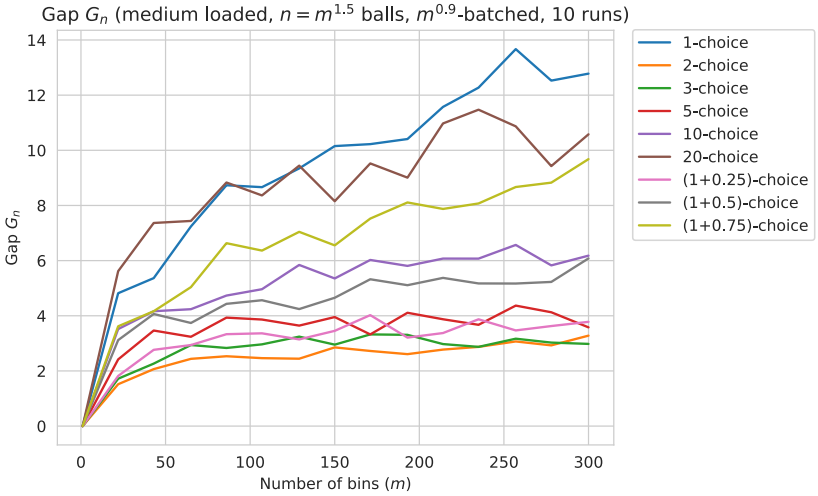
Finally, we observe that in general, the gap G_n grows in a seemingly logarithmic fashion with respect to m for this light loaded situation.

2.2 Medium loaded scenario





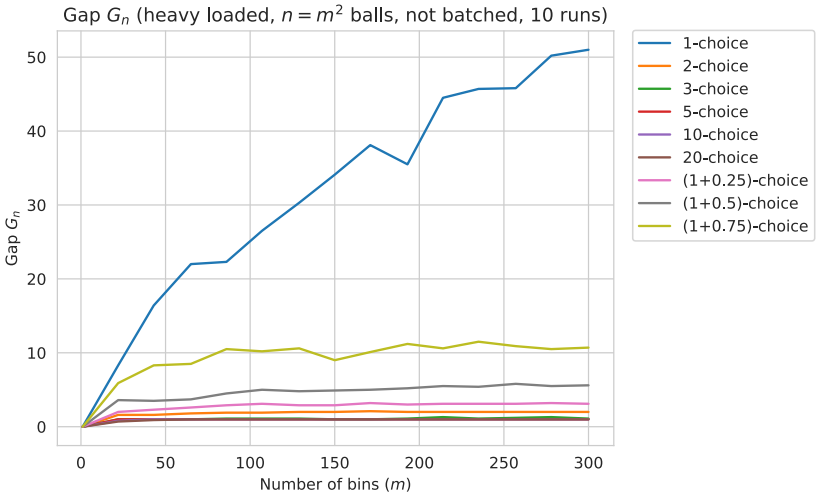


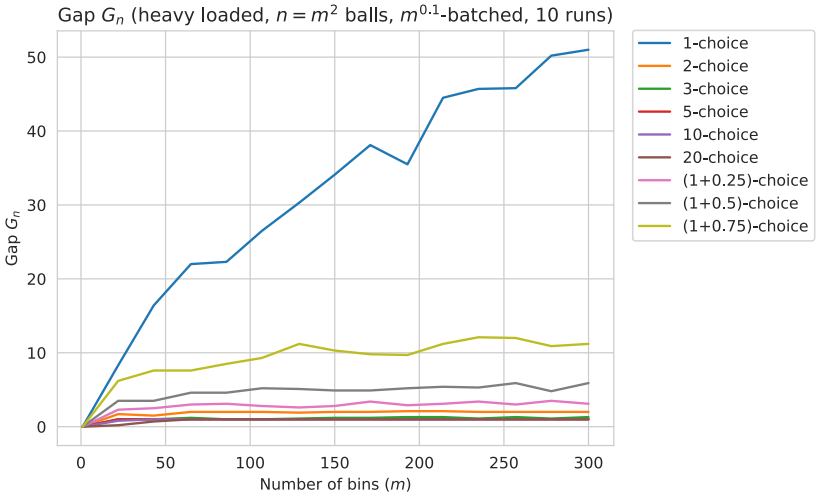


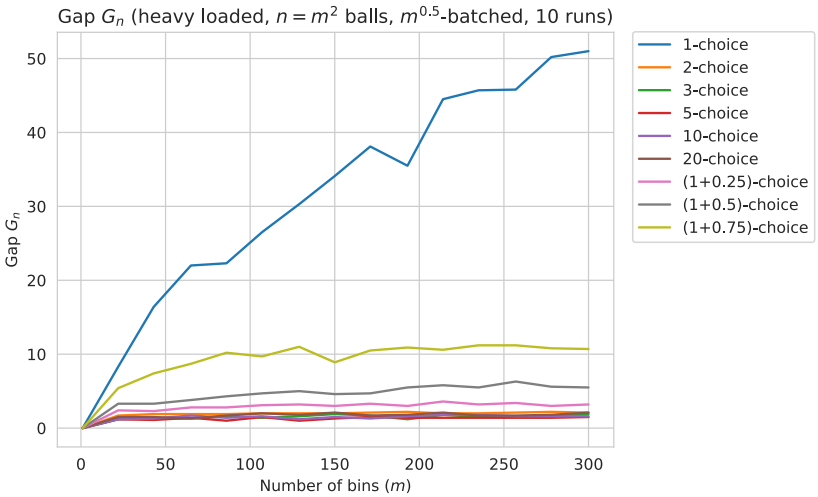
For the batch sizes, we have similar observation as in the light loaded scenario, but one special new situation: for the last batch size (of size $\sim m^{0.9}$), we see that 20-choice degraded too much, now being as bad as 1-choice. This can be explained because now we have many more balls than bins, so having outdated information about bins loads makes the algorithm take for many balls some spots that previously were the most empty, but these bins rapidly get an average load, and our algorithm has outdated information so it keeps moving balls to those positions, generating an overload situation.

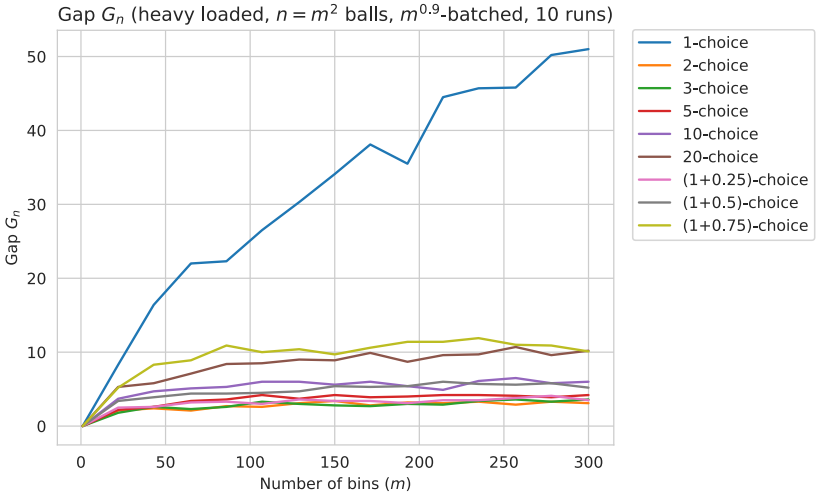
Finally, we observe that we have a similar logarithmic growth for the gap G_n in this medium loaded scenario for most strategies, except from the d-choice strategies with $d < 2$, where we observe that growth increased, at least to something faster than the logarithm. 1-choice exhibits a more square root like growth.

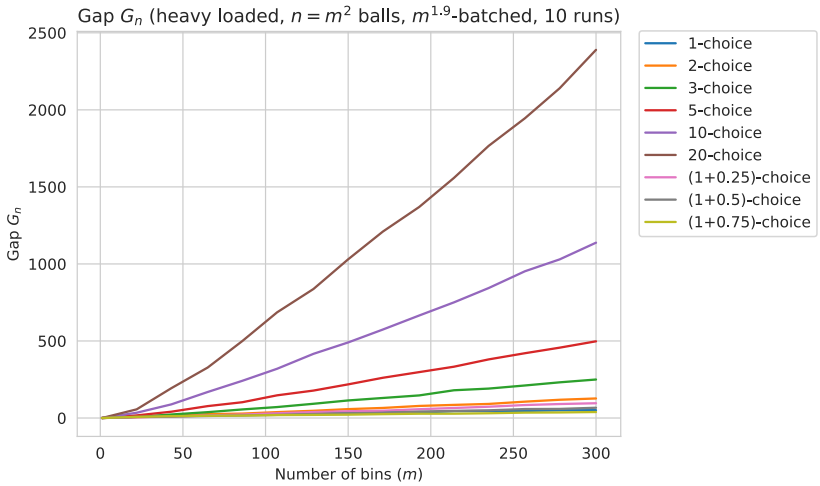
2.3 Heavy loaded scenario






Gap G_n (heavy loaded, $n = m^2$ balls, $m^{0.5}$-batched, 10 runs)
Gap G_n
Number of bins (m)
0
10
20
30
40
50
0
50
100
150
200
250
300
1-choice
2-choice
3-choice
5-choice
10-choice
20-choice
(1+0.25)-choice
(1+0.5)-choice
(1+0.75)-choice


Gap G_n (heavy loaded, $n = m^2$ balls, $m^{0.9}$-batched, 10 runs)
Gap G_n
Number of bins (m)
0
10
20
30
40
50
0
50
100
150
200
250
300
1-choice
2-choice
3-choice
5-choice
10-choice
20-choice
(1+0.25)-choice
(1+0.5)-choice
(1+0.75)-choice



For the batch sizes, we have similar observation as in the light loaded scenario, but one special new situation: for the last batch size (of size $\sim m^{1.9}$), we see that 20-choice degraded extremely, to the point of the gap starting to increase faster than in any previous situation. This can be explained simarly to the situation in the medium loaded scenario, but now the ratio balls to bins is much higher so the effect is much stronger, to the point where the gap grows super-linearly for all strategies.

Finally, we observe that in general the growth is logarithmic for the d-choice with $d \geq 2$ in the non-batched case.