UNIVERSITAT POLITÈCNICA DE CATALUNYA

RANDOMIZED ALGORITHMS

LAB. 2: **BALANCED ALLOCATIONS**

STUDENT: BRUZZESE LUIGI
DATE: 10/11/2024

GITHUB FOLDER WITH SOURCES: https://github.com/luigibruzzese/UPC-RA-MIRI-Lab.git

INTRODUCTION

This report is part of the deliveries for the second lab assignment of the course Randomized Algorithms held at Universitat Politècnica de Catalunya in Barcelona (Spain).
In the shared GitHub folder there are the source python file used for showing the results explained later and a README.md file that explains how to run it to obtain the same results.

The assignment asks to make a series of experiments simulating the so-called “balls and bins” problem, i.e., a scenario in which there are **n balls** that are thrown into **m bins** with some fixed strategy that can involve and mix randomized and deterministic approaches. The aim is to compare different strategies, based on the **maximum gap** between the most loaded bin and the average load (which is $\frac{n}{m}$), and understand their mathematical properties to find which ones are better.
Several parameters will be used; most of them are based on the different strategies and will be explained later, but some of them are general and independent from the chosen one:

- $\boldsymbol{n}$ and $\boldsymbol{m}$, as already explained: respectively, the number of balls and bins;
- **T**: each configuration of (n, m) will be thrown for **T** rounds, in order to have, in the end, not just the gap for a single run but an average gap on these rounds;
- **analysisStep**: most of the time, instead of fixing n and m and run only T rounds, we will fix only m, and the script will run all the configurations from (n, m) with $\boldsymbol{n = m}$ (we will call this the **light-loaded scenario**) to $(\boldsymbol{m^2}, m)$ (the **heavy-loaded scenario**), with the specified *analysisStep* (so that, from (n, m), the next configurations are $(n + step, m)$, $(n + 2 * step, m)$ and so on so forth). In this case, we just specify $\boldsymbol{n}$**,** and m is implied. After the computation, our script prints in a plot all the average maximum gaps we’ve obtained for each value of n, to better show the results.

The following paragraphs are related to the different strategies we analyzed and try to explain the main results obtained and the comparisons among them.

D-CHOICE

Following the convention of the assignment, we refer to d as the parameter for the so-called $d-choice$ strategy, which consists of, for each ball, selecting randomly $\boldsymbol{d}$ **bins** and then putting the ball into the one among them **with the smallest number of balls**.
In that way, we will talk about the **standard allocation** scheme here, since it consists of just putting each ball into a random bin and, so, becomes the naïve approach of the d-choice strategy in which $d = 1$.

With a simple case of $n = 100$, performing the analysis from the light-loaded through the heavy-loaded scenario with a $step = 50$, we can compare the results for different values of d:

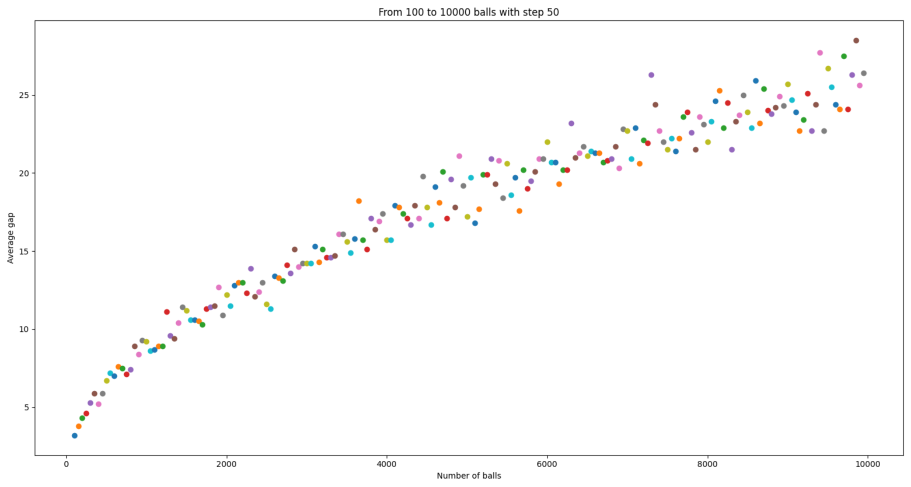
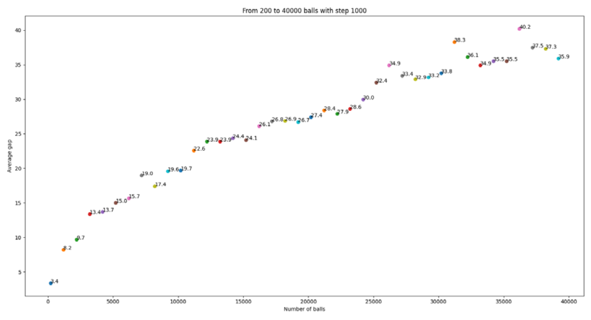
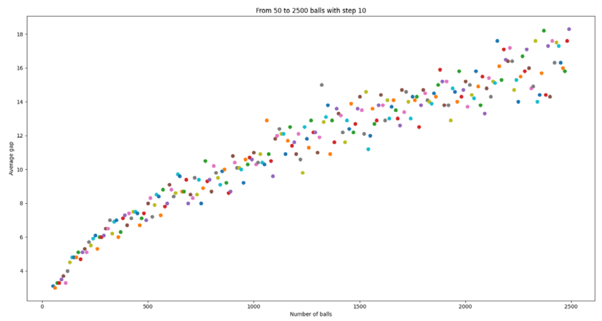


Figure 1: *d = 1 (random allocation), m = 100, from n = m to n =* m^2 *with step = 50*

As visible, with $\boldsymbol{d\ =\ 1}$ we have a kind of "linear" increasing of the gap as we increase the number of balls. The first point on the left is the light-loaded scenario, while the last one on the right is the heavy-loaded. This result about the standard allocation is general even changing the values of *n (*and *m)*:





Figures 2-3: *d = 1, m = 200 (on the left) and 50 (on the right), from n = m to n = m^2*

If we switch to $d = 2$, using the same configurations as above, we obtain a very different result:

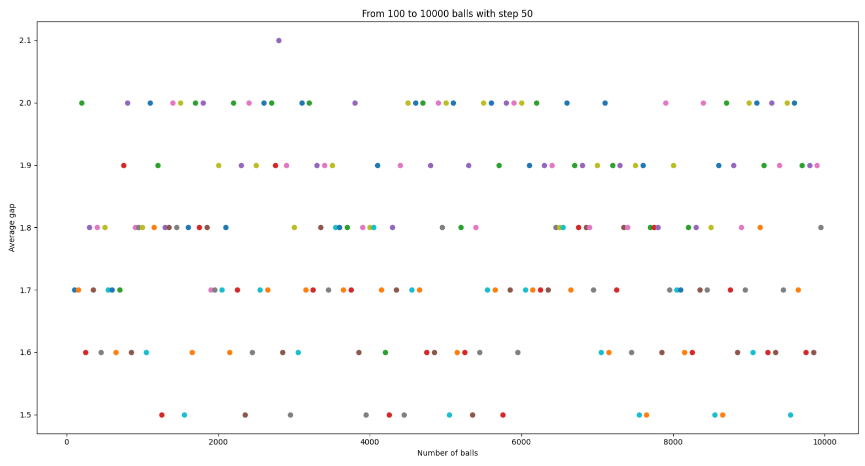
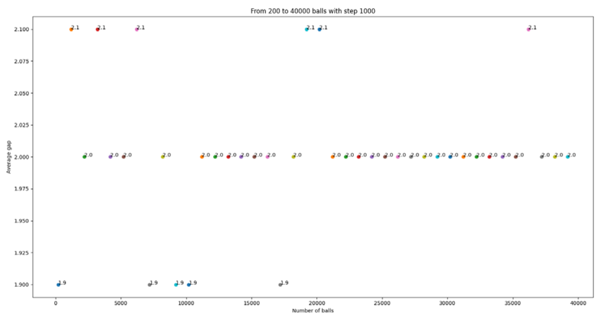
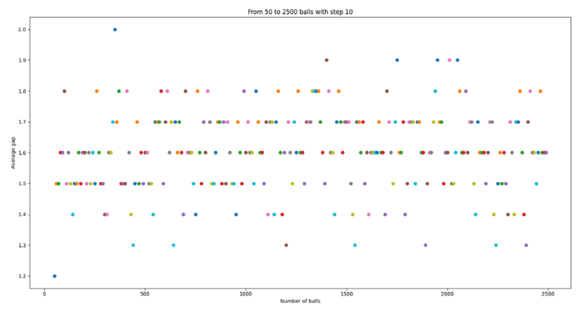


Figure 4: *d = 2, m = 100 with step = 50*

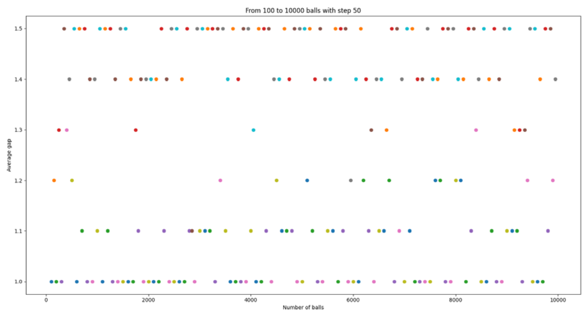


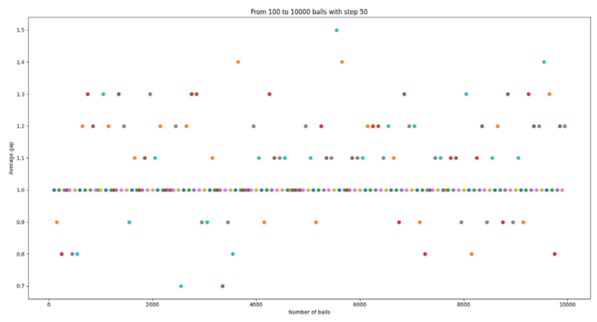


Figures 5-6: *d = 2, m = 200 (on the left) and 50 (on the right)*

The improvement is very significant: the maximum gap that we reach is reduced a lot by each run of roughly 10 times (from *29, 40* and *18* for $d = 1$ to *2.1* for $d = 2$ in all the cases). Moreover, we can see that we don't have an increase of the average gap going toward big values of *n* (i.e., to the heavy-loaded scenario), but the values remain more or less the same with a very small gap: the difference between the light and the heavy-loaded scenarios is ≤ 1 (with respect to values between *20* and *35* for $d = 1$).

At this point, we expect that increasing the value of d leads to very significant improvements. Unfortunately, that's not the case: the results are better, but the improvement is very small with respect to what we've seen between $d = 1$ and $d = 2$:



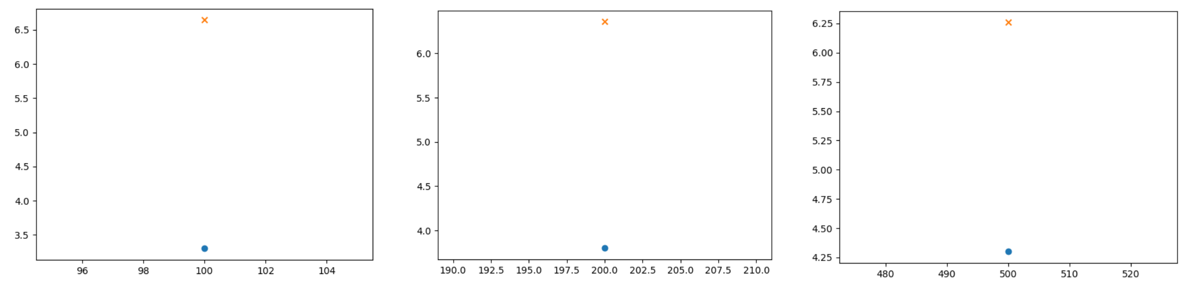


Figures 7-8: *m = 100, d = 3 (on the left) and d = 4 (on the right).*

From the previous maximum gap of *2.1* with $d = 2$, in this case of $m = 100$ we reach *1.5* both for $d = 3$ and $d = 4$.

If we don't have so many computational resources, there's no reason to increase the value of *d* with values greater than 2, since the improvement will be very small. On the other hand, switching from $d = 1$ to $d = 2$ is very convenient, especially when we go through the heavy-loaded scenario, since the gap will not increase even if the number of balls does, and that could be important to maintain a good allocation scheme without taking care of how many balls we throw.

Notice that, as the theoretical results suggest, with $d = 1$ (i.e., random allocation) the maximum gap is $\theta\left(\frac{logn}{loglogn}\right)$ ([Gon81]). Indeed, if we set a cross to this value, we see, without performing the analysis but just with $n = m$:


Figures 9-10-11: *m = 100 (on the left), 200 (center), 500 (right) with n = m. The cross is the [Gon81] bound.*

Furthermore, this bound goes through $\vartheta\left(\sqrt{\left(\frac{nlogm}{m}\right)}\right)$ with high probability if $n > mlogm$ so $n >> m$ [RS98]. Indeed, if we show it with respect to the different values of n with the $analysisStep$:

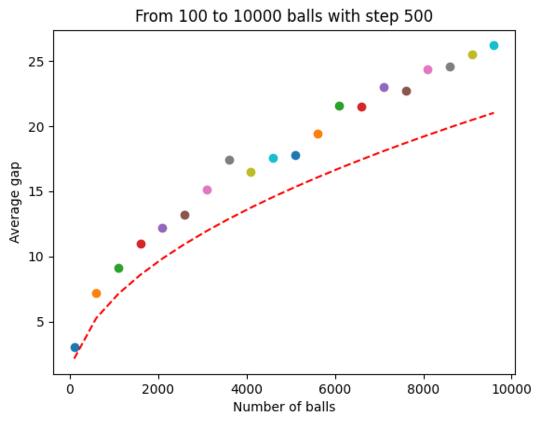


Figure 12: *d = 1, m = 100, n from m to m^2 with step = 500 (the points); [RS98] bound (the red dotted line)*

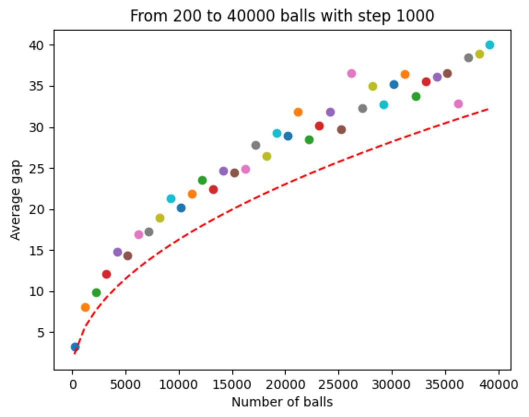


Figure 13: *d = 1, m = 200, n from m to m^2 with step = 1000 (the points); [RS98] bound (the red dotted line)*

On the other hand, with $d > 1$, the maximum load becomes $\vartheta\left(\frac{loglogm}{logd}\right) + \vartheta(1)$:

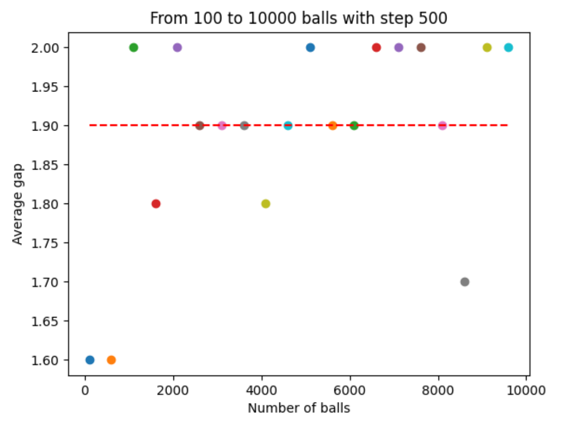
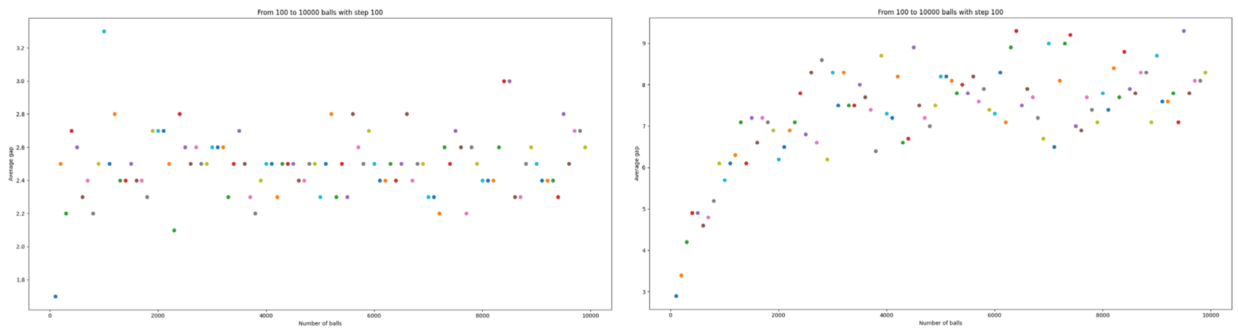


Figure 14: *d = 2, m = 100, n from m to m^2 with step = 1000 (the points); bound (the red dotted line)*

and this result explains how leading to $d > 2$ gives only very small improvements (only the constant at the denominator will change) with respect to switching from $logm$ $(d = 1)$ to $loglogm$ $(d = 2)$.

(1+β)-Choice

A slight variant of the previous strategies is the **(1+β)-choice**: in this case, each ball is allocated using **one-choice** with **probability β** and **two-choice** in the other case (**probability 1-β**).
Following what we showed previously, it's intuitive that **small values of β** leads to better results than higher ones (since, with high β, we're going toward the 1-choice, which is worse than the 2-choice as already explained):

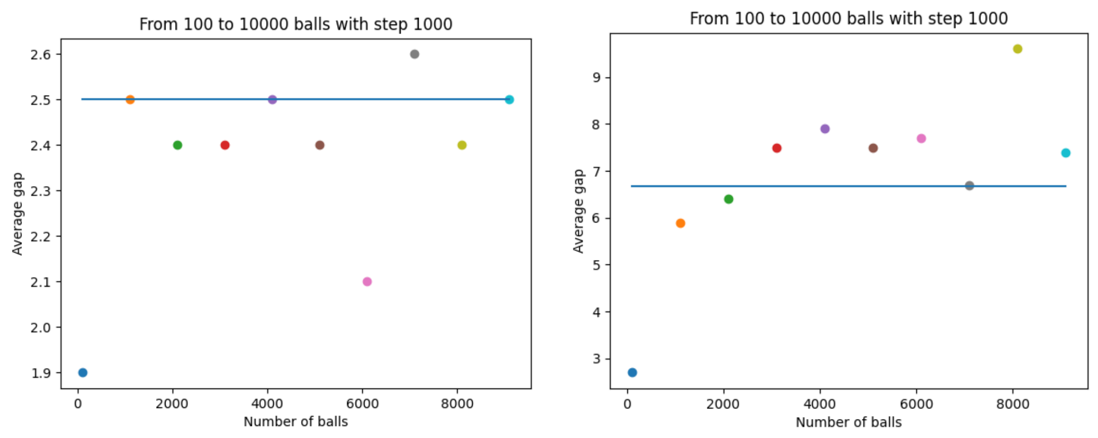


Figures 15-16: *m = 100, n from m to m^2 with step = 100, β = 0.2 (on the left) and 0.7 (on the right)*

For the smallest value of β, we reach a maximum gap of roughly *3.3*, while on the right, with the greatest one, we arrive to *9.2*.

A mathematical result about this strategy has been explained in [PTW15]. Notice that, in their convention and in most of the papers we will cite in this report, n and m are used in the opposite way, and the same for β (as the probability of 2-choice and not 1-choice): we will translate and use their results with our convention.

This paper shows that the gap in the heavy-loaded case is $\theta\left(\frac{logm}{1-\beta}\right)$, independently of n, and it becomes less accurate for greater values of β. It's, indeed, something we can prove with our experiments:



Figures 17-18: *m = 100, n from m to m^2 with step = 1000, β = 0.2 (on the left) and 0.7 (on the right)*

The blue line represents this bound: as we can see, on the left side (so with smaller values of β) it is more accurate.

At this point one can ask: why should we use this strategy since it doesn't lead to better results than a simple *2-choice* one? In general, a process like this can be motivated when the cost to query the load of the bins is high; in that case, using this strategy, the expected number of times in which we query the bins becomes (1-β)%, with respect to the 100% of $d = 2$. If we compare the results this strategy obtains with a simple $d = 1$, even in the case of $\beta > 0.5$, we can see that there's an improvement that can be a good trade-off between the simplest strategy and the expensive one with 2-choice (with the same number of bins, we reached a maximum gap in the heavy-loaded scenario of roughly 9 with $\beta = 0.7$ with respect to 27 with $d = 1$).
Furthermore, this strategy will be used in combination with others in the next sections and we will clearly see its benefits.

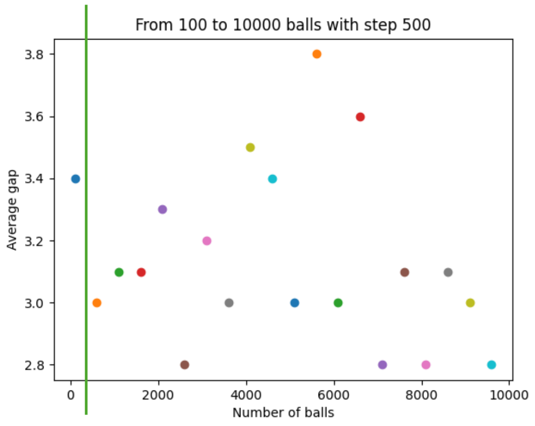
B-BATCHED

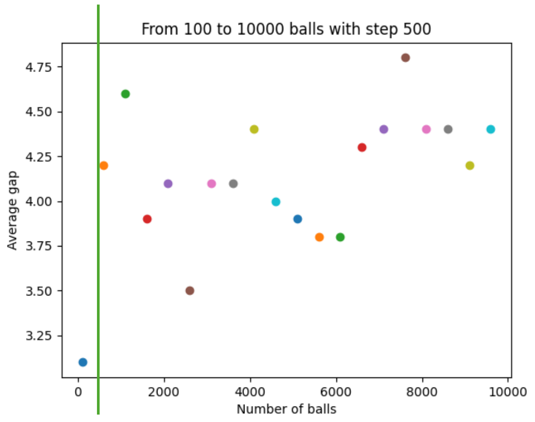
From what we've seen so far, the load of the bins is updated with the insertion of each ball thrown, so that the next ball uses the updated loads for its checks and calculations (in the case of *d-choice* with $d > 1$).
Now we switch to the so-called ***b-batched*** **strategy**, which uses a new parameter, called ***b***, to be interpreted as how many balls we throw until we upload the information about the loads of the bins. It means that, after the first ball, all the next *b-1* balls will use the same information about the loads available at the beginning of the first one, without considering where the other balls before (and in the same batch) were thrown.
This strategy can be interpreted as a kind of parallel environment, instead of the sequential settings used so far, in which we can throw all the balls in a **window of size *b* in parallel**, since they will all use the same information about the loads of the bins of the beginning. This strategy can be useful when we want to parallelize our process, for instance using more than one processor (if we think of it

from an IT perspective) that just update each other about the loads of the bins at the end of each batch but throwing the balls of the same batch at the same time in parallel.

Notice that the previous results can be seen as a particular (trivial) case with $b = 1$ (and that's how the python script works for simplicity), while with $b \geq n$ we obtain exactly $d = 1$, since we're using a single batch for all the balls (which will not contain any information, all the loads will be 0 and we will choose randomly). In general, using $d = 1$ with this strategy is something useless, we're not considering at all the loads of the bins: we will use $d > 2$. On the other hand, since we look at values of n from m to m^2, our values of b will follow this trend being $\geq m$, and we will set on our plots a delimiter (a green vertical line) after $b = n$, to show that before that delimiter we're just using a standard random-choice strategy ($b \geq n$), while after we have more consistent results.

[LS22] is one of the papers that analyzes this strategy from the mathematical point of view. Following their suggestions, we can see that, for **small batch sizes**, i.e., $b \in [m, m\ logm]$, the **2-choice still achieves an (asymptotically) optimal gap** with respect to other strategies. If we compare it with a (1+β)-choice with the same number of bins and batch size:

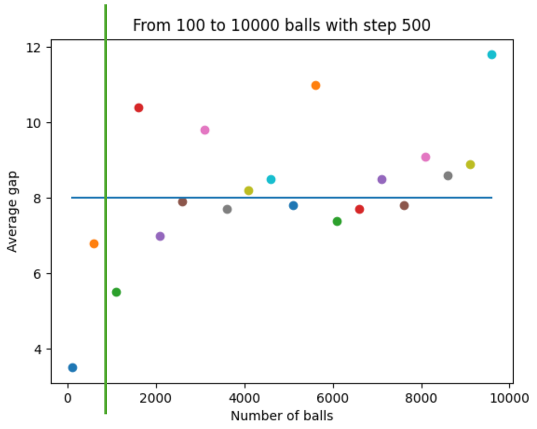


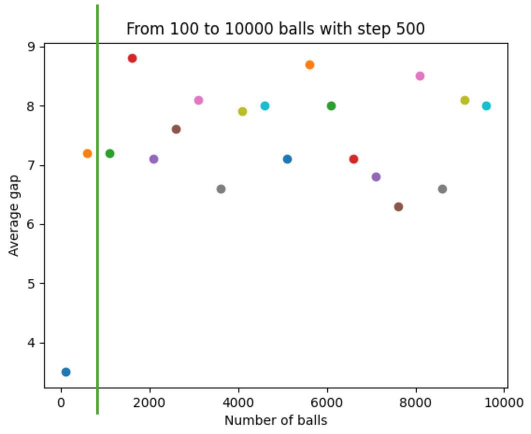


Figures 19-20: *m = 100, b-batched with b =* ***110****, d = 2 (left) vs (1+beta) with beta = 0.4 (right)*

we see that the distance is not so relevant, but the 2-choice still achieves a better result.

If we try **a larger batch size,** i.e., $b \epsilon [mlogm, m^3]$ following the suggestion of the paper:





Figures 21-22: *m = 100, b-batched with b =* ***800****, d = 2 (left) vs (1+beta) with beta = 0.4 (right)*

First, we notice that increasing *b* leads to worse results in terms of the gap (comparing figures 19-20 with 21-22): that's intuitive, since we're going toward a more parallelized environment by using more outdated information about the loads of the bins when we select a bin instead of another.
What is more interesting, however, is that in the last shown case **the (1+β) choice leads to better results** (the maximum gap is smaller!) **than the 2-choice**. This is something that [LS22C] proves: indeed, on the left figure it's represented, in blue, the $\vartheta\left(\frac{b}{m}\right)$ gap they estimate for 2-choice (and it's respected), that is roughly what we reach for the maximum value on the right. Notice that Los and Sauervald gives to their work the name of "The tower of two choices", to explain why this happens: 2-choice with b-batched allocates *more aggressively* to lightly loaded bins, leading to the formation of **"towers" of balls**, that are not good from the point of view of the gap and let it to be higher.

Furthermore, they suggest that the (1+β) strategy can lead to a gap of $\theta\left(\sqrt{b * \frac{logm}{m}}\right)$ with an appropriately chosen (1-β) = $\theta\left(\sqrt{\frac{mlogm}{b}}\right)$, for $b \in [2mlogm, m^3]$; this result proves, mathematically, why (1+β) becomes better than the $\vartheta\left(\frac{b}{m}\right)$ gap estimated for 2-choice.
Indeed, if we do an experiment with *b* = ***2000***, using $\beta = 0.7$ following their formula, we see:

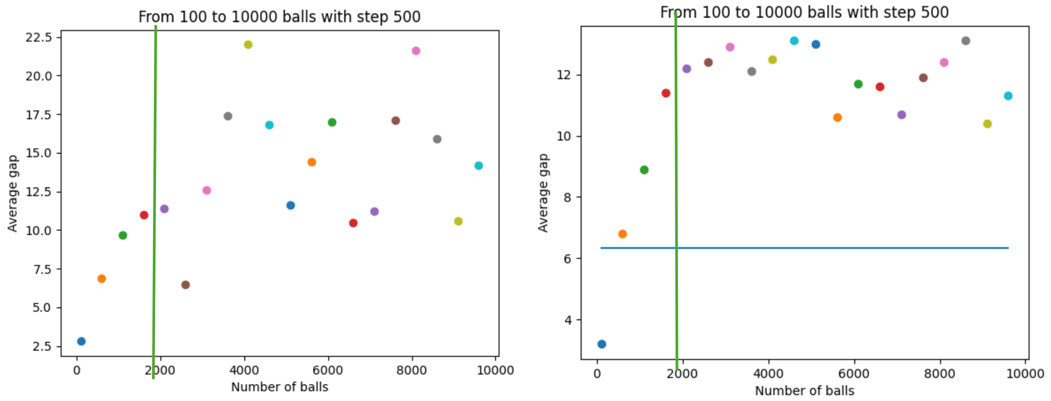


Figure 23: *m = 100, b-batched with b =* ***2000****, d = 2 (on the left) and (1+β)-choice with β = 0.7 (on the right).*
The blue line on the right is the estimated approximation of the bound from [LS22C].

Increasing b, the gap between (1+β) and $d = 2$ increases in favor of the former, and the approximation of the bound from [LS22C] is more respected:

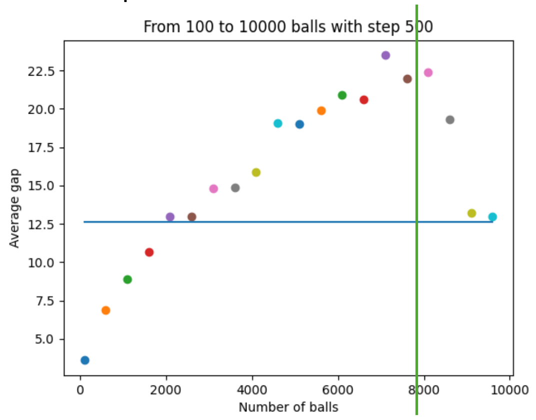
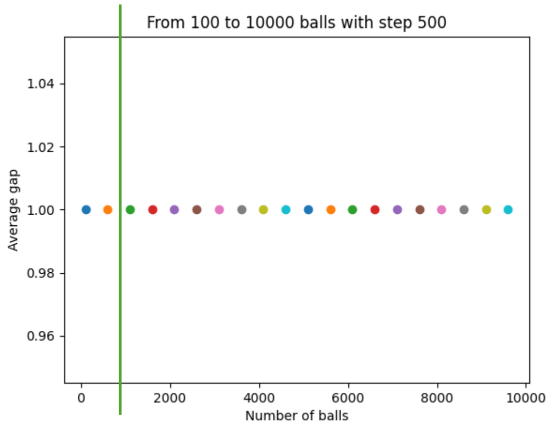
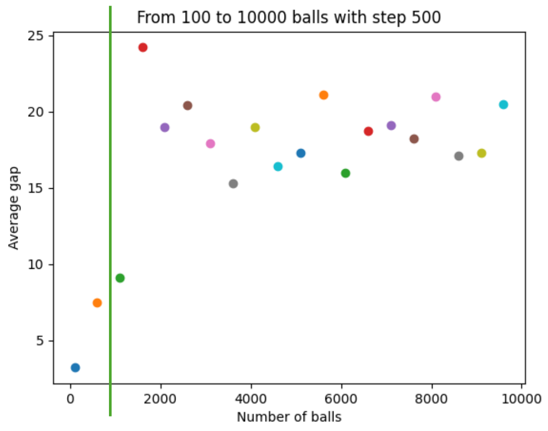


Figure 24: *m = 100, b-batched with b =* ***8000****, (1+β)-choice with β = 0.158 (using the formula).*
The blue line is the estimated approximation of the bound from [LS22C].

Now we can justify more the motivation for introducing a (1+β)-choice: it can improve the results a lot when we want to switch to a parallelized environment and not a set of sequential steps. The combination of ***b-batched* and *(1+β)-choice*** allows us to save computational resources, avoiding asking every time about the loads of the bins and parallelizing on different processors, but still having relevant improvements with respect to standard 1-choice.

What happens to the b-batched strategy if we increase *d*?





Figures 25-26*: m = 100, d = 4 without batching (on the left) and with 800-batched (on the right)*

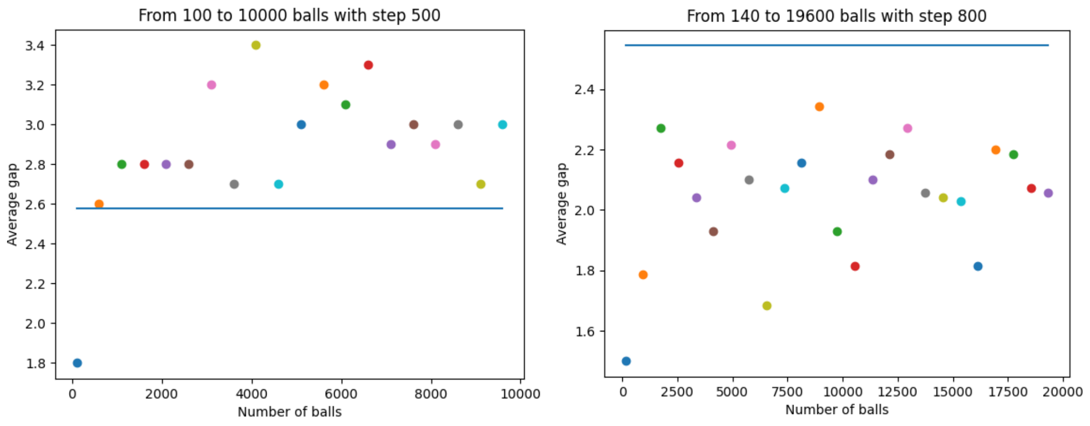
We saw that increasing d in a sequential environment leads to better results (even if it isn't worth it since we're not so far from the results of *d = 2*), and that's why, with *d = 4* we're achieving very good results in the left figure.
On the other hand, if we mix this strategy with a b-batched one (right figure), we're doing worse if we compare with the same batch size but a lower value of *d* (or, on the same level, if we compare with the (1+beta)-choice). This result is nothing more than the amplification of the previous one: with a batched strategy with a high *b*, the more *d*, the more we have the formation of towers, since the information is not updated at each ball, and we lose the relevant improvement we had by increasing *d* in a standard environment.

K-QUESTIONS

The last strategy we present in this report is the so-called **k-questions**. In that case, always working with $d > 1$, instead of looking only at the loads of the bins when selecting one of them, we can "ask" more specific questions, i.e., retrieve different information from the bins. K represents the number of questions we ask:

- ***k* = 1,** we select the bin **below the median** (and, if more than one is or neither, we just select one at random);
- ***k* = 2,** we ask the same question as *k = 1* but, if there are ties, we see which bins are among the most loaded; if there are still ties, our selection is random, otherwise we select the one which is not among the most loaded. Notice that "the most loaded" refers to two percentages we use: **75%** in the case in which, with question 1, we saw that **more than one bin is below the median**; **25%** in the case in which more than one was **above the median**.

Let’s start by seeing what happens with one query (*k = 1*) with two different values of *d*:



Figures 27-28: *k = 1, m = 100 and d = 2 (on the left) vs m = 140 and d = 3 (on the right)*

If we compare the left case with figure 4, that had the same configuration but with a pure 2-choice strategy, we see that we’re doing worse (maximum gap *3.4* vs *2.1*) but not so much. It’s interesting to notice that the light-loaded scenario has a very good performance, quite like the 2-choice; that’s not a coincidence. The blue line in the figures, indeed, represents the bound found by [FGG2021] using k = 1, which is $O\left(\sqrt{\frac{logm}{loglogm}}\right)$, and is actually a very good bound. They proved that it’s correct in the light-loaded scenario, and indeed in both cases the first point on the left (which corresponds to that) is below this bound, but with an open question on if it could be applied also to the heavy. Unfortunately, this paper shows that it doesn’t work in general, and indeed is something we can see in our experiments in the left figure (figure 27) with $d = 2$.
On the other hand, our numerical results with *k = 1* are more similar with the (1+β)-choice (see Figure 15).

Let’s switch to k = 2:

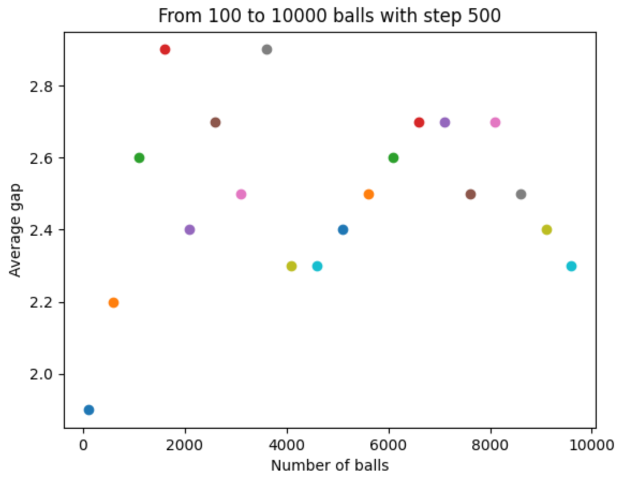
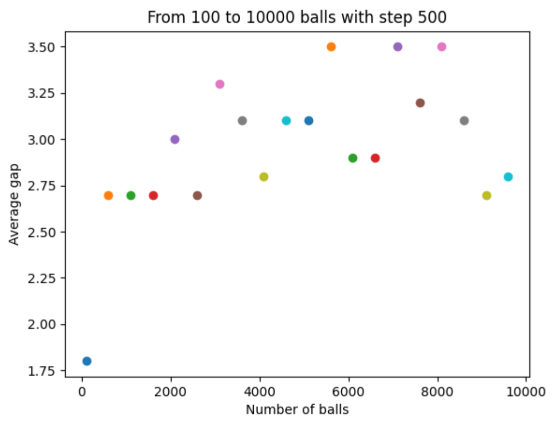
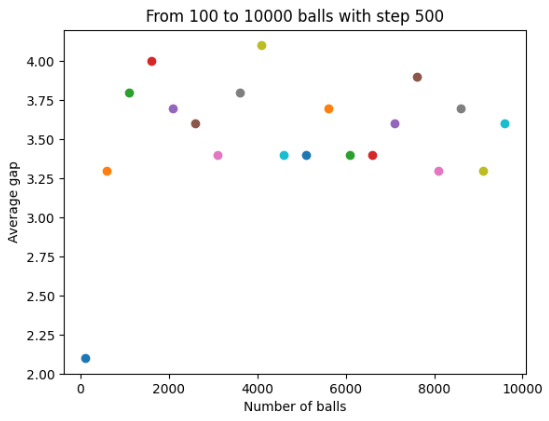


Figure 29: *m = 100, d = 2, k = 2*

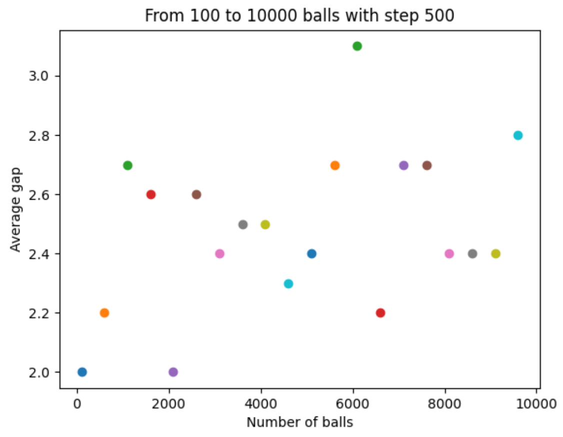
We’re improving the result of *k = 1* (figure 27) and are going away from the values of (1+β)-strategy toward the values we saw in figure 4 related to the pure 2-choice strategy. This result is general, based on the same paper we cited above: for a **large value of k**, we’re going toward **the 2-choice**, while for **k=1 resembles the (1+β)-choice**. For the latter case it’s more evident:





Figures 30-31: *m = 100,* ***k = 1*** *(on the left) vs* ***(1+β)-choice*** *on the right, with β = 0.4*

while for the former, we don’t see a perfect comparison:



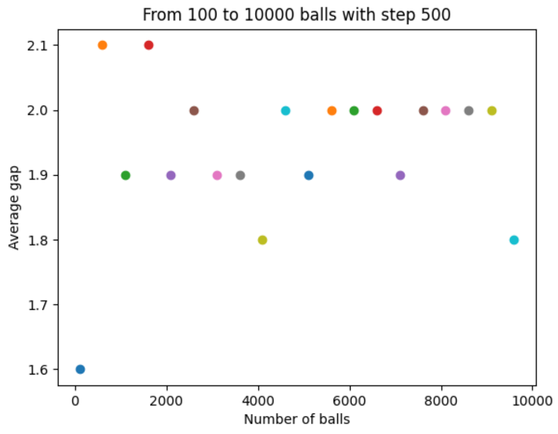
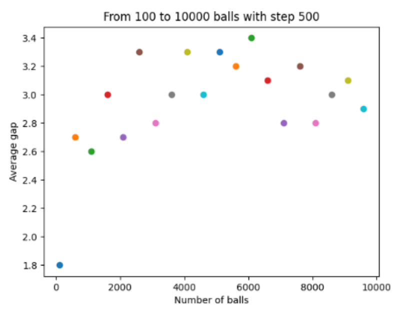
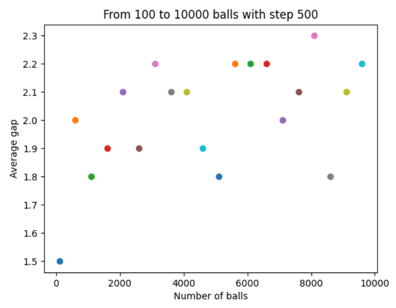


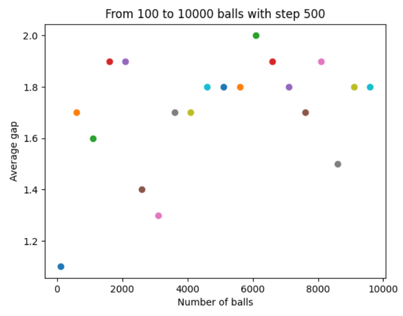
Figure 32-33: *m = 100,* ***k = 2*** *(on the left) vs* ***pure 2-choice*** *(on the right)*

and that happens because we’re not working with high values of k but stopping at k = 2: however, even with our small experiment, we see the difference and how the trend is going toward it.

Now, let’s see what happens to the k-strategy by increasing the values of d:



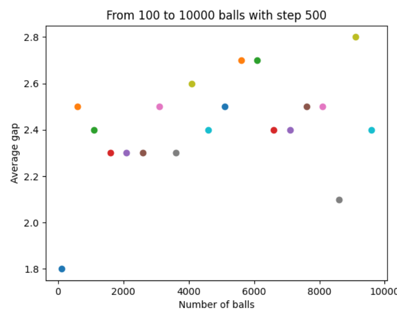


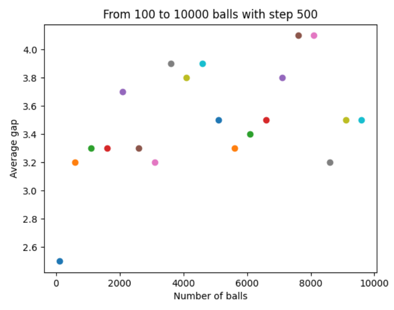


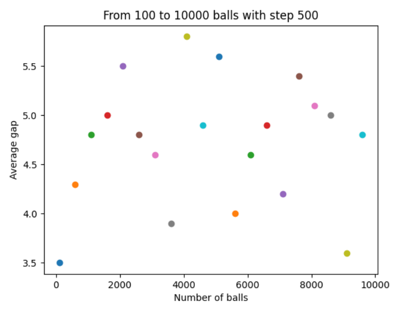
Figures 34-35-36: *k = 1, d = 2 (left), 3 (center), 4 (right)*

We're back to what happened with a pure d-choice strategy, in which increasing d led to better results: that's intuitive, since with *k = 1* we're asking a question about the loads and with more bins to choice we have more possibilities of finding one below the median. On the other hand, always intuitively, we don't have the same improvement we saw increasing *d* without *k* (just compare these last figures with figures 7-8 at the very beginning), since here, if we have ties on the question, we select randomly and not the least loaded as before.

What feels weird is going to *k = 2*:



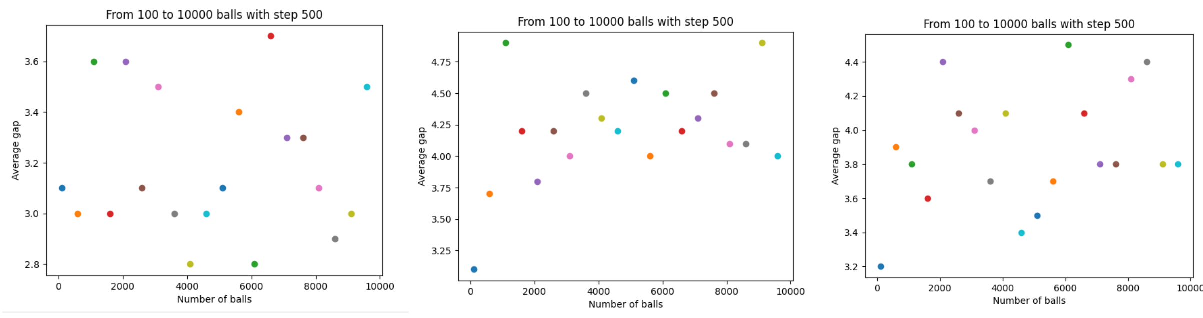




Figures 37-38-39: *k = 2, d = 2 (left), 3 (center), 4 (right)*

The behavior is the opposite (increasing *d* leads to worse results) and, furthermore, if we compare the same value of *d* but different *k* (so *figure 35* with *38* and *36* with *39*) we see that we obtain worse results increasing *k* (differently from what we saw above).

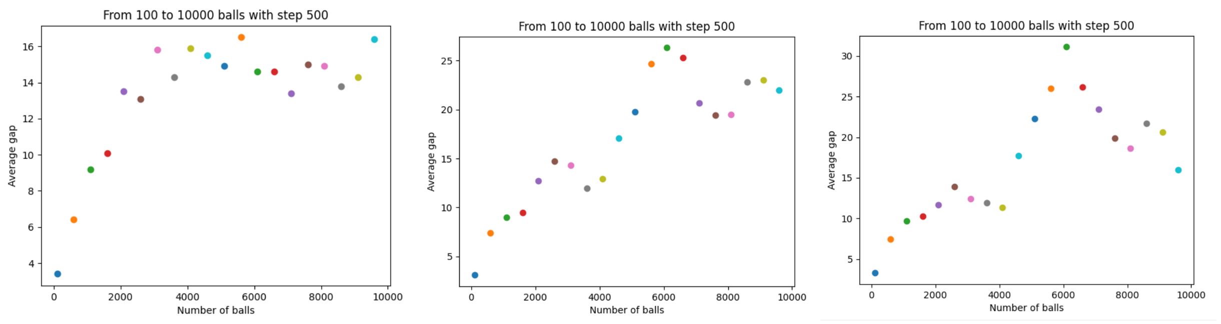
In the end, considering all the strategies proposed, we're missing a mix between the k-strategy and the b-batched one.
Starting from a small batch size, e.g., *b = 110*, and using *d = 2* since we've seen that it's the best choice in that case:



Figures 40-41-42: *m = 100, d = 2,* ***b = 110****, without k (left), with k = 1 (center) and k = 2 (right)*

we can find something expected: with small batch sizes, *k = 2* is better than *k = 1*, and that's correct since we saw that it goes toward the results of the 2-choice (the left figure), that is the best one in that configuration.

On the other hand, by increasing the value of *b* to **3000**, we saw that the best strategy became (1+β)-choice, and using *k*:



Figures 43-44-45: *(1+beta) with beta = 0.75, no k (left), k = 1 (center) and k = 2 (right)*

we confirm that *k = 1*, which goes toward (1+β), is better than *k = 2*.

In the end, from what we've obtained, the usage of *k-questions* is worse than a simple *d-choice* (or (1+beta)) strategy, but can be convenient, again, if it's less computationally expensive, since we obtain similar results by appropriately setting the parameters based on what we need (if combined with the *b-batched*, the batch size can define which *k* we have to choose based on its correlation with the *d-choice* strategy).
On the other hand, in all the experiments, we can see that the difference between k = 1 and k = 2 is not very relevant, concluding that, if the second question requires more resources than the first one, there's no reason of switching to it.

Conclusions

In an ideal environment in which we don't care about the execution time and the computational resources we use, a sequential setting with d-choice strategy and $d = 2$ is the best choice. Values of $d > 2$ are unnecessarily expensive for the improvement they give.

On the other hand, we can switch to a *b-batched* approach if we want to parallelize our computation, and in that case the dimension of the batch can lead to the usage of a *(1+β)-choice* instead of a pure 2-choice, improving the gap and reducing the power we need (since we'll use 1-choice and not 2-choice in some cases): this result is, in my opinion, the most important.
The k-questions strategy can help if it requires less resources than the pure *d-choice* one (or we can't access the full information about the loads but only the partial one provided by these questions) and will lead to similar results.