

Theoretical Analysis of the Container Relocation Problem

Mathis Degryse

July 2025

0.1 General Context

In modern container terminals, efficient management of storage yards is crucial to minimise operational delays and costs. This is why a lot of different operations that are frequently done in these terminals are heavily being studied. This goes from yard crane scheduling, in order to optimise the movements of cranes, to the stacking of containers in a way that maximises space usage. A big part of the research has focused on minimising the number of movements (called relocations) that are needed to retrieve a container from the yard. This is because these relocations are costly in terms of time and resources, and they can significantly impact the overall efficiency of the terminal operations.

0.2 The Research Problem

Several problems have been studied in this context, such as the *Container Pre-marshalling Problem* (CPMP) which aims to arrange the containers during less-busy periods of time, in a way that no relocations will be needed when the requests for containers arrive. The problem we studied in this internship is generally referred to as the *Container Relocation Problem* (CRP), and sometimes as the *Block Relocation Problem* (BRP). This problem aims to minimise the number of relocations when retrieving all the containers from the yard, given a sequence of requests. More formally, a bay composed of containers stacks is given, and the algorithm is asked to retrieve these containers in a given order. To do so, it has to free the area above it, by relocating the blocking containers. The goal for the algorithm is to minimise the number of relocations. This problem is NP-hard, as shown by [1] using a reduction from the *Mutual Exclusion Scheduling Problem* (MES) with permutation graphs.

Hence the problem cannot be efficiently solved in general, and several approaches have been proposed to address it. Some of them solve the problem exactly, using a wide range of different methods, such as integer linear programming, dynamic programming [2], Branch and bound or the Corridor method [3]. While others focus on heuristics and metaheuristics as [4] which designs a Branch and Bound algorithm using new lower bounds with excellent performance on practical examples. For a recent survey about optimisation methods for this problem, one can check [5].

While most of the literature study the CRP in a context with full information, it is relevant to assume that only partial information is available, whatever shape this information takes. For instance, [6] consider a model in which several trucks reserve a time-window, yet the arrival sequence of trucks inside each window is unknown. This internship mainly focused on the online variant of the CRP, which is a variant where the requests are revealed one by one, and the algorithm has to decide how to handle each request without knowing the future requests. The goal was then to

increment the algorithm with predictions, which need to be relevant for the algorithm, and to be computable, usually by a machine learning model.

0.3 My Contribution

The initial goal of the internship was to study the CRP by incorporating a prediction model. Thankfully, we had the opportunity to talk with someone from the industry, working at Orano, a company managing uranium barrel facilities.

The problem Orano is facing adds specific aspects to the original *Container Relocation Problem*, such as batch requests (the retrieve order among a batch is let to be decided by the algorithm) and some stochastic predictions on the requests. We realised that theoretical foundations of the CRP was still to be developed and decided to focus on the online variant without predictions first.

I managed to show that the *Leveling* heuristic is optimal in the worst-case setting and some extensions of this theorem. I naturally went to study the case of a uniform random adversary, which is a common model when studying online algorithms. This was the main focus of my internship, and unfortunately, it only led to a conjecture, which had already been stated in [7], and is still open.

During the remaining time, I was able to bring a lower bound on the competitive ratio of deterministic online algorithms for the CRP by using a variant of a very well known problem in online algorithms, the *Paging Problem*. It is a first result in that direction, however, this bound is not optimal, and we have no clue about how far it is from the optimal competitive ratio, even with the help of computational experiments.

Our results do not advance the practical state of the art, but give a theoretical explanation of why *Leveling* work so well in practice, and confirms its optimality.

0.4 Arguments supporting its validity

Beyond the proven results, namely the optimality of *Leveling* in the worst-case setting, and the lower bound on the competitive ratio, we conducted several experiments, mostly to comfort the validity of the conjecture about the average-case optimality of *Leveling*. Some experiments also support the arguments underlying our attempts to prove the conjecture.

About the lower bound on the competitive ratio, I computed the optimal ratio on very small instances (up to 9 containers and 3 stacks), but unfortunately, these do not show a tendency, which makes it hard to draw conclusions about our lower bound.

0.5 Future Work

This internship leaves many doors open, as the conjecture about the average-case optimality of *Leveling* remains unresolved and our lower bound for the competitive ratio is not tight. It would be interesting to either find a better lower bound, or to design an algorithm with a better competitive ratio than *Leveling*.

Another direction that was not deepened during the internship but could lead to interesting results would be to allow randomisation for the algorithm. We already have some premature results but the study could be brought further.

0.6 Notations

We give a table of the notations used throughout the report, so one can go back to this section whenever it is needed. A link to this section for the pdf version is on every footer. These are roughly given in their order of apparition.

Some notations might be used twice in different settings, to avoid having too much of them, and it is done only when they play two equivalent roles in two equivalent frameworks. For instance, F can be a sequence of requests that are in the bottom row of the biggest stack against L (in the CRP framework), as well as a strategy requesting the bottom container of the biggest stack (in the game theory framework).

| | | | |
|-----------------------------|---------------------------------------------------------------------------------------------------------|--------------------------------|-------------------------------------------------------------------------------|
| N | Number of containers | \mathcal{H} | Horizon for the OCRP |
| H | capacity of the stacks | I | Instance of the CRP |
| W | Number of stacks | $A \in \mathcal{A}$ | Set of deterministic online algorithms |
| L | <i>Leveling</i> heuristic | $\text{cost}_A(I)$ | Cost induced by algorithm A on instance I |
| F | Sequence of requests at bottom of biggest stack against L Equivalent strategy in the 2-player game | OPT | Optimal offline algorithm, the concerned problem being clear from the context |
| c_A | Competitive ratio of algorithm A | $s, t \in \mathcal{S}_{N,W,H}$ | Set of configurations |
| $p \in \mathcal{P}(s)$ | Set of permutation of containers | s^p | configuration augmented with a sequence of requests |
| $\text{wcost}_A(s)$ | Worst case cost of A on s | $s \xrightarrow{R} t$ | t can be obtained from s with one relocation |
| $s \xrightarrow{D} t$ | same as \xrightarrow{R} but the relocation goes down | $s \xrightarrow{DL} t$ | same but the relocation goes to one of the two smallest stacks |
| V_i, E_i | Vertices and edges of a structure in the GCRP. | $C_i \subseteq V_i$ | Set of vertices having a container |
| $\text{pot}(s)$ | Potential of the configuration s | $\text{avg}_A^p(s)$ | Average cost of algorithm A by configuration s |
| $\text{avg}_A(s)$ | Normalised average cost of algorithm A by configuration s | $s^{i,j}$ | configuration s with a request on coordinates i, j |
| $\Delta_{\text{pot}}(s, t)$ | Immediate difference of cost between s and t | $\Delta_{\text{fut}}(s, t)$ | Future difference of cost between s and t |

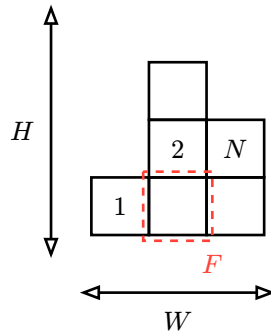


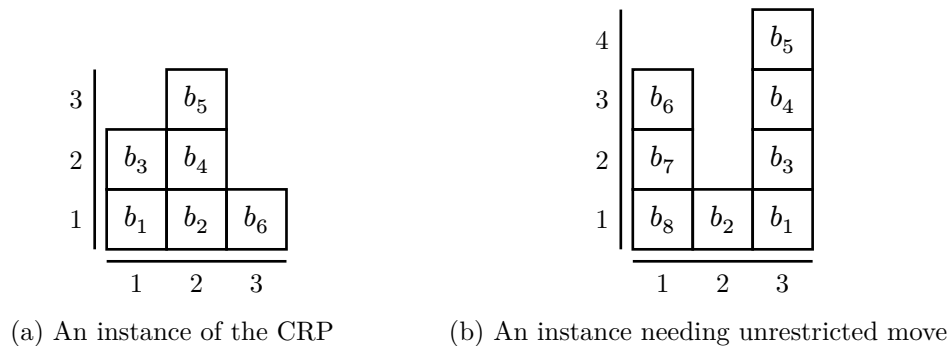
Figure 1: Example instance with notations

The first section of this report will present the *Container Relocation Problem*, its online variant and will go over the results from [8] about the leveling heuristic for the OCRP. The second section will introduce the mathematical settings we use to study the leveling heuristic in a worst-case setting, as well as the proof of its optimality presented in Theorem 2.2. We then try to extend this theorem to a bigger framework, and see the limits of these generalisations. The third section describes our attempt to prove the conjecture Conjecture 3.5, so our main temptative of proof and experimental results comforting the validity of this conjecture. In the fourth section we give the first lower bound on the competitive ratio for the OCRP. In the fifth and last section, we will quickly go over adjacent works that can be done to deepen this study and conclude.

1 The *Container Relocation Problem*

In the *Container Relocation Problem* (CRP), we are given a set of N containers, each located in a yard composed of W stacks each with a maximal capacity of H (we might also refer to the capacity as the maximal height of the stacks). We are also given a sequence of requests, each asking for a specific container to be retrieved from the yard. Each time one of these requests is made, the algorithm has to move (this move is called a relocation) every blocking container (i.e. a container that is above the requested container in the stack) to the top of an other stack. The goal of the CRP is to minimise the number of relocations needed to retrieve all containers in the sequence of requests.

Example. In Figure 2a, the first requested container is the one at coordinates $(1,1)$, hence b_3 has to be relocated. The optimal strategy relocates it on stack 3, above the container b_6 , so it won't have to move again. Then, when b_2 is requested, one can move b_5 and b_4 on stack 1, which is empty. Note that these 3 relocations were unavoidable, therefore this strategy is optimal with a total of 3 relocations.



The CRP as it is relies on assumptions that are mostly implicit in the definition of the problem, we list the other ones here:

- A1 : No new containers arrive during the retrieval process.
- A2 : The requests are known in advance.
- A3 : Only containers that are blocking the requested container may be relocated.

These assumptions are not always realistic : if the assumption A3 does not hold, we talk about the *Unrestricted Container Relocation Problem* (UCRP), which leads to solutions with fewer relocations as in Figure 2b, but is harder to solve due to the increased number of dimensions in the search space. If A1 does not hold, the problem is called the *Dynamic Container Relocation Problem* (DCRP).

Example. in Figure 2b, the optimal solution uses 6 relocations. Indeed, when b_1 is requested, we move b_5 on stack 1, and b_4, b_3 on stack 2. Then when b_2 is requested, we have to move b_3 and b_4 on stack 3, to finally move b_4 again, on stack 2. Yet, if we allow unrestricted moves, we can move

b_2 on stack 1 at the very beginning, and then relocate b_5, b_4, b_3 on stack 2 and avoid any cost in the future, for a total of 4 relocations.

As mentionned above, there are only few theretical results about the CRP. We quickly sum up the litterature about the difficulty of the problem: Whether A3 holds or not, the problem is NP-hard, as shown by [1] using a reduction from the *Mutual Exclusion Scheduling Problem* (MES) with permutation graphs. The same remark can be done about A1 since it makes the problem strictly harder. The NP-hardness is still true with a constant height $H \geq 6$, and the question is still open for $H \in \{3, 4, 5\}$. Even though a lot of well-performing heuristics have been developed, no approximation guarantee has been given for any of these.

Assuming A2 may be too unrealistic in pratice, thus it is often relaxed by different approaches, such as in [7], which studies a variant in which the requests are revealed in batches, i.e. the algorithm knows which are the next k requests, but not the internal order of these requests.

In this study, we focus on the *Online Container Relocation Problem* (OCRP), in which the requests are revealed one by one, and the algorithm has to decide how to handle each request without knowing the future requests. Following the framework of [8], we define the OCRP- \mathcal{H} as the OCRP in which the algorithm has an horizon of \mathcal{H} , meaning he has access to the next $\mathcal{H} + 1$ requests. So the OCRP coincides with the OCRP-0, and the CRP coincides with the OCRP- $(N - 1)$. We will mainly focus on the OCRP, and later on the OCRP-1.

An algorithm for the OCRP is given so few information that only one heuristic, called *Leveling*, has been studied. When *Leveling* has to relocate a container, it selects a stack with minimum height, and if multiple stacks verify this property, it selects the leftmost one, for the sake of determinism. We denote it by L .

This strategy is natural in that framework for multiple reasons. First it minimises the maximal cost that can be induced during the next step, by minimising the maximal height of the stacks. Secondly, it minimises the average cost induced at the next step in case of a uniform random request.

We now introduce the framework we will use to study the performances of algorithms in the online setting, in particular the *Leveling* heuristic.

1.1 Online setting and competitive ratio

For most problems, it is impossible for an online algorithm (an algorithm which has no knowledge of the future) to performs as well as an offline algorithm (one which is given the future information, hence in that case an optimal algorithm for the CRP). A commonly used method to measure the performance of an online algorithm is the competitive ratio, which denotes how much did pay the online algorithm compared to the offline one. More formally, given an online algorithm A and an optimal offline algorithm for the problem, and denoting by $\text{cost}_A(I)$ the cost induced by this algorithm on an instance I of the problem, and OPT the optimal offline algorithm. A is said to be c -competitive for $c \geq 1$ (for minimisation problems) if for all instances I of the problem we have :

$$\text{cost}_A(I) \leq c \cdot \text{cost}_{\text{OPT}}(I)$$

Moreover, the competitive ratio c_A of A is the smallest $c \geq 1$ for which A is c -competitive.

Said otherwise, it guarantees that A pays at least c times than what it could have paid if it knew all requests in advance. An important result shown in [8] about the competitive ratio of *Leveling* is the following :

Theorem 1.1.1 (Competitive ratio of L). *For instances with W stacks and N containers, the competitive ratio c_L of L is equal to $2 \lceil \frac{N}{W} \rceil - 1$.*

The tightness of this ratio is obtained in [8] with the example given in Figure 3a, for which $\text{cost}_L(I) = 5 = 2 \cdot \lceil \frac{9}{3} \rceil - 1$ and $\text{cost}_{\text{OPT}}(I) = 1$, because the algorithm can just move b_6 on b_7 and

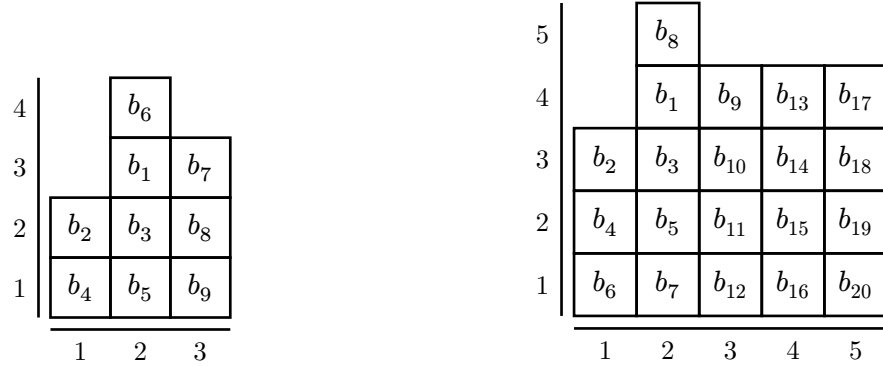
from there, no relocation will be needed. Whereas L will relocate b_6 between stack 1 and 2 until they are emptied. However, this example alone does not ensure the tightness of the bound for any N and W , but still the same shape of example for many different values/

Proposition 1.1.2. *let $W \geq 3$ and N be a multiple of W , there exists an instance I with N containers and W stacks for which the competitive ratio $c_L = 2\lceil \frac{N}{W} \rceil - 1$ is tight.*

Proof. Given W and $N = k \cdot W$, we can consider an instance where the stacks have height $(k - 1, k + 1, k, k, \dots)$ and the order of requests is such that, from top to bottom, we have :

- $b_2, b_4, \dots, b_{2k-2}$ on stack 1.
- $b_{2k}, b_1, b_3, \dots, b_{2k-1}$ on stack 2.
- Any increasing sequence for the other stacks.

We give an example for $W = 5$ and $N = 20$ in Figure 3b. Any instance I of this form satisfies that $\text{cost}_{\text{OPT}}(I) = 1$ because the first request only asks to relocate b_{2k} , which we can move on the third stack. From there, every stacks has its containers requested from top to bottom, so the future induced cost is zero. On the other side, *Leveling* relocates this same container c_L times, which concludes the proof. □



(a) tight example from [8]

(b) generalisation to an instance with $W = 5, N = 20$

We will see later how to generalise this procedure in order to obtain a lower bound on the competitive ratio for any online algorithm.

The ratio obtained in Theorem 1.1.1 is not very satisfying as *Leveling* seems to be pratically good in pratice and to have a very small ratio for most instances. We need to look for an other way to measure its performances, that fits more the intuition we have about the algorithm. Since, as seen above, *Leveling* aims to minimise the maximal/average cost induced at the next step, it is natural to evaluate its cost in the worst and in the average case.

2 Worst case analysis of *Leveling*

We can consider two variants: one that includes a cost of 1 for each retrieved container and one that does not. While for the cost minimisation, these variants only differ of N , and hence are equivalent, for the competitive ratio, these variants are different problems. In this section and the next one, we account a cost of 1 for the retrievals, since it will simplify a lot the obtained formulaes.

For the sake of clarity, we introduce the notion of configuration (of the bay), which is an instance I of the OCRP without the sequence of requests. It is actually the information given to the online algorithm at the beginning of the process (the algorithm may also use its memory of previous

configurations). Let us denote by \mathcal{S} the set of configurations of the OCRP, and by $\mathcal{S}_{N,W,H}$ the set of configurations with N containers, W stacks and height H .

Given $s \in \mathcal{S}_{N,W,H}$, one could obtain an instance of the OCRP by adding a sequence of requests. We won't define it formally, but let us call $\mathcal{P}(s)$ the set of permutations of the containers in s , representing the possible request sequences. Hence, for $p \in \mathcal{P}(s)$, we get an induced instance s^p of the OCRP.

Definition 2.1 (Worst-case cost). *Given an online algorithm A and a configuration $s \in \mathcal{S}_{N,W,H}$, the worst-case cost of A on s is defined as :*

$$\text{wcost}_A(s) = \max_{p \in \mathcal{P}(s)} \text{cost}_A(s^p)$$

Since *Leveling* minimises the next induced number of relocations, it is a good candidate to be optimal in the worst case. To prove this intuition is correct, one can directly find which permutation of the containers in s induces the worst-case. That's why we introduce a game theoretical setting, in which the requests are chosen after each step by an adversary. We show that *Leveling* is optimal against a specific adversary, which requests the deepest container in the stack with maximal height, and that this adversary is optimal against *Leveling*. Moreover, the play induced by these 2 players has a very specific structure, which allows us to give a formula for the value of the game.

Theorem 2.2 (Worst-case cost of L). *Let us call \mathcal{A} the set of deterministic online algorithms for the OCRP, and let F be a permutation that requests the deepest container of a biggest stack against *Leveling*. For any configuration $s \in \mathcal{S}_{N,W,H}$, denoting, $s_1 \geq s_2 \geq \dots \geq s_W$ its stacks heights in decreasing order, we have :*

$$\begin{aligned} \text{wcost}_L(s) &= \min_{A \in \mathcal{A}} \text{wcost}_A(s) = \text{cost}_L(s^F) \\ &= \begin{cases} s_1 + \sum_{i=2}^W \max\left(s_i, \left\lceil \frac{n-i+1}{W-1} \right\rceil\right) + \sum_{i=W+1}^n \left\lceil \frac{n-i+1}{W-1} \right\rceil & \text{if } n \leq (W-1)H + 1 \\ s_1 + \frac{h(2H-h+1)}{2} + (W-1)\frac{H(H+1)}{2} & \text{otherwise, with } h = n - 1 - (W-1)H \end{cases} \end{aligned}$$

Let us work out an example.

Example. Consider the configuration $(6, 5, 1, 1)$ in Figure 4a, we represent it, as every other configurations, with stacks in decreasing order of height to match the formula. Note that the process is entirely deterministic since the algorithm used is *Leveling*, and that the sequence of requests is F .

When the requests are chosen by F , the obtained sequence of configurations has the property that every configuration except the first one has an empty stack. Hence we add with a red dashed line the shape of the flattest obtainable configuration with the same number of containers. Moreover, *Leveling* has the nice property to only relocate containers inside the dashed red shape.

Let us start the example walkthrough, when the request is done for the configuration $(6, 5, 1, 1)$, the algorithm pays exactly $s_1 = 6$. as stated in the formula. We reach the configuration $t = (5, 4, 3, 0)$ in Figure 4b. Note that the stack t_1 is the stack s_2 to which some containers **might** have been added. Indeed,

- Either we have $s_2 < 4$ (the height of the red line), and *Leveling* would have to relocate containers on it, until it reaches the dashed line
- Either we have $s_2 \geq 4$ and no container are relocated to this stack since *Leveling* does not relocate above the red line, which is the case in our example.

Hence, we get that *Leveling* has to relocate $t_1 = \max(s_2, 4) = s_2 = 5$ containers.

After this, we reach a stable state in Figure 4c: the configuration is “leveled”, i.e. it is as flat as possible knowing it has only $W - 1$ stacks. From here, the number of relocated containers will always be the height of the dashed line. This stable state is reached in at most $W - 1$ steps. Noting that the line has height $\lceil \frac{n-i+1}{W-1} \rceil$ on stack i , we get the formula above.

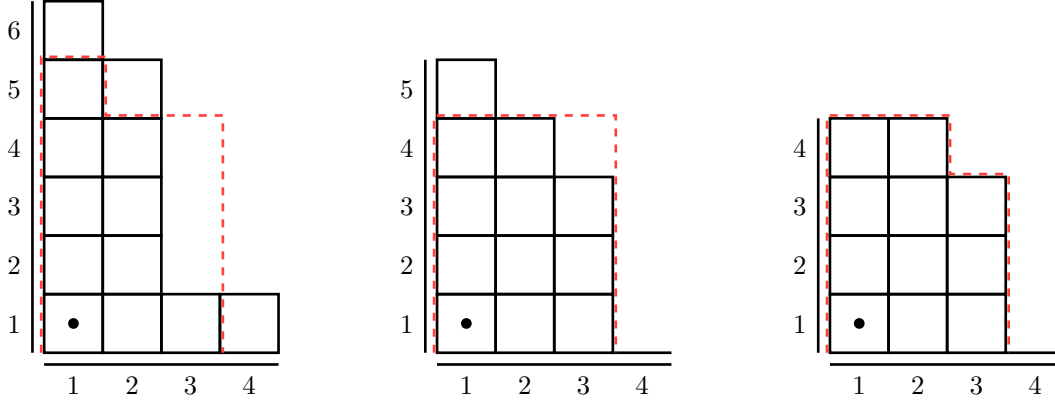


Figure 4: Detailed example for Theorem 2.2

2.1 Sketch of the proof

Let us give some of the needed definitions to understand the core of the proof, while all the details of the definitions and the proofs can be found in Section 6.

Since the configurations don't have any attached request, it is natural to define it as a vector of W natural numbers, the heights of the stacks. Moreover, while an online algorithm may be able to distinguish the stacks, thanks to the previous relocations, the value of the game from that point is the same if we swap these two stacks. Hence we consider that the stacks are only distinguishable by their heights, so we can consider the components of the vector as being ordered in decreasing order.

Definition 2.1.1 (Configuration). A configuration $s \in \mathcal{S}_{N,W,H}$ is a vector (s_1, s_2, \dots, s_W) such that $\sum_{i=1}^W s_i = N$ and $H \geq s_1 \geq s_2 \geq \dots \geq s_W \geq 0$.

To study *Leveling*, we need a definition that matches its behavior. Since it always relocated the containers on the stack with minimal height, it chooses a configuration of the bay which is minimal for a relation order we define now.

Definition 2.1.2 (down relocation). Given $s, t \in \mathcal{S}_{N,W,H}$, we say that s downs to t which we denote by $s \xrightarrow[D]{DL} t$ if t can be obtained from s by relocating a container from a stack to a stack with lower height. If the container is relocated on the smallest stack, we note $s \xrightarrow{DL} t$.

And finally, let us call F the adversary that requests the deepest container in the stack with maximal height. We have three properties to prove : (1) L is optimal against F , (2) F is optimal against L , and (3) the play induced by these two strategies has the value given in Theorem 2.2.

We prove (3) by induction. Note that this value is splitted in 2 cases because the adversary cannot choose deep containers if the bay is almost full, because the algorithm may be technically unable to relocate them. Let us now give an intuition about the proof of (3).

When L and F play against each other, F makes sure a stack is always empty, because since he asks the bottom container of its stack, it will be empty after the relocations. And since L relocates at the lowest possible position, it will always relocate the containers below the line, of height $\lceil \frac{n-i+1}{W-1} \rceil$, which can be seen in Figure 4. Indeed, at the i -th request, there are $n - i + 1$ containers, distributed among $W - 1$ stacks, one being empty when $i \geq 2$. Hence the average height among those stacks is

$\frac{n-i+1}{W-1}$, and one of the states has at least $\lceil \frac{n-i+1}{W-1} \rceil$. The number of containers above this line is fixed at the beginning, since no containers will go above the line during the whole process. And F has the guarantee that the stack with maximal height will always be above the line. In the end, L pays what is above the line at the beginning plus the sum of the heights of the line at each step.

We won't go over the case where $N > (W - 1) \cdot H + 1$, since it easily reduces to the initial case. Once we prove (3), we directly get that L prefers states that are leveled when playing against F .

Proposition 2.1.3. *Let $s, t \in \mathcal{S}_{N,W,H}$, if $s \xrightarrow{D} t$, then $\text{cost}_L(t^F) + 1 \geq \text{cost}_L(s^F) \geq \text{cost}_L(t^F)$.*

This property is the core of the proof, it directly implies (1) by induction, and (2) by induction with some more structural arguments.

2.2 Discussion and Extensions of *Leveling* optimality

This theorem comforts the intuition that *Leveling* is the best we can do when having no information about the future requests, yet *Leveling* may be a good heuristic in a more general context. The aim of this section is to find a more general problem in which *Leveling* is still optimal.

We can first take a look at the CRP variants mentionned earlier, the UCRP and DCRP which respectively allows the algorithm to relocate a container that is not blocking the requested one, and the adversary to add containers during the process. In both cases the proof of Theorem 2.2 can be adapted without too many complications.

Proposition 2.2.1 (Extensions of Theorem 2.2). *We have the following :*

- *Theorem 2.2 holds for the UCRP*
- *Theorem 2.2 holds for the DCRP and UDCRP by replacing $\text{val}_{L,F}(s)$ by $\text{val}_{L,F}(t)$, where t is the state obtained by adding the k containers the adversary is allowed to add on lowest positions of the state s .*

Another way to generalise the problem would be to directly tackle the structure of stacks. Since the algorithm has no information on the future requests, the stacks are reduced to a value : their height. On the adversary side, a stack is just a value and a set of possible number of containers he can request at once. For example, with a stack of size 7, the adversary can request any number of containers in the set $\llbracket 0, 6 \rrbracket$. We could wonder what would happen if we reduced this set of possibilities for the adversary.

This generalisation is motivated by theoretical and concrete examples, where the used structure is slightly different from the stacks we find in port bays, which is the most studied case. In Figure 5 is given a representation of double-sided queues, as an example of other container system. The bay is composed of several queues in which container can be retrieved from both sides.

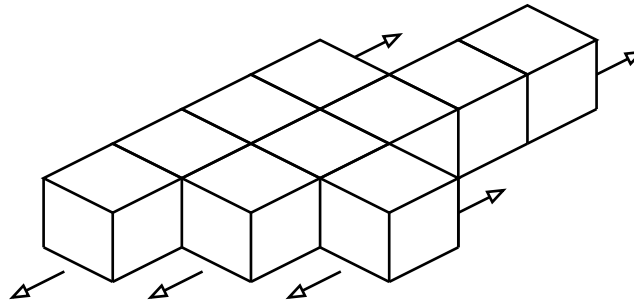


Figure 5: Double sided queue, state (4, 5, 1)

Our first attempt to generalise the problem was to give the adversary a function $f : \mathbb{N} \rightarrow \mathbb{N}$ which tells him how deep in a stack he can request a container, so that its set of possible request would be $\llbracket n - f(n), n \rrbracket$ for a structure of size n . In the case of stacks, the function f is the mapping $n \mapsto n - 1$. It is clear that *Leveling* is not optimal in that setting because f could be chaotic, so we can require f to be increasing and continuous (we will detail what it means just below). In the end, we require f to verify the following conditions, for all $n \in \mathbb{N}$:

- $0 \leq f(n) < n$
- $f(n) \leq f(n+1) \leq f(n) + 1$

This model is way more generic, it includes the double-sided queue given above. Indeed, choosing $f : n \mapsto \max(0, \lfloor \frac{n-1}{2} \rfloor)$, we get a problem roughly equivalent (up to some details) to it. Yet this same example is a case for which *Leveling* is not optimal, as shown by the following example. The configuration $(3, 3, 0)$ needs at least 2 relocations, yet the configuration $(4, 2, 0)$ only needs 1, even if it is less leveled.

It alone does not say much because *Leveling* may just be a particular case of some more clever heuristic in this setting. Yet we tried several variants of *Leveling*, such as one that consider in priority relocation such that the targeted stack verifies $f(h) = f(h+1)$. In this particular example seen above, it will choose $(4, 2, 0)$ over $(3, 3, 0)$, since $f(4) = f(3)$, yet other examples still tackle this heuristic.

Let us instead consider our proof of Theorem 2.2 and find a model that satisfies everything used in it. Since the proof needs the adversary to be F , we need a model that allows it to exist. We get that if the set of possible requests for the adversary verifies that all the containers in the structure can be requested at once, F can be played by the adversary and thus L is the optimal strategy for the worst-case setting. We give here a quite formal definition of the GCRP, as it was never introduced (at least at our knowledge).

Definition 2.2.2 (Generalised CRP). *Let us define the GCRP, in which a bay of W structures is given. Each structure numbered $i \in \llbracket 1, W \rrbracket$ is a directed acyclic graph (V_i, E_i) . The problem is the same as the CRP, the difference is it not played on stacks but on graphs. N containers are initially placed on the vertices, with at most one container per vertex. They are given by sets $C_i \subseteq V_i$, which must verify $\sum_{i=1}^W |C_i| = N$. The graph edges form a precedence relation saying which slots are “below” others. If we have $(u, v) \in E_i$, then $v \in C_i \implies u \in C_i$, meaning there must be a container in slot u to have one in slot v .*

The problem is naturally defined on these structures, a sequence of requests is given, as an order on the elements of $\cup_{i=1}^W C_i$, and the algorithm will need to relocate the blocking containers (the successors of the requested container in its graph) in other graphs.

In the scope of this study, we may or may not allow to move a container in a different slot of the same structure, even if it is not forbidden by the precedence relation.

We give an example of such a structure to help understanding this quite heavy definition in Figure 6. Each stack is replaced by a 2-dimensionnal structure of containers. We can imagine that a container is retrieved by a vehicule with extendable arms, hence a container can be retrieved/relocated if there is no blocking container above it, and if there is no container between it and the vehicle. For such a 2-dimensionnal arrangement, we get the directed acyclic graph (V, E) depicted in the same figure. The subset $C \subseteq V$ is simply the set of vertices on which a container is present.

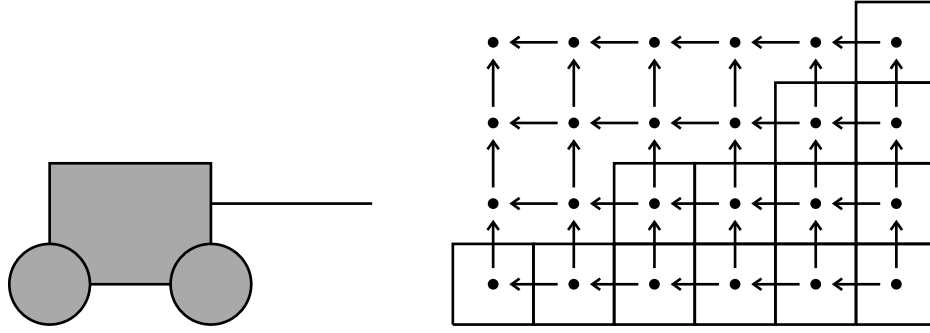


Figure 6: 2-dimensionnal arrangement

The notion of GCRP induces an OGCRP, in which the requests are revealed one by one, exactly as for the OCRP. We give the following result for the leveling heuristic in the worst-case setting for the OGCRP.

Proposition 2.2.3 (restricted requests). *Let us consider an instance of the OGCRP given by $((V_i, E_i), C_i)_{1 \leq i \leq W}$. If for every $1 \leq i \leq W$, the graph V_i has exactly one source vertex, i.e. a vertex with no predecessor, then Theorem 2.2 holds.*

The prerequisites of this proposition are satisfied by the structure shown in Figure 6, but also by the real-life case from Orano, which is a bit more complex. In this case, a structure is a 3-dimensionnal arrangement, of depth D , height H and width 2. To manipulate a container (a barell in that case) with coordinates (d, h, i) , one needs to free the coordinates $(d - 1, 1, 1)$, $(d - 1, 1, 2)$, $(d, h + 1, 1)$ and $(d, h + 1, 2)$. Unfortunately, such a structure has two sources, of coordinates $(D, 1, 1)$ and $(D, 1, 2)$, so the property does not apply.

2.3 Extending the look-ahead

There is no written proof of this section results at the moment.

The problem gets harder with a bigger look-ahead because it enables way more possibilities for the algorithm. In this subsection, we will consider the OCRP-1, i.e. the OCRP in which the algorithm has access to the next 2 requests. The *Leveling* strategy still seems suitable, but it would sometimes relocate a container above the next request, which would seem unnatural. Hence we define the *Leveling-avoid* heuristic, which plays as *Leveling* but marks a stack it will avoid for relocations steps. If the next requested container is among the blocking one, then it marks the biggest stack. Else, it marks the stack containing the next request.

We think that the proof of Theorem 2.2 carries through this setting as well.

3 Random uniform adversary

Both competitive ratio and worst-case analysis suffer from the fact that in practice, the requests may not be adversarial. Hence one may prefer to evaluate the performance of a heuristic on a random case. Several experiments already have been done to evaluate the performances of *Leveling* on random cases, such as in [8] and [7]. Immediate conclusions that come from these are that *Leveling* performs very well compared to the guarantee given by Theorem 2.2 and Theorem 1.1.1 in average.

We decide for this section to keep the game point of view, even if the adversary is not a formal “adversary” since he has no choice in his requests, which are chosen uniformly at random. Hence instead of considering a uniform permutation of the requests, we will tackle the problem with an inductive pattern.

We will as well remove some constraints to make the study more convenient, as removing the maximal height does not change how hard it is to prove the conjecture, but makes the details way more clear. Hence $\mathcal{S}_{N,W}$ denotes the set of configurations composed of N containers on W stacks.

For a configuration $s \in \mathcal{S}_{N,W}$, each container has the same probability of being requested first, i.e. $\frac{1}{N}$. After a specific container being requested, an arbitrary given strategy $A \in \mathcal{A}$ chooses the relocation, and we get a new configuration $t \in \mathcal{S}_{N,W}$. With this process, we can design an induction formula for the average cost of a configuration. Thus we only need to compute how much relocations are done in average at the first step.

Proposition 3.1 (Average induced cost). *Given a configuration $s \in \mathcal{S}_{N,W}$ the average induced number of relocations at the first step is*

$$\frac{1}{2} + \frac{1}{2N} \cdot \sum_{i=1}^W s_i^2$$

Proof. This induced cost is just, up to a factor $\frac{1}{N}$ due to probabilities, the sum of the depths of each containers in the configuration.

$$\begin{aligned} \mathbb{E}(\text{induced cost at first step}) &= \sum_{i=1}^W \sum_{j=1}^{s_i} \frac{1}{N} \cdot \mathbb{E}(\text{induced cost if } i, j \text{ is requested}) \\ &= \frac{1}{N} \sum_{i=1}^W \sum_{j=1}^{s_i} (s_i - j + 1) \\ &= \frac{1}{N} \sum_{i=1}^W \frac{s_i(s_i + 1)}{2} \\ &= \frac{1}{2N} \sum_{i=1}^W s_i^2 + \frac{1}{2N} \sum_{i=1}^W s_i = \frac{1}{2} + \frac{1}{2N} \cdot \sum_{i=1}^W s_i^2 \end{aligned}$$

□

We see that the value $\sum_{i=1}^W s_i^2$ will be important in the analysis since it is, up to an additive constant and a constant factor, the immediate average cost of a configuration, and it is also a measure of how flat a configuration is. It has a similar role to a potential, hence we define it as one.

Definition 3.2 (potential of a configuration). *We define the potential of a configuration $s \in \mathcal{S}_{N,W}$ as $\text{pot}(s) = \sum_{i=1}^W s_i^2$.*

Indeed we have the following lemma, which establishes that relocating a container downward diminishes the potential of a configuration.

Lemma 3.3. *Let $s, t \in \mathcal{S}_{N,W}$ such that $s \xrightarrow{D} t$. We have $\text{pot}(s) > \text{pot}(t)$.*

Proof. Immediate by convexity of the square function. □

Now we can define a simple function to measure the average cost induced by an algorithm. Before, we need a final notation to depict a configuration with a specific request. We denote a configuration $s \in \mathcal{S}_{N,W,H}$ with a request on coordinates i, j by $s^{i,j}$. Hence an algorithm $A \in \mathcal{A}$ takes as input a configuration with a request and outputs a configuration without request.

Definition 3.4 (average cost). *Given a state $s \in \mathcal{S}_{N,W}$ and an algorithm $A \in \mathcal{A}$, the average cost induced by A on s is the function verifying*

$$\text{avg}_A^p(s) = \begin{cases} 0 & \text{if } N = 0 \\ \frac{1}{2} + \frac{1}{2N} \cdot \text{pot}(s) + \frac{1}{N} \sum_{i=1}^W \sum_{j=1}^{s_i} \text{avg}_A^p(A(s^{i,j})) & \text{otherwise} \end{cases}$$

Yet we might prefer its normalised and integral version $\text{avg}_A(s) = 2N! \cdot \text{avg}_A^p(s) - (N-1)N!$ which verifies :

$$\text{avg}_A(s) = \begin{cases} 1 & \text{if } N = 0 \\ (N-1)! \cdot \text{pot}(s) + \sum_{i=1}^W \sum_{j=1}^{s_i} \text{avg}_A(A(s^{i,j})) & \text{otherwise} \end{cases}$$

Everything is defined to state the conjecture, that was already given in a different form in [7]:

Conjecture 3.5 (AVGOPT). *For any $s \in S_{N,W}$,*

$$\text{avg}_L(s) = \min_{A \in \mathcal{A}} \text{avg}_A(s)$$

The remaining of this section will be dedicated to our attempt to prove Conjecture 3.5, since it has been the main focus during this internship. Thanks to the structure of the problem and *Leveling* behavior, as for the proof of Theorem 2.2, we get an equivalent statement of this conjecture which is easier to manipulate :

Proposition 3.6 (Equivalence of AVGOPT and LINC). *The Conjecture 3.5 is correct if and only if for every $s, t \in \mathcal{S}_{N,W}$ such that $s \xrightarrow{DL} t$, we have $\text{avg}_L(s) \geq \text{avg}_L(t)$. We call this property LINC.*

Proof. Let us suppose that the conjecture is correct, and that there exists $s, t \in \mathcal{S}_{N,W}$ such that $s \xrightarrow{D} t$ and $\text{avg}_L(s) < \text{avg}_L(t)$ towards a contradiction. Let us note $1 \leq i, j \leq W$ the stacks indices involved in the downward relocation from s to t , i.e. t is obtained from s by relocating a container from i to j . Hence we consider a configuration w , same as s with two more containers on a stack $k \notin \{i, j\}$ and one less on stack i , which exists because $W \geq 3$. If the random request for w is on coordinates $(k, s_k - 1)$, one container has to be moved and several choices appear, among which we find t , that will be chosen by *Leveling*, and s which won't be chosen even though it is a better configuration, hence we get our contradiction, *Leveling* not being optimal.

In the other direction, let us suppose that for every $s, t \in \mathcal{S}_{N,W}$ such that $s \xrightarrow{DL} t$, we have $\text{avg}_L(s) \geq \text{avg}_L(t)$, then we prove the conjecture by induction on N . For $N = 0$ it is trivial, since there is only one configuration and no decision to take. Let us suppose the conjecture holds for some $N \in \mathbb{N}$, and let $s \in S_{N+1,W}$. We have for all $A \in \mathcal{A}$

$$\begin{aligned} \text{avg}_A(s) &= (N-1)! \cdot \text{pot}(s) + \sum_{i=1}^W \sum_{j=1}^{s_i} \text{avg}_A(A(s^{i,j})) \\ &\geq (N-1)! \cdot \text{pot}(s) + \sum_{i=1}^W \sum_{j=1}^{s_i} \text{avg}_L(A(s^{i,j})) \quad (\text{Induction hypothesis}) \\ &\geq (N-1)! \cdot \text{pot}(s) + \sum_{i=1}^W \sum_{j=1}^{s_i} \text{avg}_L(L(s^{i,j})) \quad (\text{Because } A(s^{i,j}) \xrightarrow{DL} L(s^{i,j})) \\ &= \text{avg}_L(s) \end{aligned}$$

which concludes the proof. \square

We just reduced the optimality of L among all deterministic online algorithms to a property about L , which is clearer. Yet LINC is still not proper to work with, because the relation $s \xrightarrow{DL} t$

is not stable by requests. Let us consider the configuration $s = (4, 3, 2, 0)$ and $t = (3, 3, 2, 1)$ such that $s \xrightarrow{DL} t$ by relocating from stack 1 to 4. Consider a common request on stack 3, on the lowest container. After the moves from L on both configurations, we get $s' = (4, 3, 1, 0)$ and $t' = (3, 3, 2, 0)$. We have $s' \xrightarrow{D} t'$ but no longer $s' \xrightarrow{DL} t'$, so LINC will be hard to prove by induction.

That is why we consider the property SLINC (for Strong LINC) which implies the correctness of the conjecture, but it might be possible for SLINC to be false while the conjecture is true.

Conjecture 3.7 (SLINC). *For every $s, t \in \mathcal{S}_{N,W}$ such that $s \xrightarrow{D} t$, we have $\text{avg}_L(s) \geq \text{avg}_L(t)$.*

The formula for $\text{avg}_L(s)$ has a lot of constants and the induction part will be similar if the two configurations considered are similar. The cost difference between s and t can be splitted in two parts, the difference of potential, which is the difference between the number of relocations done instantly, in the first step, and the future difference, which is the difference of cost which will be induced later in the process. We have, for $s, t \in \mathcal{S}_{N,W}$ such that $s \xrightarrow{D} t$:

$$\begin{aligned} \Delta(s, t) &= \text{avg}_L(s) - \text{avg}_L(t) \\ &= \underbrace{(N-1)! \cdot (\text{pot}(s) - \text{pot}(t))}_{\Delta_{\text{pot}}(s, t)} + \underbrace{\sum_{i=1}^W \sum_{j=1}^{s_i} \text{avg}_L(L(s^{i,j})) - \sum_{i=1}^W \sum_{j=1}^{t_i} \text{avg}_L(L(t^{i,j}))}_{\Delta_{\text{fut}}(s, t)} \end{aligned}$$

Example. Let us consider the configurations $s = (3, 1, 0)$ and $t = (2, 1, 1)$ represented in Figure 7, noting that $s \xrightarrow{D} t$ by relocating from stack 1 to 3. We have $\Delta_{\text{pot}}(s, t) = 3! \cdot ((3^2 + 1^2 + 0^2) - (2^2 + 1^2 + 1^2)) = 24$, and to compute $\Delta_{\text{fut}}(s, t)$, we need to look at all the successors of s and t . To do this, we iterate over all the 4 possible requests on both configurations, and look at which configuration is obtained when processing *Leveling*. For s , we get $(2, 1, 0)$, $(1, 1, 1)$, $(2, 1, 0)$ and $(3, 0, 0)$, on the other hand, for t , we get $(1, 1, 1)$, $(2, 1, 0)$, $(2, 1, 0)$ and $(2, 1, 0)$. In the end, we have $\Delta(s, t) = 4 + \Delta((3, 0, 0), (2, 1, 0))$ which still has to be computed recursively !

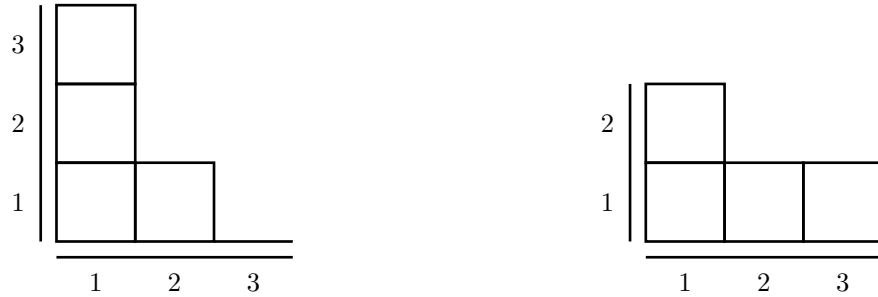


Figure 7: Two states, $(3, 1, 0)$, $(2, 1, 1)$

The term $\Delta_{\text{pot}}(s, t)$ is always positive, so we don't have to put interest in it for the moment. $\Delta_{\text{fut}}(s, t)$ can be decomposed into N differences between average values of the successors, which for some are positive. Indeed, if the request is on a stack different from u and v , then L will keep the downward relocation relation between s and t and thus the induction hypothesis can be used to get the positivity of the term. In our example from Figure 7, if the request is on stack 2, which is not concerned by the relocation from 1 to 3, then we can compare in parallel what happens to s and t . For s , we get $(3, 0, 0)$ and for t we get $(2, 1, 0)$, yet we have $(3, 0, 0) \xrightarrow{D} (2, 1, 0)$, which could be used for the induction.

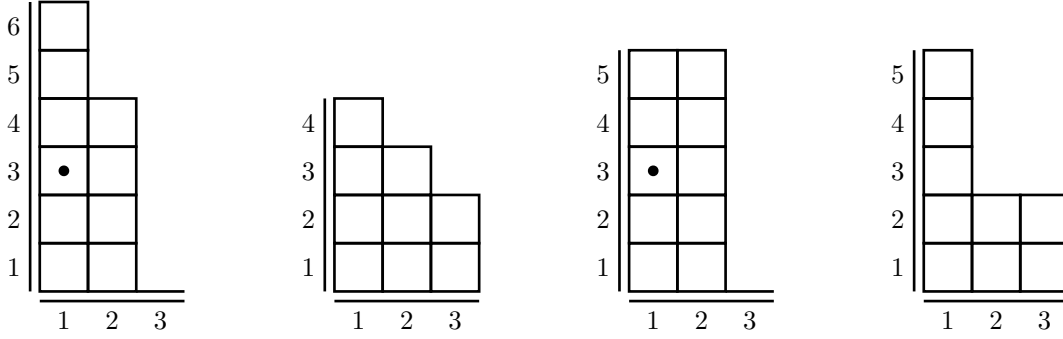


Figure 8: States s, s', t, t'

The terms with requests on stacks u and v (the ones concerned by the relocation) for both s and t remains, hence we need to show that :

$$\Delta_{\text{sub}}(s, t) = \sum_{\substack{i=1 \\ i \notin \{u, v\}}}^W \sum_{j=1}^{s_i} \text{avg}_L(L(s^{i,j})) - \sum_{\substack{i=1 \\ i \notin \{u, v\}}}^W \sum_{j=1}^{t_i} \text{avg}_L(L(t^{i,j})) \geq 0$$

Since there are at many terms on the side of s than on the side of t , it is natural to pair this term in such a way so that every difference is positive. Hence we call *pair* a pair of requests, the first one being on s , and the second one on t , such that the stacks of these requests are neither u or v . It may be possible that this solution is a dead-end, since we could have $\Delta_{\text{fut}}(s, t) \geq 0$ but $\Delta_{\text{sub}}(s, t) < 0$. Yet it is necessary to try this because a wide family of configurations verify $\Delta_{\text{fut}}(s, t) = \Delta_{\text{sub}}(s, t)$, i.e. any configuration with only 2 non-empty stacks. So from here, the goal is to show that $\Delta_{\text{sub}}(s, t) \geq 0$ by pairing the terms together, and using the induction hypothesis, which can give the positivity of a pair s', t' if $s' \xrightarrow{D} t'$.

Yet some of these pairings might be in the wrong direction : if $s = (6, 4, 0)$ and $t = (5, 5, 0)$, as in Figure 8, and the request is made on stack 1 at height 3, then the obtained configurations are $s' = (4, 3, 2)$ and $t' = (5, 2, 2)$, so that $s \xrightarrow{D} t$ but $t' \xrightarrow{D} s'$. We will see later that these bad pairings are unavoidable, excepted in some scenarios, as stated in the next proposition :

Proposition 3.8. *Let us suppose SLINC for every configuration $s, t \in \mathcal{S}_{n,W}$ for $n \leq N - 1$. Let $s, t \in \mathcal{S}_{N,W}$ such that $s \xrightarrow{DL} t$, we have $\text{avg}_L(s) \geq \text{avg}_L(t)$.*

Proof. Let us notice that we have $L(s^{u,j}) = L(t^{u,j})$ for $1 \leq j \leq s_u - 1$. In general we have that $L(s^{u,s_u}) = L(t^{v,t_v})$ and that $L(s^{v,j}) \xrightarrow{D} L(t^{v,j})$ for $1 \leq j \leq s_v$, and for every $i \neq u$ and $1 \leq j \leq s_i$ we have $L(s^{i,j}) \xrightarrow{D} L(t^{i,j})$. Finally, we can notice that since the relocation from s to t is going on stack v , which is the smallest, and since L relocates on the smallest stack, a request on stack u leads both states to be the same, i.e. for $1 \leq j < s_u$, $L(s^{u,j}) = L(t^{v,j})$. Hence we have $\Delta_{\text{fut}}(s, t) \geq 0$ and therefore $\text{avg}_L(s) \geq \text{avg}_L(t)$. \square

But it is not true in general and we need to tackle bad pairings in some way. To limit their impact on the value, one can use $\Delta_{\text{pot}}(s, t)$ which gives some budget we can use on $\Delta_{\text{fut}}(s, t)$, to prove that $\text{avg}_L(s) - \text{avg}_L(t)$ is positive.

Proposition 3.9. *For any $s, t \in \mathcal{S}_{N,W}$ such that $s \xrightarrow{D} t$, assuming LINC for configurations of size N and less, we have $\text{avg}_L(s) - \text{avg}_L(t) \leq 2 \cdot N!$.*

Proof. Let us call u, v the stack indices involved in the relocation from s to t . Let us consider a strategy M for the configuration s , such that for a given sequence of requests on s , it simulates *Leveling* on s and t in parallel, getting a sequence of configurations s_1, \dots, s_k and t_1, \dots, t_k . We have that for every $1 \leq i \leq k$, either $s_i \xrightarrow{D} t_i$ or $t_i \xrightarrow{D} s_i$. let us call u_i the stack index in s_i of the container that has to be relocated (either up or down) to obtain t_i . Once a request is made on u_i (it might never happen), the strategy M relocates this container, both the simulation of t and the process on s becomes the same configuration and no difference of cost is induced in the future.

Before the request i , the same requests were done on each side, either on a stack that had the same size in s and t , or on a stack that had one more container in t than in s , so M does less relocations on s than L does on t . Then, when the request is on u_i for some i , M relocates one more container. We get that M relocates on s one more container at most than L on t , and this is true for every request sequence.

We get that $\text{avg}_L^p(s) \leq \text{avg}_M^p(s) \leq \text{avg}_L^p(t) + 1$. It suffices to multiply this inequality by $2N!$ to get the result. \square

This property is very useful because if we find a pairing of successors $L(s^{i,j})$ and $L(t^{i,j})$ such that $L(t^{i,j}) \xrightarrow{D} L(s^{i,j})$, which is bad for the induction hypothesis that says this value is negative, then we know this value is greater than $2(N-1)!$ and can use some of the credits from $\Delta_{\text{pot}}(s, t)$ to compensate.

More precisely, the allowed margin of error for $\Delta_{\text{pot}}(s, t)$ is exactly $(N-1)! \cdot \Delta_{\text{pot}}(s, t)$, so when looking for a bijection between requests of s and t , the bijection is allowed to miss (leading to a use of the induction hypothesis in the wrong direction) at most $\frac{\Delta_{\text{pot}}(s, t)}{2}$ times, which is exactly equal to $\Delta(s, t)$: the height lost by the container relocated from s to t .

Unfortunately, such a pairing does not always exist, as in the following example.

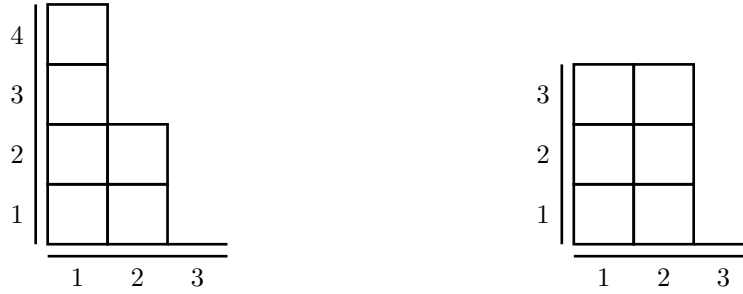


Figure 9: Example of a dead-end for pairings

Example. Let us consider $s = (4, 2, 0)$ and $t = (3, 3, 0)$, as in Figure 9. We have $\Delta_{\text{pot}}(s, t) = 2 \cdot 5!$, and here is a list of successors from s and t :

- $(4, 1, 0), (4, 1, 0), (3, 2, 0), (3, 2, 0), (2, 2, 1), (2, 2, 1)$
- $(3, 2, 0), (3, 2, 0), (3, 2, 0), (3, 2, 0), (3, 1, 1), (3, 1, 1)$

Between these two sets of successors, there is no pairings for which the induction hypothesis can be used on everyone. But we can allow one bad pairing, indeed, let us pair $(2, 2, 1)$ as successor of s and $(3, 2, 0)$ as successor of t . We have by Proposition 3.9, that $\Delta((2, 2, 1), (3, 2, 0)) \geq -2 \cdot 5!$, hence $\Delta((2, 2, 1), (3, 2, 0)) + \Delta_{\text{pot}}(s, t) \geq 0$. Yet, even with this bad pairing removed, there is no convenient bijection between the 5 remaining configurations of each set. In conclusion, a different proof technique, or a better bound than Proposition 3.9 is needed here.

To finish this section, we present some experimental results that support this conjecture: for some values of W , we computationally checked the optimality of *Leveling* for every configuration having less than some number $M(W)$ that is given in Table 1. To do so, we use SLINC so that it is not necessary to compute the optimal strategy by branch and bound, the only counterpart is that a check is valid only if every smaller configurations has been checked as well for the induction to work, which is not really an issue. Moreover, we use LINC instead of SLINC since we have less configuration pairs to check.

We can notice a big difference between the values for $W = 3$ and $W > 3$. This is because :

- The code has been ran during 1 hour for $W = 3$ and only about 1 minute for the other cases.
- An optimised version of the code has been implemented for the case $W = 3$, since it stood out as a special case to us in the theoretical analysis.

Table 1: Number of containers for which AVGOPT has been checked, for different values of W

| W | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|--------|-----|----|----|----|----|----|----|----|----|----|----|
| $M(W)$ | 850 | 85 | 70 | 60 | 55 | 53 | 50 | 48 | 45 | 42 | 40 |

A last interesting observation is that in the very specific case of $W = 3$ which is the first non trivial case, the value $\Delta_{\text{fit}}(s, t)$ seems to be always positive, which is not the case for $W \geq 4$, with plenty of counter examples at disposal. This 2nd conjecture led us to think the case $W = 3$ would be simpler to deal with, but it has been a dead-end as well. This small conjecture has been checked for every configuration until $N = 850$.

Several other strategies have been tried to tackle this conjecture, but none went “as close” as this one, and were all about showing SLINC.

4 Competitive ratio Lower Bound

We recall that in this section, the retrieval of a container does not induce a cost of 1 for the algorithm.

Although the competitive ratio of *Leveling* was determined in [8], the paper states that no lower bound on the competitive ratio was known, and this question remains open.

A constant lower bound can easily be found by analysing some examples. A quick enumeration of permutations and strategies for the state $(2, 2, 2)$ leads to the fact that OPT pays at most 4, and Theorem 2.2 says that for any online algorithm, there exists a sequence of requests which induces a cost of at least 5. Hence we get a lower bound of $\frac{5}{4}$.

This process is on one hand, extremely long, and of no theoretical interest. Even automating the process does not lead to significantly bigger ratios.

In this section, we will play the role of the adversary, choosing the requests so that the algorithm we are playing against has to do a lot of relocations, while we are making sure that OPT won’t have to do much. Yet this task seems impossible since determining the number of relocations OPT has to make is NP-hard, and no other lower bound than the number of blocking containers is known. That’s why we will make the analysis as simple as possible for OPT, by making the algorithm move the same container during the whole process.

Let $A \in \mathcal{A}$ be a deterministic online algorithm. We ask to move a chosen container that we call \diamond by requesting the one just below. When A chooses where to relocate it, we ask to move it again, etc... This process is exactly what happens in the example giving the tightness of Theorem 1.1.1 in [8], as well as the generic one we presented in Proposition 1.1.2.

The core of the argument is an analogy with the *Paging Problem* which initial definition is as follows : We are given a cache of size k which can contain pages. At each step, a page is requested, if it is already in the cache, nothing happens, otherwise, the algorithm pays a cost of 1 and chooses a page in the cache to evict in order to let the new page in, this is called a page fault. The goal for the algorithm is to minimise the number of page fault. The deterministic competitive ratio of this problem is k , the size of the cache [9].

In our case, we will have to slightly modify the problem. See each one of the W stacks as a page, and the number of containers in that stack is the number of occurrences of that page (meaning this page can be requested by the adversary only a limited ammount of times). The algorithm is given a cache of size $W - 1$, which contains every page on which \diamond is not. When we request the container below \diamond , it is as if we were asking for a page, still playing the role of an adversary. When the algorithm relocates \diamond , it chooses which page it evicts from the cache to replace it by the one we requested.

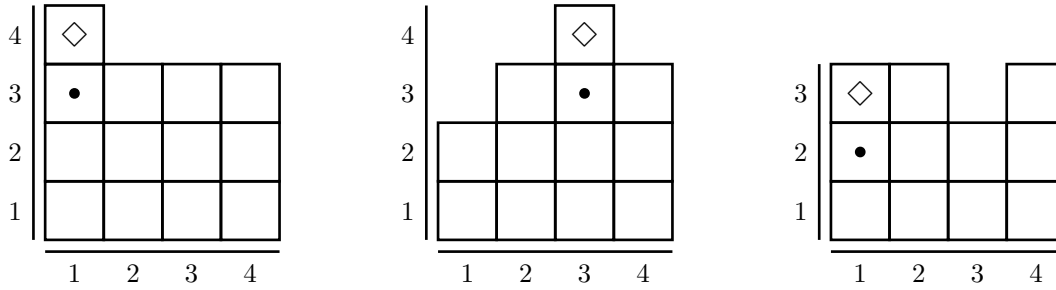


Figure 10: Beginning of a play simulating LPP

Example. In Figure 10, we give 3 steps of a realisation of the process described before, from the CRP point of view. For the LPP, starting from Figure 10a, we have 3 occurrences of every page (\diamond does not count in the occurrences). The cache is initially made up of page 2, 3, 4 and the algorithm has to choose which one of these pages it will evict to add page 1 instead, as it was requested by the adversary. The algorithm chooses to evict the page 3 in Figure 10b and an occurrence of page 1 is consumed.

The specificity of this problem needs some clarifications. An occurrence of a page is consumed when it arrives in the cache, which in our initial problem happens when \diamond is relocated from this stack, and not to this stack. And finally, when all occurrences of a page are consumed, the process is stopped.

Hence we call this problem the *Limited* Paging Problem (LPP), for which an instance is the sequence (n_1, \dots, n_W) , i.e. the number of occurrences of each page as well as an initial page $abs \in \llbracket 1, W \rrbracket$ which is not in the cache.

We show the following about the competitive ratio of the LPP problem.

Theorem 4.1 (LPP competitive ratio). *The competitive ratio of the LPP problem for deterministic algorithms is at least*

$$\min\left(2n - 1, (W - 1)\left(1 - \frac{1}{n}\right)\right)$$

with n being the number of occurrences of the page appearing the least.

Proof. See Section 6. Yet the idea behind this formula is that the algorithm has two main choices. Either it tries to minimise its cost, without considering the cost induced to OPT. In that case, it will alternate between the two pages having the least occurrences until one goes bankrupt. Otherwise, it tries to maximise the cost induced to OPT by alternating between all the pages. \square

This result is consistent with the litterature about the *Paging Problem*, since the ratio for deterministic algorithms for the Classic version is $W - 1$. Indeed, if we fix W and make all the $(n_i)_{1 \leq i \leq W}$ tend to infinity, we find the expected ratio $W - 1$.

The same result goes for our initial problem, hence this value is a lower bound on the competitive ratio of any deterministic online algorithm for the OCRP. Note that while we could adapt the paging optimal adversary for the *Paging Problem* to get a lower bound, it is not the case (at least not directly) for randomised algorithms. Indeed, the lower bound of $\log(W - 1)$ for the randomised competitive ratio of the *Paging Problem* is proved using an adversary that might do a lot of requests, which is not possible here since our pages have a limited number of occurrences.

Let us focus for a bit on a family of configurations where every stack has a same fixed height $h \leq H$. If W is big enough, the ratio becomes $2h - 1$ which is exactly the same as the one from Theorem 1.1.1. Which means *Leveling* is optimal for the competitive ratio for this kind of states. Yet, even if this seems very precise, actual ports have a very limited height for each of containers stacks, and a very large number of stacks.

In arbitrary families of configurations, it is harder to evaluate how good this bound is. Hence we tried to compute the exact bound for some small configurations, hoping it would give an idea about the behavior of the competitive ratio. At the moment, its asymptotic behavior is unknown, there is no clue if it is either bounded or not as well.

In Table 2 is given highest competitive ratio for certain values of N and W . For instance, with $N = 8$ and $W = 3$, the highest obtained lower bound is 2.5 and is reached with the configuration $(7, 1, 0)$.

Table 2: biggest comp. ratio obtained for certain values of N and W .

| $\frac{N}{W}$ | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---------------|---|---|---|---|---|-----|---|
| 3 | 1 | 2 | 2 | 2 | 2 | 2.5 | ? |
| 4 | 1 | 1 | 2 | 3 | 3 | ? | ? |
| 5 | 1 | 1 | 1 | 2 | ? | ? | ? |

5 Conclusion

Even though a review of the litterature of the CRP led us to study classic versions of the problem, we encountered several challenges. First, no tools from online algorithms have been effective, except

the link with the *Paging Problem* done in Section 4. Moreover, the problem has revealed being very hard to study from scratch, because of its chaotic behavior that can mostly be observed in Section 3.

We still managed to bring new theoretical guarantees (Theorem 2.2, Theorem 4.1) to this very practical area, and to a heavily used heuristic. Contact with people from the industry helped us understanding what was important in the models, which led us to consider different variants and try to generalise our results to wider frameworks, as Proposition 2.2.3.

The study we did could be extended to randomised algorithms, since we already know deterministic algorithms are not optimal, neither for the worst-case scenario nor for the competitive ratio.

Bibliography

- [1] M. Caserta, S. Schwarze, and S. Voß, “A mathematical formulation and complexity considerations for the blocks relocation problem,” *European Journal of Operational Research*, vol. 219, no. 1, pp. 96–104, 2012, doi: <https://doi.org/10.1016/j.ejor.2011.12.039>.
- [2] M. de Melo da Silva, G. Erdoğan, M. Battarra, and V. Strusevich, “The Block Retrieval Problem,” *European Journal of Operational Research*, vol. 265, no. 3, pp. 931–950, 2018, doi: <https://doi.org/10.1016/j.ejor.2017.08.048>.
- [3] M. Caserta, S. Voss, and M. Sniedovich, “Applying the corridor method to a blocks relocation problem,” *Operations Research-Spektrum*, vol. 33, pp. 915–929, 2011, doi: [10.1007/s00291-009-0176-5](https://doi.org/10.1007/s00291-009-0176-5).
- [4] B. Jin and S. Tanaka, “An exact algorithm for the unrestricted container relocation problem with new lower bounds and dominance rules,” *European Journal of Operational Research*, vol. 304, no. 2, pp. 494–514, 2023, doi: <https://doi.org/10.1016/j.ejor.2022.04.006>.
- [5] C. Lersteau and W. Shen, “A survey of optimization methods for Block Relocation and PreMarshalling Problems,” *Computers & Industrial Engineering*, vol. 172, p. 108529, 2022, doi: <https://doi.org/10.1016/j.cie.2022.108529>.
- [6] Y. Feng, D.-P. Song, D. Li, and Y. Xie, “Service fairness and value of customer information for the stochastic container relocation problem under flexible service policy,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 167, p. 102921, 2022, doi: <https://doi.org/10.1016/j.tre.2022.102921>.
- [7] V. Galle, V. H. Manshadi, S. B. Boroujeni, C. Barnhart, and P. Jaillet, “The Stochastic Container Relocation Problem,” *Transportation Science*, vol. 52, no. 5, pp. pp.1035–1058, 2018, Accessed: Aug. 01, 2025. [Online]. Available: <https://www.jstor.org/stable/48748022>
- [8] E. Zehendner, D. Feillet, and P. Jaillet, “An algorithm with performance guarantee for the Online Container Relocation Problem,” *European Journal of Operational Research*, vol. 259, no. 1, pp. 48–62, 2017, doi: <https://doi.org/10.1016/j.ejor.2016.09.011>.
- [9] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young, “Competitive paging algorithms,” *Journal of Algorithms*, vol. 12, no. 4, pp. 685–699, 1991, doi: [https://doi.org/10.1016/0196-6774\(91\)90041-V](https://doi.org/10.1016/0196-6774(91)90041-V).

6 Appendix

6.1 Proof of Theorem 2.2

Let us fix $W \geq 3$ and $H \geq 1$. We define a 2-player game on a bay of W stacks each of height H . We will refer to the algorithm as player 1, and to the adversary as player 2. The OCRP is equivalent to a finite game whose sets of states are finite and form a directed acyclic graph.

Definition 6.1.1 (states and notations).

- Q_2 , the set of states of player 2, is composed of every $s \in \llbracket 0, H \rrbracket^W$.
- For any $s \in Q_2$ and $1 \leq i \leq W$, we denote by s_i the i -th coordinate of s and $|s|$ the total number of containers in the configuration, i.e. $|s| = \sum_{i=1}^W s_i$.
- Q_1 , the set of states of player 1, is composed of every $s^{u,v} = (s, u, v) \in Q_2 \times \llbracket 1, W \rrbracket \times \llbracket 1, H \rrbracket$ such that $1 \leq v \leq s_u$ (player 2 must request a container, not an empty space) and $(W - 1) \cdot H \geq |s| - v$ (there must be enough space for player 1 to move the containers above the requested one).
- For any $s^{u,v} \in Q_1$, the number of containers player 1 will have to relocate is independant from its decision, this number is $s_u - v$ and is denoted by $\text{rel}(s, u, v)$.

Now we defined the states of the graph, we have to define the allowed moves in the game, i.e. the edges of the graph.

Definition 6.1.2 (retrieve - relocation).

- For $\llbracket 1, W \rrbracket$, we call e_i the vector composed of 0s and a 1 at position i . Hence, when $s_i < H$, $s + e_i$ is the state s with one more container on stack i .
- Similarly, if $s_i > 0$, $s - e_i$ is the configuration obtained by retrieving a container from stack i .
- For any $1 \leq i, j \leq W$ and $s \in Q_2$, if $s_i, s_j < H$ we say that $s + e_i$ relocates to $s + e_j$, which we denote by $s + e_i \xrightarrow{i,R,j} s + e_j$. Moreover, if $s_i > s_j + 1$ we can precise the notation with $s + e_i \xrightarrow{i,D,j} s + e_j$.

These help us define the so called transition of the game. Intuitively, player 2 just adds a request on a configuration and player 1 has to move the blocking containers, and retrieve the requested one.

Definition 6.1.3 (CRP transition). For any two states $s \in Q_2, t^{u,v} \in Q_1$, we have :

- $s \rightarrow t^{u,v}$ if and only if $s = t$ and $(W - 1) \cdot H \geq N + v$
- $t^{u,v} \rightarrow s$ if and only if $t - e_u \xrightarrow{u,R} s$.

Definition 6.1.4 (CRP Game). Given $N, H, W \in \mathbb{N}$, the CRP game G is defined as the tuple (Q_1, Q_2, \rightarrow) . We also denote by G_s the CRP game with initial state $s \in Q_2$.

Definition 6.1.5 (strategies). A strategy for player 1 is a mapping $\sigma : Q_1 \rightarrow Q_2$ such that for any $s^{u,v} \in Q_1$ we have $s^{u,v} \rightarrow \sigma(s^{u,v})$. respectively, a strategy for player 2 is a mapping $\pi : Q_2 \rightarrow Q_1$ such that for any $s \in Q_2, s \rightarrow \pi(s)$. We call Σ, Π the sets of strategies of player 1 and 2.

When both players fix a strategy and an initial state is given, a play is induced. The cost of this play is the number of relocations made by player 1. This play is necessarily finite and converges to the state with no containers, since any move from player 1 removes one container.

Definition 6.1.6 (play, value of a play). Given an initial state $s \in Q_2$ and two strategies $\sigma \in \Sigma, \pi \in \Pi$, a play $s \rightarrow s_1 \rightarrow \dots \rightarrow s_{2N}$ is induced. The cost of this play is $\sum_{k=0}^{N-1} \text{rel}(s_{2k+1})$. We call this value $\text{val}_{\sigma,\pi}(s)$.

Let us now start the analysis. First, we can suppose without loss of generality that the stacks are in decreasing order of height, to simplify both the proof and the visualisation. Moreover, let us suppose that *Leveling* maintains this order. This can be done by breaking the tie in the following manner : choose the rightmost stack as source of the relocations and leftmost stack as target of the relocation, between the tied stacks.

Since F always asks for the deepest container of stack 1 when considering the stacks in decreasing order, we can visualise the play as the first stack being emptied, hence being moved to the right (since it becomes the smallest stack) and the blocking containers are moved to the first $W - 1$ stacks without modifying their order.

The behavior of these strategies is regular, hence it is possible to compute the number of relocations induced by L and F on any state. We first give an intuition of what happens during the play which is converging to a certain form of state, where a stack is empty and all the other stacks have the same height, up to a difference of 1. This leads us to define the notion of regular state, and some other definitions to handle the side cases that can happen at the beginning of a play.

Definition 6.1.7 (class of states). A state s is said :

- *almost-full* if $|s| > (W - 1) \cdot H + 1$, i.e. when the opponent cannot ask for the deepest container.

- regular if it is not almost full.
- h -filled for $h \in \llbracket 0, H \rrbracket$ if it has a stack of size h and all the other stacks are full. A h -filled state is always almost full, whatever the value of h .
- Leveled if $s_W = 0$, and for any $i, j \in \llbracket 1, W-1 \rrbracket$, $|s_i - s_j| \leq 1$. A leveled state is regular.

When L and F are playing, the state converges quickly to a leveled state, which is unique up to a permutation of the stacks. Indeed, if s is leveled for $i \in \llbracket 1, W \rrbracket$, $s_i = \lceil \frac{n-i+1}{W-1} \rceil$. Imagine a line on each stack at height $\frac{n-i+1}{W-1}$, *Leveling* never relocates a container above this line. This result is given by the following lemma:

Lemma 6.1.8. *For any $s \in Q_2$ which is regular, let $t = L(F(s))n$ and let (x, y) be coordinates in the state t (it is important because the stacks are considered in decreasing order). If L relocates a container to (x, y) , then every container below and/or on its left is filled, for a total of at least $(W-1)(y-1) + x$ containers. Which implies that $y \leq \lceil \frac{n-x}{W-1} \rceil$.*

Proof. The first statement of the lemma directly follows from the definition of L . Then, supposing a relocation is made on (x, y) , since every container from stack 1 has been relocated, we have $\sum_{i=1}^{W-1} t_i = n-1 \geq (W-1)(y-1) + x$. So $y \leq \frac{n-1-x}{W-1} + 1$ which is, since y is an integer, equivalent to $y \leq \lceil \frac{n-x}{W-1} \rceil$. \square

Thanks to this lemma, we can tackle the theorem for the different classes of states.

Proposition 6.1.9. *Let $s \in Q_2$ be a regular state.*

$$\text{val}_{L,F}(s) = s_1 + \sum_{i=2}^W \max\left(s_i, \left\lceil \frac{n-i+1}{W-1} \right\rceil\right) + \sum_{i=1}^{n-W} \left\lceil \frac{i}{W-1} \right\rceil$$

Proof. By induction on n . If $n = 0$, then $s_i = 0$ for every $i \in \llbracket 1, W \rrbracket$, thus the above formula is equal to 0 which is the number of needed relocations.

Else, since the state is not almost-full, the request of F is on the first container of the first stack, so $F(s) = s^{1,1}$. The cost of the first move is s_1 , let us note t the state $L(F(s))$, we have by induction

$$\text{val}_{L,F}(s) = s_1 + t_1 + \sum_{i=2}^W \max\left(t_i, \left\lceil \frac{n-i}{W-1} \right\rceil\right) + \sum_{i=1}^{n-W-1} \left\lceil \frac{i}{W-1} \right\rceil$$

We will prove the following :

- (1) $t_1 = \max(s_2, \lceil \frac{n-1}{W-1} \rceil)$
- (2) For all $i \in \llbracket 2, W-1 \rrbracket$, $\max(t_i, \lceil \frac{n-i}{W-1} \rceil) = \max(s_{i+1}, \lceil \frac{n-i}{W-1} \rceil)$
- (3) $t_W = 0$

The equation (3) is trivial since a stack has been emptied.

For equation (1), either $s_2 \geq \lceil \frac{n-1}{W-1} \rceil$, in which case it cannot receive any new container. Indeed, the stack s_2 becomes the stack t_1 (recall the visualisation, stack s_1 is emptied and becomes t_W , and every other stack is shifted on the left). So, by Lemma 6.1.8, a relocation on $(1, y)$ where $y > \lceil \frac{n-1}{W-1} \rceil$ is impossible, thus $t_1 = s_2$. Else, for the same reason, the stack can not grow above this value, so $t_1 \leq \lceil \frac{n-1}{W-1} \rceil$. Yet we have $n-1 = \sum_{i=1}^{W-1} t_i \leq t_1 \cdot (W-1)$, hence $t_1 \geq \frac{n-1}{W-1}$, and since t_1 is an integer, the same holds with the upper rounding, which gives us that $t_1 = \lceil \frac{n-1}{W-1} \rceil$.

For the equation (2), the reasoning is similar. Either $s_{i+1} \geq \lceil \frac{n-i}{W-1} \rceil$ and we get $t_i = s_{i+1}$. Else, $t_i = \lceil \frac{n-i}{W-1} \rceil$ and we get $\max(t_i, \lceil \frac{n-i}{W-1} \rceil) = \lceil \frac{n-i}{W-1} \rceil = \max(s_{i+1}, \lceil \frac{n-i}{W-1} \rceil)$.

By using these 3 equations, we get :

$$\begin{aligned}
\text{val}_{L,F}(s) &= s_1 + t_1 + \sum_{i=2}^W \max\left(t_i, \left\lceil \frac{n-i}{W-1} \right\rceil\right) + \sum_{i=1}^{n-W-1} \left\lceil \frac{i}{W-1} \right\rceil \\
&= s_1 + \max\left(s_2, \left\lceil \frac{n-1}{W-1} \right\rceil\right) + \sum_{i=2}^{W-1} \max\left(s_{i+1}, \left\lceil \frac{n-i}{W-1} \right\rceil\right) + \left\lceil \frac{n-W}{W-1} \right\rceil + \sum_{i=1}^{n-W-1} \left\lceil \frac{i}{W-1} \right\rceil \\
&= s_1 + \sum_{i=2}^W \max\left(s_i, \left\lceil \frac{n-i+1}{W-1} \right\rceil\right) + \sum_{i=1}^{n-W} \left\lceil \frac{i}{W-1} \right\rceil
\end{aligned}$$

□

Proposition 6.1.10. *Let $s \in Q_2$ and $h > 0$ such that s is h -filled. We have*

$$\text{val}_{L,F}(s) = \sum_{i=0}^{h-1} (H-i) + (W-1) \frac{H(H+1)}{2}$$

Proof. By induction on n , or equivalently on h .

- If $h = 1$, then by definition, the state s is equal to $(H, H, \dots, H, 1)$. So F requests the bottom container of the first stack and $t = L(F(s)) = (H, H, \dots, H, 0)$. Therefore, t is leveled (hence regular) and $|t| = (W-1)H$ so we get

$$\begin{aligned}
\text{val}_{L,F}(s) &= H + \text{val}_{L,F}(t) \\
&= H + H + \sum_{i=2}^W \max\left(H, \left\lceil \frac{(W-1)H-i+1}{W-1} \right\rceil\right) + \sum_{i=1}^{(W-1)H-W} \left\lceil \frac{i}{W-1} \right\rceil \\
&= H + \sum_{i=1}^{(W-1)H} \left\lceil \frac{i}{W-1} \right\rceil \\
&= H + (W-1) \frac{H(H+1)}{2}
\end{aligned}$$

- Else, $s = (H, H, \dots, H, h)$, so we have $t = (H, H, \dots, H, h-1)$ which is $(h-1)$ -filled. By induction hypothesis, we have

$$\begin{aligned}
\text{val}_{L,F}(s) &= (H-h+1) + \text{val}_{L,F}(t) \\
&= (H-h+1) + \sum_{i=0}^{h-2} (H-i) + (W-1) \frac{H(H+1)}{2} \\
&= \sum_{i=0}^{h-1} (H-i) + (W-1) \frac{H(H+1)}{2}
\end{aligned}$$

□

Proposition 6.1.11. *Let $s \in Q_2$ such that s is almost-full. Let $h = n-1 - (W-1)H$, we have*

$$\text{val}_{L,F}(s) = s_1 + \frac{h(2H-h+1)}{2} + (W-1) \frac{H(H+1)}{2}$$

Proof. It follows immediately from the fact that the first move will have a cost of $s_1 - n + (W - 1)H + 1 = s_1 - h$ and that the state $L(F(s))$ is h -filled. \square

We have not proved that this value is the value of the game because nothing says apriori that L and F are the optimal strategies for both players. To do so, we can prove that player 1 can ensure the value of the play to be at most $\text{val}_{L,F}(s)$ and that player 2 can ensure it is at least this amount. The core of the proof is that leveled configurations are better for L when playing against F .

Proposition 6.1.12. *Let $s, t \in Q_2$ such that $s \xrightarrow{D} t$, we have $\text{val}_{L,F}(t) + 1 \geq \text{val}_{L,F}(s) \geq \text{val}_{L,F}(t)$.*

Proof. Let us do a case analysis on the classes of states s and t .

- If s is almost-full, then by definition t is almost-full as well. Since they have the same size, by Proposition 6.1.11, $\text{val}_{L,F}(s) - \text{val}_{L,F}(t) = s_1 - t_1 \in \{0, 1\}$ since s and t are the same configurations up to one container.
- Otherwise, t is not almost-full as well, hence we are in the case of Proposition 6.1.9. Let $i < j$ be the indices such that the relocated container from s to t is relocated from stack i to stack j . Without loss of generality, we can suppose that i is the rightmost stack of its height, and j is the leftmost stack of its height. This gives us the property that the decreasing order of stacks in the same in s and t . Notice that if $i = 1$ the property is trivial by the formulae. If $i \geq 2$, we have

$$\begin{aligned} \text{val}_{L,F}(s) - \text{val}_{L,F}(t) &= \overbrace{\max\left(s_i, \left\lceil \frac{n-i+1}{W-1} \right\rceil\right) - \max\left(s_i - 1, \left\lceil \frac{n-i+1}{W-1} \right\rceil\right)}^{\Delta_i} \\ &\quad - \underbrace{\left[\max\left(s_j + 1, \left\lceil \frac{n-j+1}{W-1} \right\rceil\right) - \max\left(s_j, \left\lceil \frac{n-j+1}{W-1} \right\rceil\right)\right]}_{\Delta_j} \\ &= \delta_{s_i > \lceil \frac{n-i+1}{W-1} \rceil} - \delta_{s_j \geq \lceil \frac{n-j+1}{W-1} \rceil} \end{aligned}$$

It suffices to observe that $s_i \leq \lceil \frac{n-i+1}{W-1} \rceil$ and $s_j \geq \lceil \frac{n-j+1}{W-1} \rceil$ cannot be true simultaneously. Indeed, if $s_j \geq \lceil \frac{n-j+1}{W-1} \rceil$, we have $\lceil \frac{n-i+1}{W-1} \rceil \leq \lceil \frac{n-j+1}{W-1} \rceil + 1 \leq s_j + 1 < s_i$. The first inequality is a consequence of $j \leq i + (W - 1)$ and the last one is the definition of \xrightarrow{D} . \square

Now, to prove the optimality of L and F , it remains to show that these are optimal one against the other.

Proposition 6.1.13. *For any $s \in Q_2$ and $\sigma \in \Sigma$, $\text{val}_{L,F}(s) \leq \text{val}_{\sigma,F}(s)$.*

Proof. Let us do an induction on $|s|$.

If $|s| = 0$, the property is trivial. Otherwise, let u, v be the request from F on s . Now let us take a look at $L(s^{u,v})$ and $\sigma(s^{u,v})$. By definition of L , it relocates the containers to the lowest stacks at each step, hence we have $\sigma(s^{u,v}) \xrightarrow{D}^* L(s^{u,v})$. More formally :

- We need to see the states as a set of coordinates, hence we introduce the order \preceq on coordinates, such that $(x, y) \prec (x', y')$ if and only if $y < y'$ or $(y = y' \text{ and } x < x')$. By definition, L choose to relocate containers on the smallest free coordinate for the order \prec . Hence, defining $t = s - k \cdot e_u$ with $k = \text{rel}(s, u, v)$, i.e. the state s with every blocking containers removed, we have $L(s^{u,v}) = t \cup A$ where A is the set of k smallest coordinates for the order \preceq that are free in the configuration t . Yet $\sigma(s^{u,v}) = t \cup B$ where B is a set of k free coordinates in t . Thus we have a bijection $r : A \rightarrow B$ such that $r((u, v)) \preceq (u, v)$. This gives a sequence of relocations that are only downward that goes from $L(s^{u,v})$ to $\sigma(s^{u,v})$.

By induction hypothesis,

$$\begin{aligned}
\text{val}_{\sigma,F}(s) &= k + \text{val}_{\sigma,F}(\sigma(s^{u,v})) \\
&\geq k + \text{val}_{L,F}(\sigma(s^{u,v})) \\
&\geq k + \text{val}_{L,F}(L(s^{u,v})) \\
&= \text{val}_{L,F}(s)
\end{aligned}$$

Which concludes the proof. \square

Lemma 6.1.14. *For any $s \in Q_2$ and $1 \leq r \leq H$ and $1 \leq i, j \leq W$, if $s_i > s_j$, defining $\Delta = s_i - s_j$, we have*

$$L(s^{i,r}) \xrightarrow[R]{\Delta} L(s^{j,r})$$

Proof. By induction on Δ . If $\Delta = 0$ then $s^{i,r}$ and $s^{j,r}$ are the same state up to a permutation, hence they are still the same after the move from L . Otherwise, suppose $s_i - s_j = \Delta + 1$, let t be the state obtained after relocating one container from s_i according to the strategy L , hence there exists u such that $s' = s - e_i + e_u$, and we have $L(s^{i,r}) = L(t^{i,r})$. By induction hypothesis, $L(t^{i,r}) \xrightarrow[R]{\Delta} L(t^{j,r})$ and by definition of t we have $L(t^{j,r}) \xrightarrow[R]{1} L(s^{j,r})$ hence we finally get that $L(s^{i,r}) \xrightarrow[R]{\Delta+1} L(s^{j,r})$ which concludes the proof. \square

Proposition 6.1.15. *For any $s \in Q_2$ and $\pi \in \Pi$, $\text{val}_{L,\pi}(s) \leq \text{val}_{L,F}(s)$.*

Proof. Let us do an induction on $|s|$.

If $|s| = 0$ the property is trivial. Else, let u, v be the request of π on s , i.e. $\pi(s) = s^{u,v}$. Let us now consider a strategy π_F that plays the same move as π for the first request and then plays as F . By induction hypothesis, this strategy is at least better than π .

If it is possible (if $v > 1$), consider the strategy π'_F that requests the container $u, v-1$ and then plays F . We have $L(s^{u,v}) \xrightarrow[R]{1} L(s^{u,v-1})$ hence by Proposition 6.1.11 we have $\text{val}_{L,F}(s^{u,v}) - 1 \leq \text{val}_{L,F}(s^{u,v-1})$. Hence, by noting $k = \text{rel}(s, u, v)$, we get:

$$\begin{aligned}
\text{val}_{L,\pi}(s) &\leq \text{val}_{L,\pi_F}(s) \\
&= k + \text{val}_{L,F}(L(s^{u,v})) \\
&\leq (k+1) + \text{val}_{L,F}(L(s^{u,v-1})) \\
&= \text{val}_{L,\pi'_F}(s)
\end{aligned}$$

Which means that the player 2 must play as deep as possible in the stack he is considering and then play F . Now it remains to prove he has interest in the biggest stack.

Let us consider two strategies π_i and π_j such that $s_i > s_j$. Each of them playing as deep as possible in their stack and then play as F . By using Lemma 6.1.14 we have $L(\pi_i(s)) \xrightarrow[R]{\Delta} L(\pi_j(s))$ where $\Delta = s_i - s_j$. Thus, by proposition Proposition 6.1.11 we have $\text{val}_{L,F}(L(\pi_i(s))) \geq \text{val}_{L,F}(L(\pi_j(s))) - \Delta$. So, denoting k the number of relocated constainers in the case of π_j :

$$\begin{aligned}
\text{val}_{L,\pi_i}(s) &= \Delta + k + \text{val}_{L,F}(L(\pi_i(s))) \\
&\geq \Delta + k + \text{val}_{L,F}(L(\pi_j(s))) - \Delta \\
&= \text{val}_{L,\pi_j}(s)
\end{aligned}$$

It means that player 2 must play as deep as possible in the biggest stack, which is by definition the strategy F , hence it is optimal against L . \square

Corollary 6.1.16. *Given $s \in Q_2$, G_s is determined and its value is*

$$\text{val}_{L,F}(s) = \min_{\sigma \in \Sigma} \max_{\pi \in \Pi} \text{val}_{\sigma,\pi}(s) = \max_{\pi \in \Pi} \min_{\sigma \in \Sigma} \text{val}_{\sigma,\pi}(s)$$

6.2 Sketches of proof of Proposition 2.2.1

Let us not go into the details of the proofs, which would be as long as the one of Theorem 2.2 and don't add anything theoretically interesting.

For the UCRP, the problem is easier for the algorithm since new moves are allowed, so we know the value of the game is at most $\text{val}_{L,F}(s)$ for a configuration s . It only remains to prove it is also a lower bound. It suffices to observe that during a phase of the algorithm, during which it :

- relocates blocking containers
- relocates non blocking containers
- retrieves the requested container

every move can be exchanged without modifying anything.

We fix the adversary strategy as being F , and consider an a strategy $\sigma \in \Sigma^+$, the set of strategies augmented with unrestricted moves.

We do an induction on $|s|$, the base case being trivial again. For an arbitrary state s , we use the commutativity of moves done by σ and start with the retrieval. The obtained state t has one less container, hence its value is, by induction hypothesis $\text{val}_{L,F}(t)$. From here, it is easy to see σ has no interest in doing unrestricted moves, because they cost 1 and will reduce the future cost by at most 1 by Proposition 6.1.12. Hence, since unrestricted moves are not worth it, *Leveling* is optimal against F by the argument from the initial proof.

Hence, from a state s and a stock of k containers to add, the value of the game is $\text{val}_{L,F}(t)$ where t is the state s plus the k containers added to the lowest positions one by one.

For the DCRP, one needs first to define its online variant. Since no information is known, it is up to the adversary to choose when to add containers, but for the sake of having the game well-structured, the numbers of added containers is fixed by the adversary.

Then, it is easy to prove the optimality of *Leveling* against a strategy F that also add containers at arbitrary steps by an induction on $(k, |s|)$ for the lexicographic order with k being the number of containers the adversary can add in the bay. The induction is exactly the same as before.

Then, supposing the player 1 uses *Leveling*, we fix the k steps in which the player 2 adds containers, and we prove with a similar induction that F is optimal.

Finally, it remains to look for the best moments for player 2 to add containers. It is indeed the very beginning of the game, because when a container is added, *Leveling* places it at the lowest position and it will remain there until the end. Thus there is absolutely no reason to wait before placing it.

The way we change a proof from CRP to UCRP can be applied to DCRP to obtain the same value for UDCRP.

6.3 Sketch of proof of Proposition 2.2.3

This proof is almost the same as Theorem 2.2. One can do a projection of the structures $(V_i, E_i), C_i$ on the set of natural numbers by considering $|C_i|$, i.e. the number of containers inside the structure.

Once this is done, the proof is exactly the same except when we prove player 2 has to play as deep as possible. Instead of proving the strategy π'_F playing 1 container below is better than π_F , we have to directly consider the one playing the deepest container (because the move from π'_F might not be allowed by the structure).

6.4 Proof of Theorem 4.1

Note : This proof is not totally finished. This gives a good idea about how we can prove the theorem, but some flaws remain in some places, like the number of phases, which may not reach n_a in some cases, but the theorem is true up to some tweaks in the values.

Let us consider an instance of W stacks with n_i occurrences of page i . Since the adversary is fixed, always asking for the unique page not being in the cache, the behavior of the algorithm is fully determined by its sequence of requests.

Let $w = w_1 \dots w_m$ be such a finite sequence, notice that the problem is exactly the same as the classic *Paging Problem* from the offline optimal point of view. Indeed, the same offline algorithm is optimal, always evicting the page that will be requested in the maximal ammount of time. Hence we divide the sequence into phases, that we define by marking the pages in the following way : Initially, no page is marked, and when an unmarked page is requested, it gets a mark. If after a request every page is marked, then this request is the start of a new phase, so just before it happens, we remove the mark from every page.

The cost for the offline optimal algorithm is exactly the number of phases, since it will, at start of each phase, evict the first page of the next phase so the cost induced in the remaining part of the current phase is zero. Now let us deduce, depending on the number of phases, a lower bound on the cost of the online algorithm, i.e. m , the length of the sequence.

Let k be the number of phases, every phase except the last one is said complete, and a complete phase is of length at least $W - 1$. This already says that $m \geq (k - 1)(W - 1)$, but we will be able to get a better bound.

- If $k = 1$, there is a single phase which is incomplete. This phase still can't be arbitrarily short since page a has to run out of stock. Moreover, a page cannot be requested twice in a row, so we get that $m \geq 2n_a - 1$.
- Else, let us fix k and a and modify the sequence to minimise its length. First, for each page different than a , we can make them appear at most once by phase by removing the one appearing twice or more. For the page a , the total number of occurrences is exactly n_a , thus, denoting by $l \leq k$ the number of phases the page a appears in, the new sequence has a length of $(k - 1) \cdot (W - 1) + n_a - l$. It means that the algorithm wants to disperse the requests of page a in different phases.
 - If $k \leq n_a$, a request of a can be inserted in all the phases, having $l = k$. Hence the minimal cost is $(k - 1) \cdot (W - 1) + n_a - k$.
 - If $k > n_a$, a request can be inserted in at most n_a phases, including the last phase, since the last request is one for the page a . We get a minimal cost of $(k - 1) \cdot (W - 1)$

In the end, the algorithm has the choice between different outputs. It can try to finish the game as soon as possible, getting a ratio of $2n_a - 1$. It can maximise the cost of OPT, and get one of the ratio among $\frac{(k-1) \cdot (W-1) + n_a - k}{k}$ for $k \in \llbracket 2, n_a \rrbracket$ or $\frac{k-1}{k} \cdot (W - 1)$ for $k \geq n_a$.

The case $k > n_a$ is not useful for the algorithm since it is strictly worse than the case $k = n_a$. The value $\frac{(k-1) \cdot (W-1) + n_a - k}{k}$ can be rewritten as $W - 2 + \frac{n_a - W + 1}{k}$ which is strictly decreasing with k . Thus, the algorithm only has interest in the cases $k \in \{1, n_a\}$, which gives the following lower bound on the competitive ratio

$$\min\left(2n_a - 1, (W - 1) \cdot \left(1 - \frac{1}{n_a}\right)\right)$$

To get the desired formula, it just remains to take the minimum among every page a .