

Internship Report

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. It is why a lot of different operations that are frequently done in these terminals are heavily being studied. This goes from yarn crane scheduling, in order to optimise the movements of cranes, to the stacking of containers in a way that maximises space utilisation. 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, for example the Container Pre-marshalling Problem (CPMP) 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 I 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. 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 tackle it. Some of them solve the problem exactly, such as TODO (talk ILP, Branch and bound, corridor, etc.. + survey of the literature). While others focus on heuristics and metaheuristics, such as TODO (talk about the different heuristics, metaheuristics, etc.. + survey of the literature).

While most of the litterature study the CRP in a context with full information, it is relevant to assume only partial information is available, whatever the shape this information takes. TODO (talk about the different types of information, and how they can be used to solve the problem, lot of examples in).

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, who encountered this

kind of problem in practice in an uranium barrel storage facility. Yet the problem revealed itself to be way more complex than expected, and we could not get satisfying results about the predictions models we came with.

Since the problem was too complex for this kind of models, and the literature was poor in terms of theoretical results (thus in terms of tools to study the problem), we went back to the online variant of the CRP, which had already a few results.

I managed to show that the Leveling heuristic is optimal in the worst-case setting, and some extensions to this theorem. I naturally went to study the case of a random uniform adversary, which is a common model when studying online algorithms. This has been the main focus of my internship, and unfortunately, it only led to a conjecture, which was already stated in [2], 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 are not very useful in practice. It mostly gives theoretical insights about the Leveling heuristic, which is already used, and starts the way to narrow the study of the online CRP optimal competitive ratio.

0.4 Arguments supporting its validity

Outside of the results that have been proven, i.e. the optimality of Leveling in the worst-case setting, and the lower bound on the competitive ratio, I have ran several experiments, mostly to comfort the validity of the conjecture about the average-case optimality of Leveling. Some are also about arguments of 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 let a lot of doors open, as the conjecture about the average-case optimality of Leveling is still open 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.

The first section of this report will present the Container Relocation Problem, its online variant and will go over the results from [3] 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 try to prove the conjecture Conjecture 3.5, so our main tentative 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 of maximal capacity H (we might also refer to the capacity as the 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 the containers in the sequence of requests.

Example. In Figure 1, 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. Noting that these 3 relocations were non avoidable, this strategy is optimal with a total of 3 relocations.

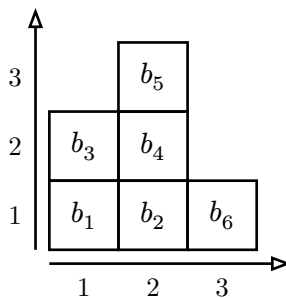


Figure 1: The optimal solution for an instance of the CRP

The CRP as it is rely on assumptions that are for most 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 solution with fewer relocations, but is harder to solve due to the increased number of dimensions in the search space, yet TODO says that the solution are usually close. If A1 does not hold, the problem is called the Dynamic Container Relocation Problem (DCRP). Whether A1 and A3 hold or not, the problem is NP-hard, as shown by [1] using a reduction from the Mutual Exclusion Scheduling Problem (MES) with permutation graphs.

Assuming A2 may be too unrealistic in practice, thus it is often relaxed by different approaches, as for example by [2], 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 [3], 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 will 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 do as good 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 how good performs 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 done if it had the information in the beginning. An important result shown in [3] 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 $2\lceil \frac{N}{W} \rceil - 1$*

Note that the theorem does not state that L is c_L -competitive, hence this value is tight. Yet it would still be possible to find a better competitive ratio considering different parameters than N and W .

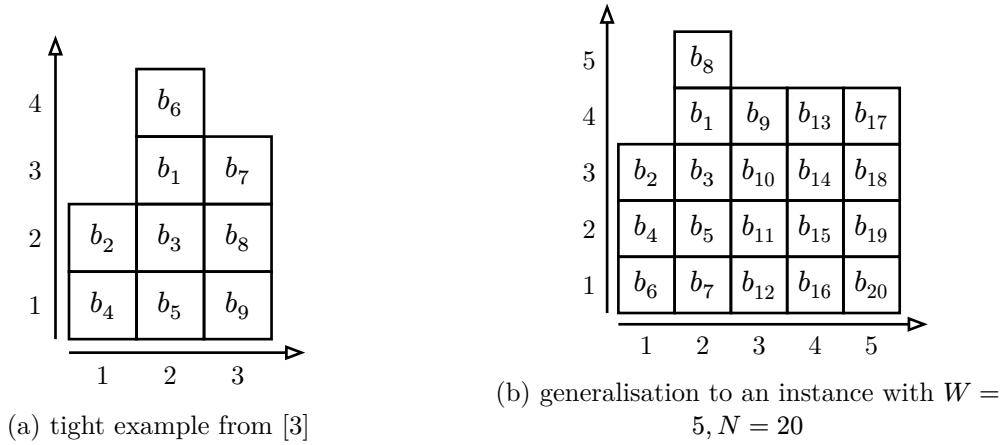
The tightness of this ratio is obtained in [3] with the example given in Figure 2a, for which $\text{cost}_L(I) = 5 = 2 \cdot \lceil \frac{9}{3} \rceil - 1$ and $\text{cost}_{\text{OPT}}(I) = 1$, because the algorithm can just move b_6 on b_7 and from there, no relocation will be needed. Whereas L will 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 2b. 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 null. On the other side, Leveling has to relocate this same container c_L times, which concludes the proof. \square



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

Let us look back at the Theorem 1.1.1, this ratio is not very satisfying as Leveling seems to be practically good in practice and to have a very small ratio for most instances. We need to look for another 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 average case.

2 Worst case analysis of Leveling

Before going into some details, let us notice that accounting a cost of 1 when the algorithm retrieve the container don't change anything to the worst-case analysis nor the average case analysis later in the report, since it only modifies the total cost of the process by the constant N . Yet this can't be done when using the competitive ratio, so from now on and until we start using the competitive ratio again in Section 4, we account a cost of 1 for the action of retrieving.

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 memories of previous configurations). Let us note by S the set of configurations of the OCRP, and by $S_{N,W,H}$ the set of configurations with N containers, W stacks and height H .

Given $s \in S_{N,W,H}$, 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 . 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 $s \in S_{N,W,H}$, 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 theory 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 p_L be a permutation that requests the deepest container of a biggest stack against Leveling. For any configuration $s \in S_{N,W,H}$, denoting, $s_1 \geq s_2 \geq \dots \geq s_W$ its stacks heights in decreasing order, we have :

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

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 the Section 6.

Since the configurations don't have any request attached to it, 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 $s \in S_{N,W,H}$ is a vector (h_1, h_2, \dots, h_W) such that $\forall i, h_i \in \llbracket 0, H \rrbracket$, $\sum_{i=1}^W h_i = N$ and $h_1 \geq h_2 \geq \dots \geq h_W$.

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 $s, s' \in S_{N,W,H}$, we say that s downs to s' which we denote by $s \xrightarrow{D} s'$ if s' 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 $s \xrightarrow{DL} s'$

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 plays 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 an imaginary line, of height $\lceil \frac{n-i+1}{W-1} \rceil$, which can be seen in TODO. 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 $s, s' \in S_{N,W,H}$, if $s \xrightarrow{D} s'$, then $\text{cost}_L(t^{p_L}) + 1 \geq \text{cost}_L(s^{p_L}) \geq \text{cost}_L(t^{p_L})$.*

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 mentioned 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 with the following modified value : TODO*

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.

Our first attempt was to give the adversary a function $f : \mathbb{N} \rightarrow \mathbb{N}$ which tells him how much containers he can request at once, so that its set of possible request would be $\llbracket 0, f(n) \rrbracket$ 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, yet we give a counter example that shows Leveling is not optimal in that setting. Let us imagine the containers are placed in a double access queue, so when the adversary requests a container, the algorithm can choose on which side he will retrieve the containers. It is equivalent (up to some restrictions, for example that the algorithm cannot relocate a container on the other side of the same queue) to the above model with the function $f : n \mapsto \max(0, \lfloor \frac{n-1}{2} \rfloor)$.

Let us consider the state $(5, 2, 0)$ with a request at bottom of stack 1, after the move from Leveling, we reach the configuration $(3, 3, 0)$ which induces a cost of 2, yet the configuration $(4, 2, 0)$ was reachable with an induced cost of 1.

It alone does not say much because Leveling may just be a particular case of some more clever heuristic in this setting. Yet several variants of Leveling have been tried without success, as for example 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 will 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 in the graphs, they are given by sets $C_i \subseteq V_i$, which must verifies $\sum_{i=1}^W |C_i| = N$. The graph edges is 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.

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 vertex with no predecessor, then Theorem 2.2 holds.*

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 heuristic Leveling-avoid heuristic, which plays as Leveling but marks a stack it will avoids 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.

The proof of potential optimality of this heuristic is still to be done.

3 Random uniform adversary

Both competitive ratio and worst-case analysis suffer the fact that in practice, the requests may not be adversarial. Hence one may prefer to see how good does a heuristic on a random case. Several experiments already have been done to evaluate the performances of Leveling on random cases, such as in [3] and [2]. 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 $S_{N,W}$ denotes the set of configurations composed of N containers on W stacks.

For a configuration $s \in S_{N,W}$, 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 $t \in S_{N,W}$. 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 $s \in S_{N,W}$ 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 $s \in S_{N,W}$ 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 $s, t \in S_{N,W}$ such that $s \xrightarrow{D} t$. We have $\text{pot}(s) > \text{pot}(t)$*

Proof. Immediate. □

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 $s \in S_{N,W,H}$ 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 $s \in S_{N,W}$ 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 $\text{avg}_A(s) = 2N! \cdot \text{avg}_A^p(s)$ which verifies :

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

Everything is defined to state the conjecture, that was already given in a different form in [2]:

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

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

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 $s, t \in S_{N,W}$ such that $s \xrightarrow{DL} t$, we have $\text{avg}_L(s) \geq \text{avg}_L(t)$*

Proof. Let us suppose that the conjecture is correct, and that there exists $s, t \in S_{N,W}$ 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 $s, t \in S_{N,W}$ 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 true for some $N \in \mathbb{N}$, and let $s \in S_{N+1,W}$. We have for all $A \in \mathcal{A}$

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

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 configuration, 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 $s, t \in S_{N,W}$ 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 $s, t \in S_{N,W}$ such that $s \xrightarrow{D} t$:

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

The term $\Delta_{\text{pot}}(s, t)$ is positive, and $\Delta_{\text{fut}}(s, t)$ can be decomposed into N differences, 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.

The terms with requests on stacks u and v for both s and t remains, and we are now free to pair them the way it fits better. Yet some of these pairings might be in the wrong direction : if $s = (6, 4, 0)$ and $t = (5, 5, 0)$, 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 non avoidable, even if we can, in specific scenarios, get something from it, as stated by the next proposition :

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

Proof. Let us notice that if v is the smallest stack of t , then 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 state 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 so that $\text{avg}_L(s) \geq \text{avg}_L(t)$. \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 “currency” 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 $s, t \in S_{N,W}$ such that $s \xrightarrow{D} t$, supposing 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 , either 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 sequence of permutation.

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 margin of error for $\Delta_{\text{pot}}(s, t)$ that is allowed 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.

TODO

Moreover, note that this example uses 3 stacks, and its stacks different from u and v (i.e. the 3rd stack) are empty. This means that we didn't lost any error margin when using the induction hypothesis on these stacks.

To finish this section, we present some experimental results that comfort this conjecture.

For some values of W , we computationally checked the optimality of Leveling for every configurations having less than some number $M(W)$ that is given in the TODO. 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. We don't use LINC because it doesn't reduce the numbers of configurations to check, even if it is weaker.

A last interesting observation is that in the very specific case of $W = 3$ which is already a theoretical issue, the value $\Delta_{\text{fut}}(s, t)$ 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 treat, but it has been a dead-end as well. This small conjecture has been checked for every configuration until $N = \text{TODO}$.

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.

Even though the competitive ratio of Leveling has been determined in [3], it remains open, and moreover stated as open in the same paper, that no lower bound on the competitive ratio was known.

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 way 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 the 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 will ask to move a chosen container that we will call \square by requesting the one just below. When A will choose where to relocate it, we will ask to move it again, etc... This process is exactly what happens in the example giving the tightness of Theorem 1.1.1 in [3], 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. 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. The algorithm is given a cache of size $W - 1$, which contains every page on which \square is not. When we request the container below \square , it is as if we were asking for a page, still playing the role of an adversary. When the algorithm relocates \square , it chooses which page it evicts from the cache to replace it by the one we requested.

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 \square 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 which is not in the cache, $abs \in \llbracket 1, W \rrbracket$.

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

Note, I am still writing the proof, some issues are in there and the result might have to be tweaked a bit, or maybe it will be an asymptotic ratio

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

$$\min \left(\min_{1 \leq i \leq W} 2n_i - 1, W - 1 \right)$$

Proof. See Section 6 □

This result is coherent with the literature about the paging problem, since the ratio for deterministic algorithms for the Classic version is $W - 1$. The intuition here is that the algorithm can make use of the limited number of pages to stop the process early and prevent the adversary to reach the $W - 1$ ratio.

Corollary 4.2. *The competitive ratio for deterministic algorithms for the OCRP is at least $W - 1$ if seen as a function of N, W . For a fixed configuration $s \in S_{N,W,H}$, we have that it is at least $\min(\min_{1 \leq i \leq W} 2s_i - 1, W - 1)$*

Even though this ratio is not optimal in general, if we reduce to a family of configurations for which the number of stacks is very big and no stack is lower than the others i.e. $\min_{1 \leq i \leq W} s_i = \Omega(\frac{N}{W})$ and $W = \Omega(\sqrt{N})$, this ratio is optimal. Indeed, for that kind of configurations, we have $\min(\min_{1 \leq i \leq W} 2s_i - 1, W - 1) = \Omega(\frac{N}{W})$ which is the same order than the competitive ratio of the Leveling heuristic. Moreover, the condition $W = \Omega(\sqrt{N})$ is totally natural since the height of stacks in port terminals is often very limited. Unfortunately, the other condition is quite restrictive since only one low stack is sufficient to break the bound.

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 TODO can be found the table of exact competitive ratios for deterministic algorithms for some configurations.

5 Conclusion

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] 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>
- [3] 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>.

6 Appendix