ONLINE TIMELY BIN PACKING

Master Thesis for Computing Science

Zhadyra Khattar
0033324
z.khattar@students.uu.nl

August 2023

1st supervisor: Dr. H. H. (Alison) Liu
2nd supervisor: Dr. H. Bodlaender

Department of Information and Computing Sciences
Faculty of Natural Sciences

Universiteit Utrecht

Contents

Abstract

Within the traditional online bin packing problem context, items with a positive size not greater than one are sequentially introduced. These items are assigned to bins with a capacity of 1 unit each, ensuring that the cumulative size of items placed in a bin remains within its capacity limit; the goal is to use the fewest bins possible. This thesis investigates novel configurations within the domain of online bin packing, Online Timely Bin Packing Problem (T-BPP). We introduce a new concept of *time slots*, whereby each item with a positive size not larger than one is characterised by a *release time* and an associated *deadline*. Item is *available* for packing at time slot t if it is released before t and its deadline is after t. At time slot t, an online algorithm is free to allocate available items in a manner that aligns with its strategy ensuring that the cumulative size of items placed in each bin does not exceed one. Additionally, the utilisation of each bin is confined to a single designated time slot, after which it becomes inaccessible. The goal of T-BPP is to pack these items without missing their deadlines, with the deadline established upon the item's release, using the minimal number of unit capacity bins. It is worth noting that this particular problem formulation has not been explored previously, thus standing as a unique and uncharted research area.

To address this problem, we present the Pack-at-Deadline algorithm with a competitive ratio of 2 and a lower bound of 1.67. Furthermore, our analysis demonstrates the existence of a lower bound of 1.2 for the problem.

1 Introduction

In the context of reality, the future is unknown. Consequently, a prevailing characteristic of most real-world predicaments resides in the inability to anticipate these impending inputs confidently. In the computational world, these problems are called *online problems*. Given the online problem, an online algorithm has to make an irrevocable decision based on previous events with limited information about the future. The algorithm's decision for an arriving item based on the previously given input holds profound repercussions for the algorithm's broader performance trajectory.

In the last 30 years, the interest in online algorithms has increased since *competitive analysis* was introduced. The linchpin of this analytical framework resides in the conceptual construct of the *competitive ratio*. This evaluative metric endows the observer with insights into the cost efficacy of an online algorithm. In order to calculate the competitive ratio of the online algorithm, its performance is compared against the performance of an *optimal offline algorithm*, one endowed with prescient knowledge of all forthcoming inputs. Within computer science, operations research, and economics, online problems have engendered extensive scholarly contemplation[15].

One widely recognised online problem is the *online bin packing* problem. In this scenario, we have a *list of items* released one by one with a positive size not greater than 1, and an infinite sequence of bins, each with a unit capacity. The online algorithm has to pack every item into the bin before the following item is released, ensuring that the sum of item sizes in each bin does not exceed one. The main objective is to use the fewest bins possible for this arrangement[14]. This concept applies to various practical situations where efficient resource utilisation is crucial. For instance, consider a setup involving multiple processors with a fixed capacity, where tasks of varying sizes are introduced sequentially. The primary objective is to allocate these tasks to the processors such that the load of each processor does not exceed its capacity and resource consumption is minimised. It is important to note that specific costs are associated with turning the processors on and off[6].

This thesis will focus on *online timely bin packing problem* (T-BPP). We introduce the new concept of feasible time slots for items. That is item i has an interval $[r_i, d_i]$, where r_i and d_i are the *release time* and *deadline* of item i, respectively. An online algorithm learns about the item's interval when it is released and must pack it without missing its deadline. Furthermore, at time slot t, the online algorithm can pack *available items* with release time less or equal to t and a deadline greater or equal to t into the bins in an *offline manner*. That means the algorithm can move items from

one bin to another during the packing process at time slot t and leave some items unpacked if their deadline is after t. However, the cumulative size of items placed in a bin must be at most the bin's capacity, i.e. one. It is crucial to note that each bin can only be used during a *single time slot*, after which it cannot be accessed anymore. The goal of this problem is to pack these items without missing their deadlines using the fewest bins of a unit capacity.

The Subsection 1.1 of this thesis will describe the applications of the problem. Further, in Section 2, the previous studies will be analysed and compared to T-BPP. We will give problem definitions in Section 3. In Section 4, we introduce our algorithm Pack-at-Deadline and scrutinise its performance. In Section 5, we give a lower bound of T-BPP. In Section 6, we give some observations that we have made while working on this thesis. Finally, we conclude this thesis project in Section 7.

1.1 Motivation

Online timely bin packing problem (T-BPP) can be applied in many real-case scenarios. In this section, we will describe some of these cases.

We can use the online timely bin packing concept in partitioning the orders into trucks for delivery in logistics. Consider that we have products that arrive at the company warehouse after the consumer makes an order. Those products need to be dispatched from the warehouse before their deadline that is available at the time of an order's arrival. However, trucks are expensive and have a limited capacity. Therefore, the company wants to minimise the number of trucks delivering orders. In this case, orders can be seen as items and trucks as bins. Each order has an arrival/release time and deadline, while trucks have the same capacity.

In the second scenario, consider a corporate entity that engages the services of independent contractors. These contractors are engaged for specific days, during which the company pays them a full day's, equivalent to 8 hours, salary to complete the assigned tasks. The tasks, characterised by a maximum duration of 8 hours, possess predetermined deadlines, different release times and lack the capacity for interruption once commenced. In light of the financial commitment required to remunerate these freelancers, the primary objective is to minimise the frequency of engaging their services. Consequently, the imperative arises to effectively allocate the tasks mentioned above among the available freelancers. In this context, the tasks are analogous to items, while the freelancers can be seen as bins.

2 Literature review

This part will briefly explain the papers we studied while working on this master's thesis. Our review shows that although some settings might seem similar to our problem's settings, *Online Timely Bin Packing (T-BPP)* has distinct features and hasn't been explored before. Nevertheless, these papers gave us a solid starting point, and some of the ideas they contained were valuable in guiding us through our thesis project.

2.1 Online bin packing problem with buffer and bounded size revisited[1]

The authors of this paper considered the existing online bin packing variation with a buffer and items with bounded size. In this setting, there is a buffer of a constant size and items with a size that ranges between $(\alpha, 0.5]$, where $0 \leq \alpha < 0.5$. The online algorithm can pack an item into an open bin or place it in the buffer when it arrives. Moreover, at most one bin can be opened at any time, and items from the buffer should be packed into the bin when there is no more arriving item. Zhang et al.[1] improved Zheng et al.[16] results by decreasing the upper bound from 1.4444 to 1.4243 and increasing the lower bound from 1.3333 to 1.4230. The authors considered two cases with different buffer sizes. In the first case, the buffer size was $|S| = 2$, and the proposed algorithm's *asymptotic competitive ratio* (ACR) was equal to 1.4375. In the second case, the buffer size was $|S| = 3$, and the proposed algorithm had an ACR of 1.4243.

The considered problem is similar to our proposed problem in that both have the concept of buffers. However, in our case, the buffer size is unlimited, while items have a deadline by which they need to be packed into the bins.

2.2 Online bin packing with delay and holding costs[2]

Ahlroth et al.[2] studied the online bin packing problem that has the *delay* and *holding costs* in addition to the bin opening cost. In this problem, the item should be assigned to a bin immediately upon arrival, and it collects a delay cost until the assigned bin is closed. Additionally, each opened bin incurs a holding cost proportional to the time the bin was open. The authors considered the linear costs case and the bilinear costs case. The delay cost equals 1 and 2 for linear and bilinear cases, respectively, where s_k is the size of k^{th} item.

$$D_{(k)}(t) = d \cdot t \tag{1}$$

$$D_{(k)}(t) = d \cdot s_k \cdot t \tag{2}$$

The holding cost 3 has the linear structure with a constant value h > 0 in both cases for bin i.

$$H_i = h \cdot t \tag{3}$$

Ahlroth et al.[2] came up with a lower bound of 2 for both cases even when either one of (but not both) holding cost or delay cost rates is zero. In the linear case, the authors came up with a *Release-on-balance* (ROB) algorithm with parameter α, where for a fixed $\alpha > 0$, each bin i is closed when $H_i + D_i = \frac{1}{\alpha}B$, where B is the bin's capacity. *The competitive ratios* for ROB had the following value:

- competitive ratio is at most 7, when $\alpha = \frac{5}{2}$, $h \neq 0$ and $d \neq 0$;
- competitive ratio is at most 4, when $\alpha = 3$ and $h = 0$;
- competitive ratio is at most 5, when $\alpha = 4$ and $d = 0$.

The authors developed the *Fast release-on-balance* algorithm for the bilinear case. It has a *competitive ratio* of at most 3 when the delay cost is bilinear and when there is no delay cost.

2.3 Bin stretching with migration on two hierarchical machines[3]

The authors of this paper, Akaria & Epstein, considered semi-online scheduling with migration on two machines with a hierarchy. There are two machines with different hierarchies, and jobs require different *grades of service* $g_j \in 1, 2$ that correspond to a machine hierarchy. Machine M1 can run jobs of all hierarchies, whilst machine M2 can only run jobs of hierarchy 2. When a new job j arrives, there is a possibility to migrate jobs from one machine to another. However, the migrated jobs' total processing time should not be greater $M \cdot p_j$, where $M > 0$ is a fixed *migration factor* and p_j is the processing time of job j. The objective of this paper is to schedule jobs in such a way that the makespan is minimised. The problem is semi-online because authors assume that the algorithm already knows the optimal makespan, which equals 1.

Akaria & Epstein[3] proposed an algorithm that considers migration only when an item of hierarchy 2 arrives. The idea of the algorithm is to find a

subset so that the load on machines are balanced and completion time does not exceed $1 + \mu$, where μ is a function of M.

The authors reached the following results:

- For the case $M \geq 2.5$, the lower and upper bounds of $\frac{2M+5}{2M+3}$
- For the case $\frac{3}{4} \leq M < \frac{5}{2}$, the lower and upper bounds of 1.25
- For the case $\frac{1}{2} \leq M < \frac{3}{4}$, the lower and upper bounds of 2 - M
- For the case $\frac{2}{3} \leq M < \frac{3}{4}$, the lower and upper bounds of 2 - M
- For the case $0 \leq M < \frac{1}{2}$, the lower and upper bounds of $\frac{3}{2}$

2.4 Online bin covering with limited migration[4]

As in the previous paper, Berndt et al. investigated the limited migration in an alteration - the *online bin covering* problem. Bins are covered if the total load of the bin is greater or equal to the bin size, which is usually one. In the bin covering problem, given items of size between 0 and 1, the goal is to partition those items into bins so that the number of covered bins is maximised. The authors loosened the irreversibility requirement and considered the migration of items with migration factor β. Moreover, the authors considered four variants of the problem, dynamic or static versions with amortised or non-amortized migration factors. In the static version, only insertions are allowed; in the dynamic case, it is possible to depart and insert items. Algorithms have a migration factor of β when they can migrate $\beta \cdot s(i)$ items when item i arrives, where $s(i)$ is the size of item i. Furthermore, we say that algorithm has an amortised migration factor of β when the total size of items migrated by the algorithm is at most $\beta \cdot |S|$, where S is the set of items migrated by the algorithm. Berndt et al. proposed an algorithm that has the main idea of keeping in balance the number of bins containing two big items, one big item, and no medium items where big items have size $s_i \in (\frac{1}{2}, 1]$ and medium items have size $s_i \in (\epsilon, \frac{1}{2}]$. The algorithm has a tight bound of 1.5 for all cases except amortised static version. In this case, the authors concluded that no online algorithm with a constant amortised migration factor exists.

2.5 Algorithms for the Relaxed Online Bin-Packing Model[5]

Gambosi et al. "relaxed" a traditional online bin packing problem so that repacking is possible. When a new item arrives, a constant number of items can be moved from one bin to another. It is useful when we want to revoke some operations by paying some cost. Moving cost leads to the fact that the number of such movements is counted toward the cost of the algorithm. The goal of the problem is to minimise the number of bins that were opened and the total cost of the algorithm.

The authors provided two algorithms whose main idea is to nonuniformly partition the bin into 4 and 6 chunks and fill those partitions. The moving cost is equal to the number of elements in the set being moved from one bin to another.

They achieved the *competitive ratio* of 1.5 and 1.33 for algorithms with 4 and 6 partitions, respectively. Moreover, for the first algorithm, the space and runtime complexities were linear, while, for the second one, space complexity was linear and runtime complexity was $O(nlogn)$.

Both papers, "Online bin covering with limited migration" and "Algorithms for the Relaxed Online Bin-Packing Model", consider migration but have different contexts.

2.6 Online bin packing with overload cost[6]

Luo et al. introduced another variation of the bin packing problem where *overloading* the bin is allowed[6]. Given a unit capacity bin and a list of items of size between 0 and 1, the algorithm has to pack items into the bins so that the total cost is minimised. The total load of the bin is allowed to exceed its capacity. However, the algorithm acquires an extra cost equal to $c \cdot max\{w_i - 1, 0\}$ for overloading, where w_i is the total load of the bin i and $c \geq 0$ is the overload cost factor. Moreover, there is a unit cost for opening a new bin. Therefore, the total cost that the bin collects equals $1 + c \cdot max\{w_i - 1, 0\}$.

Luo et al. proposed *First-Fit algorithm with a fixed overload* (FFO). It works the same way as *First-Fit* (FF) but permits the overloading. FFO works as follows: whenever a new item comes, the algorithm packs it into the first opened bin that fits that item with an overload; otherwise, FF opens a new bin.

The authors gave the following results:

- Competitive ratio is max(1,c), if $0 \leq c < \frac{3}{2}$

- Competitive ratio is 1.5 if $\frac{3}{2} \leq c < 1 + 2\sqrt{3}$
- Competitive ratio is 1.577 if $1 + 2\sqrt{3} < c < 17$
- Competitive ratio is 1.667 if $17 \leq c$

In our setting, overloading is not allowed. Furthermore, there are more efficient ways to pack items than FF in *Online Timely Bin Packing* since we have more information about items than the traditional *Bin Packing Problem*.

2.7 A θ - Competitive Algorithm for Scheduling Packets with Deadlines[7]

Vesely et al. considered the online packet scheduling problem with deadlines (*PacketScheduling*) and proposed a new algorithm with a tight bound. In *PacketScheduling*, packets that arrive over time have deadlines and non-negative weights representing their urgency and priority, respectively. It is worth noting that only one packet can be transmitted in one time slot. That means some packets can miss their deadlines and are dropped irrevocably. Therefore, the goal of *PacketScheduling* problem is to create a transmission schedule that maximises the total weight of packets that have been transmitted successfully.

The authors introduced a new concept of the *plan* in their algorithm. The plan is the subset of pending packets that can be feasibly scheduled with the maximum weight. When an arbitrary packet p is chosen to be transmitted from the plan, the new feasible packet $p\prime$ is inserted into the plan so that the plan's weight is maximised.

The problem was analyzed before by[17],[18],[19], and[20] where they proposed a lower bound of 1.618. The algorithm given in this paper has an upper bound of θ, where θ is 1.618, which means the analysis is tight.

This problem is similar to ours because arriving items also have a deadline. However, in our case, several items can be packed into a bin simultaneously and have the same priority (weight).

2.8 Online scheduling of packets with agreeable deadlines[8]

The problem statement in this paper is similar to the problem statement in the previously described paper[7] with minor differences. They are:

- The deadlines are agreeable, i.e. the deadline of the newly arriving job is (slightly) greater than the deadline of the previous job
- The deadlines of jobs are s-bounded, i.e. each packet has at most s span.

The authors provided three algorithms. They are *MG (modified greedy), RMG (randomised modified greedy)*, and *TS ("two-speed")* algorithms. In *MG*, they consider the most urgent and highest-priority jobs each time. If $w_u \geq w_h/\theta$, then the algorithm sends an urgent job, here w_u and w_h are the weights of urgent and high-priority jobs, respectively. In *RMG*, they also consider the most urgent and highest-priority jobs. However, in this case, the authors send the urgent job with the probability w_u/w_h and the high-priority job with the remaining probability. In *TS*, both jobs are sent in parallel with the assumption that it is allowed to send several jobs simultaneously.

The authors showed θ, 4/3, and 1-competitive asymptotic ratios for *MG*, *RMG*, and *TS* respectively.

2.9 Online analysis of the TCP acknowledgment delay problem[9]

Dooly et al. discussed *TCP Acknowledgment delay* problem. In this problem, there are n arriving packets, and the goal is to divide those packets into k partitions so there will be k acknowledgements. Packets can be delayed until they are acknowledged. However, their latency increases in that case. Moreover, there are costs for acknowledgement and the latency of packets. However, the algorithm will be charged only once for acknowledging the partition regardless of the number of packets in the partition that it acknowledges. The goal of the problem is to partition n packets into k partitions so that the cost is minimised. To ensure that all packets are acknowledged, authors set the following constraints: $k \geq 1$ and $t_k \geq a_n$.

To solve the TCP acknowledgement delay problem, the authors assume an oracle exists that tells the arrival time of the following L items, where $L \geq 0$. The idea of the algorithm proposed by the authors is to balance the acknowledgement and latency costs. That means the algorithm acknowledges the packets when the latency cost equals the acknowledgement cost.

They reached the following results:

- Competitive ratio of 2 and 1 for $L = 0$ and $L = 1$ respectively for the first algorithm
- Competitive ratio of 2 in all cases for the second algorithm.

The main difference of this problem with *OnlineTimely Bin Packing (T-BPP)* lies in the fact that, in *T-BPP*, no oracle tells the arrival time of the next item. Furthermore, *T-BPP* allows no latency, and each partition size should not exceed one.

2.10 Online bin packing with arbitrary release times[10]

The online bin packing where items in I have random release times has been considered by Shi and Ye. In this problem, bins have time axis $[0, 1]$ from bottom to top, and each time item is identified as pair of a_i and r_i. Here, a_i corresponds to the size of item i, and r_i is the release time of an item i. When item i arrives, it should be placed above r_i in the bin. The order in which items arrive is independent of their release times. Moreover, to ensure that all items fit into the bins, they set the following constraint: $a_i + r_i \leq 1$ for item i, where $i \in I$. Authors assumed that all items have the same size of $1/K$, where $K \geq 1$.

The authors considered ANY FIT (AF) first, but then they realised that it could not have a constant *competitive ratio* in this problem. Therefore, they proposed a new algorithm that puts the arriving item into the bin with the smallest feasible time. If such a bin does not exist, it opens a new bin. The authors proved that this problem has a tight bound of 2.

Even though this problem has a concept of a release time, it is different from the release time in*Online Timely Bin Packing* and bounded by one. Furthermore, bins can be used in this paper's settings unless they are full.

2.11 Online Scheduling with Hard Deadlines[11]

Goldman et al. considered the online scheduling of jobs with hard deadlines in their paper. There is only one machine that executes jobs, and it can execute one job at a time. Moreover, an online algorithm is unaware of future jobs and must schedule a job that is already known non-preemptively. Once the job is scheduled, the machine is unavailable for the amount of time equal to the job's length. Since every job has a deadline, jobs that miss their deadlines are dropped. The goal of this problem is to maximise the total resource utilisation.

The authors designed various cases and created different algorithms, the results of which are depicted in the following table.

Table 1. Summary of results.

Job lengths	Delay type	Competitive ratio
1	Arbitrary	2
1	Min. delay 1	1.5
{1, k}	Arbitrary	4
{1, k}	Uniform	$(1 + \frac{\lceil k \rceil}{k})$
$\{1, 2, 2^2, ..., 2^c\}$	Arbitrary	3 (c + 1)
$\{1, 2, 2^2, ..., 2^c\}$	Uniform	2.5 (c + 1)
$[1, 2^c]$	Arbitrary	6 (c + 1)
$[1, 2^c]$	"Uniform"	5 (c + 1)

Here, the delay is the time from arrival till the deadline, $k > 1$ is the real number, and c is an integer. Uniform jobs' delays are proportional to the jobs' sizes, while the last "uniform" means that the delay is $lg|J|$ proportional to the job's size.

2.12 A Scheduling Model for Reduced CPU Energy[12]

Yao et al. studied job scheduling that reduces the total energy usage in the CPU. In their paper, the jobs have arrival times, deadlines, and the number of required CPU cycles. All jobs are processed between their arrival time and the deadline by a single processor that has variable speed. Moreover, the energy usage per unit of time P is directly proportional to the convex function of the processor speed. Once the job is scheduled, it cannot be paused, meaning non-preemptive execution. The authors gave an offline optimal algorithm and online $AverageRateHeuristics(AVR)$ to solve the problem online. In AVR, they gave each job j a density equal to $d_j = \frac{R_j}{b_j - a_j}$ where a_j - arrival time, b_j - deadline, R_j - required number of CPU cycles. At time t, the $d_j(t) = d_j$ if $a_j \leq \text{t} \leq b_j$. AVR schedules jobs at the earliest-deadline policy where $s(t) = \sum_j d_j(t)$. They showed that the *competitive ratio* for P(s) = s^p where p ≥ 2 for AVR is between p^p and $2^(p-1)p^p$.

This problem is different from ours because it uses a single processor, while in our problem, we can use multiple bins. Moreover, the cost function in *Online Timely Bin Packing* is a linear function of the input instance's size, while in this paper, it is proportional to the convex function of the input.

2.13 Dynamic Bin Packing[13]

Coffman et al. considered the dynamic allocation of bins where items can arrive and depart at arbitrary times. It is not known when the item arrives and when it departs. However, when the item arrives, it should be packed immediately, and repacking is prohibited. When the item departs, the bin gains s_i space where s_i is the size of the departed item.

The authors used the $First-Fit(FF)$ algorithm to solve this problem and conducted an analysis of its *competitive ratio*. The upper bound for FF was equal to $\frac{k+1}{k} + \frac{1}{k-1} log \frac{k^2}{k^2-k+1}$ where 1/k is the size of the largest item. Moreover, they gave the lower bounds for FF and general case, which are equal to $\frac{k+1}{k} + \frac{1}{k^2}$ and $1 + \frac{k+2}{k(k+1)}$, respectively.

This problem is similar to ours because items have release and departure times. However, our model gives the deadline/departure time at arrival and does not allow to reuse the bins.

2.14 A New and Improved Algorithm for Online Bin Packing[14]

Balogh et al.[14] introduced a new, improved harmonic algorithm for online bin packing and a more simplified analysis of the *competitive ratio* using the weight functions. In previous approaches, mixing items of different types was not allowed. However, their new algorithm, Advanced Harmonics (AH), decreased the *competitive ratio* to 1.57829 by allowing the mix of types.

We considered this paper since we wanted to use weight functions to find the *competitive ratio* for our approach. However, using the weight functions in our setting is complicated due to the intricate nature of the problem and the different possible scenarios that can arise.

3 Definitions

This section presents a compilation of definitions that will be subsequently applied in the context of this thesis.

Problem definition: Given a concept of *feasible time slots* of items, there is a list of *items* $L = \{a_1, a_2, ..., a_n\}$ with *feasible interval* $I_i = [r_i, d_i]$, where r_i is the *release time* of a_i and d_i is the *deadline* of a_i, and size $s_i \in (0, 1]$ for each $a_i \in L$. Notice, $r_i \leq r_j$ where $i < j$. The information about I_i and s_i becomes available when a_i is released at *time slot* $t = r_i$. We say a_i is

available for packing at time slot t if $r_i \leq t \leq d_i$. Furthermore, at time slot t, the online algorithm can pack available items in an offline manner. This implies that the online algorithm can move items between bins during the packing process at time slot t and leave some items unpacked if their deadline is after t. It is essential to highlight that each bin can be used during a single time slot – after that, it becomes inaccessible for further use. Moreover, the online algorithm has a *feasible* packing if it packs all items without missing deadlines. The problem's objective is to find a feasible packing that uses as few bins as possible.

Performance analysis: We denote the cost of an online algorithm ALG, the number of bins used by ALG, on an input L as $ALG(L)$. The cost of optimal offline algorithm OPT on an input L is denoted as $OPT(L)$. The asymptotic approximation ratio compares the cost of an online algorithm against the optimal offline algorithm's cost for inputs for which the optimal offline algorithm's cost is sufficiently large. The asymptotic approximation ratio of ALG, denoted as c, equals[15]:

$$c = \lim_{N \to \infty} \left(\sup_{L:OPT(L) \geq N} \frac{ALG(L)}{OPT(L} \right) \tag{4}$$

In this thesis, we will use the asymptotic approximation ratio, c, to measure the performance of the online algorithm ALG.

Lower bound The lower bound is commonly articulated with the competitive ratio. Specifically, for a given problem, it is established that the competitive ratio of any online algorithm cannot surpass the associated lower bound[21].

Independent items Two items are independent if their intervals do not overlap.

Trigger items The items with the deadline at time slot t that were packed into the bin at time slot t.

The bin's load The load of the bin is the cumulative size of items placed in that bin.

Big bin Bin with the load greater $\frac{1}{2}$.

Small bin Bin with the load less or equal to $\frac{1}{2}$.

4 Pack-at-Deadline algorithm

This section defines our algorithm, Pack-at-Deadline, that packs all available items at time slot t when there is a job with its deadline at t.

First, we give the algorithm and then analyse its competitive ratio and the lower bound.

```
t ← 0;
while there is still an item that will be released do
    if there is/are an item/s with a release time equal to time slot t
     then
        Collect information about the item/s;
    end
    if there is an item with a deadline at time slot t then
        Pack all available items into bins so that the number of
         utilised bins is minimal and the total size of items placed in
         each bin does not exceed one;
    end
    t ← t + 1;
end
```

Algorithm 1: Pack-at-Deadline

4.1 Upper Bound of Pack-at-Deadline

In this subsection, we analyse the *upper bound* of the competitive ratio for Pack-at-Deadline. We denote the number of bins used by Pack-at-Deadline as ALG and the number of bins used by the optimal offline algorithm as OPT.

Lemma 1. *$OPT \geq k$, where k is the number of time slots when* Pack-at-Deadline *opens bins.*

Proof Given the inherent characteristics of the algorithm, it becomes evident that the intervals associated with trigger items across distinct packing time slots do not overlap. This is primarily due to the fact that should such overlap occur, these trigger items would invariably be accommodated within the same time slot. Due to the non-overlapping nature of these intervals, it becomes implausible for any algorithm to effectuate their simultaneous packing within a single bin, necessitating the assignment of an individual bin for each trigger item. Consequently, it follows that an optimal offline algorithm mandates the utilisation of at least k bins. Hence, this lemma holds. □

Lemma 2. Pack-at-Deadline *opens at most one small bin at time slot t.*

Proof Assuming that Pack-at-Deadline initiates the opening of a minimum of two small bins during time slot t, it becomes evident that the cumulative load of any two small bins from this time slot remains below one. Consequently, a potential exists to consolidate these loads within a single bin. This inference implies a suboptimal utilisation of bins by Pack-at-Deadline. Nonetheless, it is established that Pack-at-Deadline adheres to an optimal bin utilisation strategy. Hence, it logically follows that Pack-at-Deadline does not opt to open more than one small bin during time slot t. □

Lemma 3. *$OPT \geq s_{min} \cdot (ALG - k')$ where s_{min} is the cumulative size of items in the big bin with the smallest load opened by* Pack-at-Deadline *and k' is the number of small bins opened by* Pack-at-Deadline.

Proof The number of big bins opened by Pack-at-Deadline equals $(ALG - k')$. Since s_{min} is the cumulative size of items in the big bin with the smallest load, all big bins have a load of at least s_{min}. Therefore, the total size of all items in the input sequence is at least $s_{min} \cdot (ALG - k')$. According to[21], an optimal algorithm's cost is at least the cumulative size of all items in the input sequence. Therefore, this lemma holds. □

Lemma 4. *$OPT \geq s_{min} \cdot (ALG - 2 \cdot k') + k'$ where s_{min} is the cumulative size of items in the big bin with the smallest load opened by* Pack-at-Deadline *and k' is the number of small bins opened by* Pack-at-Deadline.

Proof Considering that Pack-at-Deadline initiates the opening of a small bin during time slot t due to the incompatibility of the items within the small bin with the available space in the larger bins of the same time slot, a crucial observation emerges. This observation entails that the cumulative size of items housed in a small bin and any large bin from the corresponding time slot exceeds the capacity of a single bin (i.e., greater than one).

Upon merging each small bin with any large bin from the same time slot, established during the opening of the small bin, the ensuing inequality materialises: $OPT \geq s_{min} \cdot (ALG - k' - \sum_{j=1}^{k'} 1) + \sum_{j=1}^{k'} \max\left(s_j^{big} + s_j^{small}, 1\right)$, wherein s_j^{big} denotes the load of the large bin during time slot j, and s_j^{small} represents the load of the accompanying small bin during the same time slot.

It is noteworthy that $s_j^{big} + s_j^{small} > 1$ during time slot j when Pack-at-Deadline simultaneously initiates the opening of large bins alongside a small bin. During time slot j when Pack-at-Deadline exclusively triggers the opening of a solitary small bin, even if there are not enough available items at time slot j to fill that bin so that its load is greater than one, the offline optimal algorithm would still opt to open that bin. Hence, it follows

that $OPT > s_{min} \cdot (ALG - k' - \sum_{j=1}^{k'} 1) + \sum_{j=1}^{k'} 1 = s_{min} \cdot (ALG - 2 \cdot k') + k'$. Thus, we have this lemma. □

Lemma 5. *$OPT \geq k'$ where k' is the number of small bins opened by* Pack-at-Deadline.

Proof If Pack-at-Deadline opens a small bin at every time slot t when it packs the items, from Lemma 2, it cannot open more than k small bins where k is the number of time slots when Pack-at-Deadline opens bins. Hence, $k' \leq k$. From Lemma 1, $k' \leq k \leq OPT$. □

Theorem 1. Pack-at-Deadline *has a competitive ratio of 2.*

Proof To prove theorem 1, we consider the following inequalities.

$$OPT \geq s_{min} \cdot ALG - (2 \cdot s_{min} - 1) \cdot k' \tag{5}$$

$$OPT \geq k' \tag{6}$$

eq. (5) and eq. (6) come from Lemma 4 and Lemma 5 respectively.

After multiplying both sides of eq. (5) by $\frac{1}{s_{min}}$, we have the following inequality:

$$\frac{1}{s_{min}} \cdot OPT \geq ALG - \frac{2 \cdot s_{min} - 1}{s_{min}} k' \tag{7}$$

After multiplying both sides of eq. (6) by $\frac{2 \cdot s_{min} - 1}{s_{min}}$, we get:

$$\frac{2 \cdot s_{min} - 1}{s_{min}} \cdot OPT \geq \frac{2 \cdot s_{min} - 1}{s_{min}} \cdot k' \tag{8}$$

The sum of eq. (7) and eq. (8) is equal to:

$$2 \cdot OPT \geq ALG \tag{9}$$

Hence this theorem holds. □

4.2 Lower bound of Pack-at-Deadline

In this subsection, we examine the lower bound of Pack-at-Deadline.

Theorem 2. *The* Pack-at-Deadline *has a lower bound of* $\frac{10}{6}$.

Proof To prove theorem 2, we consider instance I with the following structure, given n is the sufficiently large positive integer and ϵ such that $\frac{1}{6} - 2\epsilon \geq \frac{1}{7}$:

- Set of n items with size of $\frac{1}{6} - 2\epsilon$ and interval [0, 2] denoted as X;
- Set of n items with size of $\frac{1}{3} + \epsilon$ and interval [1, 2] denoted as Y;
- Set of n items with size of $\frac{1}{2} + \epsilon$ and interval [2, 2] denoted as Z;
- 1 item of size ϵ with interval [0, 0] denoted as $trigger_0$;
- 1 item of size ϵ with interval [1, 1] denoted as $trigger_1$;
- 1 item of size ϵ with interval [2, 2] denoted as $trigger_2$;

$trigger_0$, $trigger_1$, and $trigger_2$ items trigger Pack-at-Deadline to pack items at three time slots.

At time slot $t = 0$, $trigger_0$ and X are available for packing, and $trigger_0$ has a deadline. One bin can fit at most six items from X. Moreover, the bin that contains six items from X and $trigger_0$ has a load of less than one. Thus, Pack-at-Deadline packs $trigger_0$ with six items from X in one bin and remaining $n - 6$ items from X in $\frac{n}{6} - 1$ bins. Therefore, Pack-at-Deadline opens $\frac{n}{6}$ bins at time slot $t = 0$.

At time slot $t = 1$, $trigger_1$ and Y are available for packing, and $trigger_1$ has a deadline. One bin can fit at most two items from Y. Moreover, the bin that contains two items from Y and $trigger_1$ has a load of less than one. Therefore, Pack-at-Deadline packs $trigger_1$ with two items from Y in one bin and remaining $n - 2$ items from Y in $\frac{n}{2} - 1$ bins. Therefore, Pack-at-Deadline opens $\frac{n}{2}$ bins at time slot $t = 1$.

At time slot $t = 2$, $trigger_2$ and Z are available for packing, and $trigger_2$ has a deadline. One bin can fit exactly one item from Z. Moreover, a bin that contains an item from Z and $trigger_2$ has a load of less than one. Therefore, Pack-at-Deadline packs $trigger_2$ with an item from Z in one bin and remaining $n - 1$ items from Z in $n - 1$ bins. Therefore, Pack-at-Deadline opens n bins at time slot $t = 2$.

Overall, Pack-at-Deadline opens $\frac{10}{6} \cdot n$ bins. The partition done by Pack-at-Deadline on input sequence I is demonstrated in fig. 1.

The optimal offline algorithm opens one bin for $trigger_0$ at time slot $t = 0$ and packs it together with six items from X. At time slot $t = 1$, it opens one bin for $trigger_1$ and packs it together with two items from X and two items from Y. At time slot $t = 2$, it opens n bins to pack the remaining items; that is $n-8$ bins to pack each of $n-8$ items from X, Y, and Z, six bins to pack six items from Y and Z, two bins to pack the remaining items of Z with $trigger_3$. Overall, the optimal offline algorithm's cost is equal to $n+2$. The partition done by optimal offline algorithm OPT on input sequence I is demonstrated in fig. 2.

The competitive ratio of Pack-at-Deadline for the input sequence I is:

$$c_I = \lim_{N \to \infty} \frac{\frac{10}{6} \cdot n}{n+2} = \frac{10}{6} \tag{10}$$

□

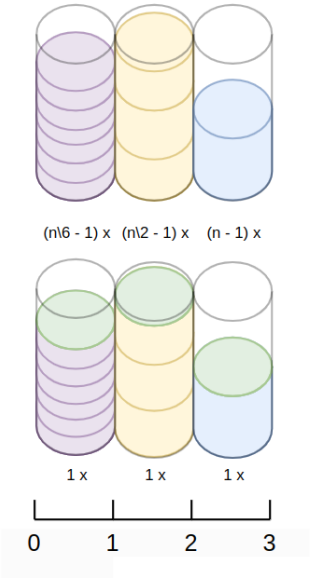

Figure 1. *Partition of I done by* Pack-at-Deadline, Pack-at-Deadline(I)

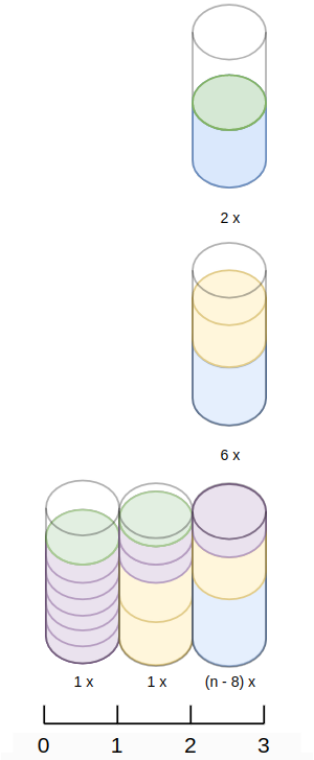

Figure 2. *Partition of I done by OPT,* OPT(I)

5 Lower bound of Online Timely Bin Packing (T-BPP)

Theorem 3. *There is no online algorithm for T-BPP with a competitive ratio less than 1.2.*

Proof First, we assume that we have an arbitrary online algorithm *ALG* with a competitive ratio of under 1.2. Next, we will introduce an adversary demonstrating that no online algorithm can achieve a competitive ratio of less than 1.2. This, in turn, will result in a contradiction with our first statement.

Given a large enough integer $n > 0$, the instance I is structured as follows:

- Set of n items with the interval $[0, 0]$ and with the size of $\frac{1}{2} - \epsilon$ denoted as X;
- Set of n items with the interval $[0, n + 1]$ and with the size of $\frac{1}{2} + \epsilon$

denoted as Y;

- Set of n items of size $\frac{1}{2} - \epsilon$ and with intervals $[j+1, j+1]$ for item j where $j = 1, 2, ..., n$ denoted as Z.

  fig. 3 demonstrates the instance I.

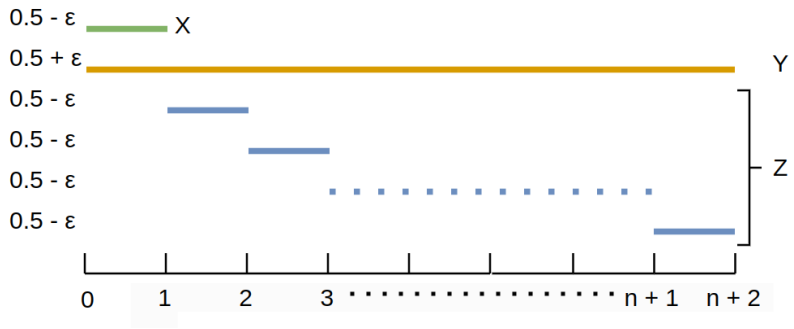

Figure 3. *Instance I*

An adversary releases sets X and Y at time slot $t = 0$ first. Based on the decision made by an arbitrary online algorithm ALG at time slot $t = 0$, the adversary further releases all items in set Z or keeps them unreleased at time slot $t = 1$. Since we denote p as the fraction of items in Y, $p \cdot n$ items from Y, that ALG packed with X at time slot $t = 0$, the remaining portion of items in Y, $(1-p) \cdot n$ items from Y, is packed in $[1, n+2]$.

We consider two cases. In the first one, the adversary does not release Z at time slot $t = 1$; in the second one, it releases Z at time slot $t = 1$.

In the first case, we denote the number of bins used by an arbitrary online algorithm ALG and optimal offline algorithm OPT as ALG_1 and OPT_1, respectively. In this case, ALG opens $p \cdot n$ bins to pack $p \cdot n$ items from X and $p \cdot n$ items from Y together at time slot $t = 0$, $(1-p) \cdot \frac{n}{2}$ bins to pack remaining $(1-p) \cdot n$ items from X at time slot $t = 0$ since two items from X can fit into one bin, and $(1-p) \cdot n$ bins to pack the remaining items from Y after time slot $t = 0$. Therefore, ALG_1 can be described as the function of p as in the equation below:

$$ALG_1 = (p + (1-p) \cdot \frac{1}{2} + (1-p)) \cdot n \tag{11}$$

The OPT_1 equals n since OPT packs one item from X and one item from Y together n times at time slot $t = 0$.

In the second case, we denote the number of bins used by an arbitrary online algorithm ALG and optimal offline algorithm OPT as ALG_2 and

OPT_2, respectively. When the adversary decides to release Z after time slot $t = 0$, ALG opens $p \cdot n$ bins to pack $p \cdot n$ items from X and $p \cdot n$ items from Y together at time slot $t = 0$, $(1 - p) \cdot \frac{n}{2}$ bins to pack remaining $(1 - p) \cdot n$ items from X at time slot $t = 0$ since two items from X can fit into one bin, $(1 - p) \cdot n$ bins to pack the remaining fraction of items of Y and $(1 - p) \cdot n$ items of Z together after time slot $t = 0$, and $p \cdot n$ bins to pack the remaining items in Z after time slot $t = 0$. Therefore, ALG_2 can be described as the function of p as in the equation below:

$$ALG_2 = (p + (1 - p) \cdot \frac{1}{2} + (1 - p) + p) \cdot n \tag{12}$$

In the second scenario, the value of OPT_2 is equivalent to $\frac{3}{2} \cdot n$, as it opens $\frac{1}{2} \cdot n$ bins to accommodate X items at time slot $t = 0$, and subsequently utilises n bins to accommodate one of the Y items and one of the Z items together n times after the time slot $t = 0$.

We can see ALG_1 is decreasing as p is increasing while ALG_2 is increasing as p is increasing.

$$ALG_1 = (\frac{3}{2} - \frac{1}{2} \cdot p) \cdot n \tag{13}$$

$$ALG_2 = (\frac{3}{2} + \frac{1}{2} \cdot p) \cdot n \tag{14}$$

By finding p when the competitive ratios of the first and second cases are equal, we find the lowest competitive ratio of ALG. That is $p = \frac{3}{5}$. If ALG packs more than 60% of Y at $t = 0$, then the adversary releases all items in Z after time slot $t = 0$, otherwise, it does not release Z. By doing that, the adversary ensures that no online algorithm ALG does better than $1.5 - 0.5 \cdot 0.6 = 1.2$-competitiveness. □

6 Observations

We made several observations when we studied the problem and we present them in this section.

Observation 1. Prioritizing big items during the packing process makes the algorithm's overall performance potentially better

Given that big items occupy greater space within a bin, prioritising them at packing guarantees adequate room allocation within bins. This approach can result in more space for accommodating smaller items available for packing, potentially decreasing the total number of bins required.

Moreover, an adversary may not present subsequent items if big items are not assigned to the bins at the earlier time slots. As a result, they will be packed in separate bins, causing space waste that could be utilised efficiently by small items in earlier time slots.

Observation 2. Items that have closer deadlines should be given priority over items with later deadlines.

By refraining from prematurely placing items with distant deadlines into bins, we afford ourselves the adaptability to modify our approach in response to potential changes in input from an adversary. The postponement of packing for such items introduces an additional layer of complexity for the adversary in terms of crafting input that undermines the effectiveness of our decisions.

Observation 3. Using deterministic offline algorithms during the packing process may help analyse the upper bound.

Given the inexhaustible range of approaches available to an adversary to undermine an online algorithm's decision, possessing an upper bound for the cumulative input size is desirable. Therefore, using decisive offline algorithms that set boundaries for item sizes during the packing process may offer distinct advantages. An example of such algorithms that has demonstrated results close to optimality includes the First-Fit-Decreasing (FFD) and Modified-First-Fit-Decreasing (MFFD) algorithms, as explored by Johnson (1985)[22]. These algorithms yield local minimum solutions and offer the additional advantage of simplicity in implementation alongside polynomial time complexity.

7 Conclusion

We introduced a variant of the online bin packing problem with the new concept of feasible intervals for items, *Online Timely Bin Packing Problem* (T-BPP). In this thesis, given a list of items $L = \{a_1, a_2, .., a_n\}$ and an infinite number of unit capacity bins, each item a_i is characterised by a feasible interval $I_i = [r_i, d_i]$ and a positive size $s_i \leq 1$, where r_i and d_i are the release time and deadline of a_i, respectively. Given that bins can be used at a single

time slot after which they are inaccessible, an online algorithm has to pack items into bins without missing deadlines so that the load of each utilised bin is not larger than one. Items can be packed between their release time and deadline, where the algorithm learns about the item's deadline when it is released. Given that the online algorithm packs items in an offline manner, the objective of the problem is to pack items feasibly using as few bins as possible.

We analysed the problem and gave an algorithm, Pack-at-Deadline, with the worst-case asymptotic approximation ratio of 2. Furthermore, we provided an instance to show that the competitive ratio of Pack-at-Deadline is strictly greater than 1.67. We gave the lower bound for T-BPP, which is equal to 1.2.

Additionally, we gave observations about the problem that might help in further research, such as packing items with a size greater than $\frac{1}{2}$ with a high priority while packing items with a size smaller than $\frac{1}{2}$ in an ascending sorted order of their deadlines.

There are some open questions for further research; for instance, if the decision tree model to design adversary helps to increase the lower bound of T-BPPor if using offline bin packing algorithms that showed near to optimal results such as MFFD^{22}, in the packing process could help to decrease the upper bound. Further research also includes relaxation of T-BPP, such as allowing overloading or delays.

8 Acknowledgements

We thank Dr. Liu for her guidance throughout the thesis project with the discussions of algorithms and their analysis.

References

[1] Zhang, M., Han, X., Lan, Y. & Ting, H.-F. Online bin packing problem with buffer and bounded size revisited. *Journal of Combinatorial Optimization* **33**, 530–542 (2017).

[2] Ahlroth, L., Schumacher, A. & Orponen, P. Online bin packing with delay and holding costs. *Operations Research Letters* **41**, 1–6 (2013).

[3] Akaria, I. & Epstein, L. Bin stretching with migration on two hierarchical machines. *arXiv preprint arXiv:2206.06102* (2022).

[4] Berndt, S. *et al.* Online bin covering with limited migration. *arXiv preprint arXiv:1904.06543* (2019).

[5] Gambosi, G., Postiglione, A. & Talamo, M. Algorithms for the relaxed online bin-packing model. *SIAM journal on computing* **30**, 1532–1551 (2000).

[6] Luo, K. & Spieksma, F. C. Online bin packing with overload cost. In *Algorithms and Discrete Applied Mathematics: 7th International Conference, CALDAM 2021, Rupnagar, India, February 11–13, 2021, Proceedings*, 3–15 (Springer, 2021).

[7] Veselỳ, P., Chrobak, M., Jeż, Ł. & Sgall, J. A-competitive algorithm for scheduling packets with deadlines. *SIAM Journal on Computing* **51**, 1626–1691 (2022).

[8] Jeż, Ł., Li, F., Sethuraman, J. & Stein, C. Online scheduling of packets with agreeable deadlines. *ACM Transactions on Algorithms (TALG)* **9**, 1–11 (2012).

[9] Dooly, D. R., Goldman, S. A. & Scott, S. D. On-line analysis of the tcp acknowledgment delay problem. *Journal of the ACM (JACM)* **48**, 243–273 (2001).

[10] Shi, Y. & Ye, D. Online bin packing with arbitrary release times. *Theoretical computer science* **390**, 110–119 (2008).

[11] Goldman, S. A., Parwatikar, J. & Suri, S. Online scheduling with hard deadlines. *Journal of Algorithms* **34**, 370–389 (2000).

[12] Yao, F., Demers, A. & Shenker, S. A scheduling model for reduced cpu energy. In *Proceedings of IEEE 36th annual foundations of computer science*, 374–382 (IEEE, 1995).

[13] Coffman, E. G., Jr, Garey, M. R. & Johnson, D. S. Dynamic bin packing. *SIAM Journal on Computing* **12**, 227–258 (1983).

[14] Balogh, J., Békési, J., Dósa, G., Epstein, L. & Levin, A. A new and improved algorithm for online bin packing. *arXiv preprint arXiv:1707.01728* (2017).

[15] Borodin, A. & El-Yaniv, R. *Online computation and competitive analysis* (cambridge university press, 2005).

[16] Zheng, F., Luo, L. & Zhang, E. Nf-based algorithms for online bin packing with buffer and bounded item size. *Journal of Combinatorial Optimization* **30**, 360–369 (2015).

[17] Andelman, N., Mansour, Y. & Zhu, A. Competitive queueing policies for qos switches. In *SODA*, vol. 3, 761–770 (Citeseer, 2003).

[18] Chin, F. Y. & Fung, S. P. Online scheduling with partial job values: Does timesharing or randomization help? *Algorithmica (New York)* (2003).

[19] Hajek, B. On the competitiveness of on-line scheduling of unit-length packets with hard deadlines in slotted time. In *Proceedings of the 2001 Conference on Information Sciences and Systems* (2001).

[20] Zhu, A. Analysis of queueing policies in qos switches. *Journal of Algorithms* **53**, 137–168 (2004).

[21] Algorithms for decision support lecture (2021).

[22] Johnson, D. S. & Garey, M. R. A 7160 theorem for bin packing. *Journal of complexity* **1**, 65–106 (1985).