AUGMENTED REALITY COMBINED WITH 3D PACKING ALGORITHM AS TOOL FOR CONTAINER LOADING

Jury:

Supervisor:

Michael SCHYNS

Reader:

Justine EVERS

Master thesis by:

Hugo PONCELET

With a view to obtaining the diploma of

Master's Degree in Business Engineering

specializing in Digital Business

Academic year 2021/2022

Executive Summary

Every year, the number of parcels transported increases significantly, partly due to the generalization of e-commerce. This increase results in an augmentation of ships, trucks and planes needed to transport all these packages. We study in the master thesis a possible improvement in the loading process, which might result in faster loading and more efficient storage in containers. This improvement comes from the combination of two very different areas, namely augmented reality and the container loading problem.

First, a container loading algorithm is developed, allowing to take a large number of boxes and pack them in a container. This algorithm uses a placement heuristic as well as an improvement heuristic to optimize the result. This software will be run on a server and data must be stored in a database and the communication must be possible through an application programming interface (API).

Afterwards, the objective is to combine this algorithm with a program developed on Unity running on a Hololens device. This software allows to visualize and interact with the solution proposed by the packing algorithm and to allow a human operator to assemble the container according to the instructions issued.

In addition to these tools, we also developed a web interface and a box generator.

maximum number of words allowed:	30 000
Number of words (introduction to conclusion):	29 314

Acknowledgments

This master thesis would not have existed without the valuable help of many people. First of all, I would like to thank Professor Schyns, who first of all integrated me very well into the University of Liege. This professor also gave many courses during these two years of master which proved to be particularly enriching and preparatory to this master's thesis. I think in particular of the *Introduction to Digital Business* class, which introduced me to web development and the Unity program, which was unknown to me until then. The *Business Analytics* course gave me the knowledge to better organize data, which was completed by the *Database for management* course, which focused on modeling a database. Finally, the *Digital Business Capstone Project* course allowed me to realize a larger project in Python, which proved to be good training for this other large-scale project that is this work. By his passion for augmented and virtual reality, it is also this professor who introduced me to this subject which was previously little known to me. Once his passion was shared, this master thesis subject became quite evident. Therefore, I would like to thank Professor Schyns for everything described above and for all the help given during these months of work on this thesis.

I would also like to thank Justine Evers, who, during these months of work, was constantly present at the meetings with Professor Schyns. During these meetings, her mastery of the subject concerning the container packing problem proved to be helpful and allowed on many occasions to bring a different vision. Justine also provided some scientific papers that proved to be particularly interesting.

I would also like to thank the Digital Business Laboratory held by Mr. Schyns, which brought valuable help during the project. Their mastery of Unity as well as technologies like augmented and virtual reality was beneficial to this work.

I would like to thank Elisa Etienne for a re-reading of this work which enabled clearer sentences and paragraphs.

Finally, I would like to thank my family, who supported me during these months of work and more globally during all my years of study. More precisely, the development of the 3D bin packing algorithm was very complicated at the beginning of this work, and it was not from deep discussions with my father that many ideas emerged.

TABLE OF FIGURE

ABBREVIATION LIST

C&P: cutting and packing

OR: operation research

SKP: Single Knapsack Problem

SBSBPP: *Single Bin Size Bin Packing Problem*

SSSCSP: *Single Stock-Size Cutting Stock Problem*

ULDs: Unit Load Devices

AR: Augmented Reality

VR: Virtual Reality

HDM: head-mounted display

IMA: industrial maintenance and assembly

API: Application Programming Interface

JSON: JavaScript Object Notation

RDBMS: Relational database management system

ORM: Object-Relational-Mapper

SOAP (APIs): Simple Object Access Protocol

RPC (APIs): Remote Procedure Calls

REST (APIs): Representational State Transfer

TABLE OF CONTENT

1 Introduction and Context

1.1 Ecommerce activity

1.1.1 Introduction

Every year, the volume of activity in e-commerce activity increases. As a result, new challenges arise as the logistics to transport all goods to their location become more complex due to larger volumes and requirements to transport the parcels with a minimum amount of time.

According to Semerádová & Weinlich (2022), electronic commerce is a concept that is quite hard to define and for which a wide range of definitions exist in the literature. A first broad definition includes activities on the internet, including hardware and software infrastructures. A second concept is an electronic transaction between two entities involving a credit card transaction and the electronic payment infrastructure to handle the financial transaction and recording. A more restrictive definition of e-commerce defines it as only the online sales or purchase of goods or services, whether this includes a payment transaction or delivery. The conclusion definition they give to the term e-commerce involves all business transactions that occur on the internet, with a prominence of user shopping experience. Since the development of the internet, people have changed the way they behave, think, and purchase or sell goods or services (Semerádová & Weinlich, 2022). Businesses have also progressively evolved with advanced digitized tools to better understand their customers, their needs, preferences, and shopping processes. With the development of e-commerce, these tools and programs have increased in quantity and quality. Big Commerce, an American company, traded on the NASDAQ involved in e-commerce, defines six different models of e-commerce that businesses can be categorized into. These types include B2C, B2B, C2C, C2B, B2A, and C2A. B stands for Business, C for Customer, and A for Administration (BigCommerce, 2022). This enables us to understand that there exist several forms of e-commerce.

The consequence of the development of e-commerce is that the volume of parcels transported by plane, ship, or truck has been increasing for decades. According to Pitney (2018, cited in Peters & Schyns, 2020), the volume of parcels transported annually is now gigantic, reaching 87 billion packages in 2018. Big firms in e-commerce, such as Amazon, demonstrate this phenomenon. Indeed, according to Sheetz (2019), Amazon should deliver around 6.5 billion packages this year (2022), while UPS would reach 5 billion packages and FedEx 3.4 billion packages. During the time interval between 2018 and 2022, Sheetz (2019) evaluated a 68% compound annual growth of Amazon Logistics US packages transported.

These recent years, another factor in the development of e-commerce increased concerning big data systems. With the help of such IT tools, companies can reach consumers in a targeted way (Semerádová & Weinlich, 2022). As a result, the advertisement that appears on users' screens often corresponds to a product that they are more likely to buy than a product with a non-targeted advertisement. This phenomenon is particularly conducive to compulsive buying. There are now even audience targeting tools available to businesses of all sizes. Thus, a small business can target the audience it wants through tailored advertising, enabling showing the ideal product to the target audience at the right time.

A new phenomenon is the influence of social media influencers that sometimes significantly impact consumers' shopping intentions (Suková & Míková, 2022). Brands use this communication tool to reach potential consumers. Using the trust that this target audience has in the influencer, the brand promotes its product in an incisive way. Still, according to Suková & Míková (2022), marketing influence using product placement positively influences a company's sales. Indeed, although the study they have done tends to demonstrate specific negative impacts of product placement, in particular the

annoyance of users for continuous advertising and the decline in the credibility of influencers, it nevertheless demonstrates that a brand can significantly increase its sales through this type of marketing. Moreover, this promotion of products is done almost exclusively on products that can be purchased online, contributing to the increase in the importance of e-commerce.

Finally, customers' expectation level in terms of waiting time for their package has increased considerably. Most people expect to receive their package in the following days or sometime in the first 24 hours after purchase. This concept brings many challenges to the supply chain as the waiting time becomes a key argument for the consumers.

1.1.2 The Covid 19 influence

At the end of 2019, the Covid19 virus is detected in China and progressively spread worldwide. On the 17th of Match 2020, the total lockdown is declared in Belgium. This pandemic has significantly impacted consumers' behaviors, especially toward e-commerce. Indeed, the complete lockdown in most countries forced people to stay at home and purchase via the internet. In Belgium, more than 20.000 online shops have opened in the year 2020, bringing the e-commerce park of the country to around 48.000 points of sale (Driessche, 2021). This increase is an absolute record for this country, the previous year (2019) having seen the creation of only 5000 online shops. Again according to Driessche (2019), e-commerce in Belgium experienced a 7.5% increase in transaction volumes, which made it possible to reach the sum of 8.8 billion euros in 2020. This relatively modest growth is explained in particular by the collapse (-70%) of online reservations for travel and events. We, therefore, understand that it is the purchase of physical products that compensates for this fall. Sectors such as food, DIY, and toys were particularly little digitized before the crisis, and these sectors, therefore, had to adapt to the first confinement. This caused a sharp increase in the volume of online sales of these products. Indeed, once the fear of online ordering was overcome, many customers realized how easy it was and started to increase their confidence in these online platforms (Semerádová & Weinlich, 2022). At the end of the confinement, a face-to-face return of customers took place, but a large volume of purchases is still being made online. Mores have therefore evolved and are now mature enough to remain in the long term. Lockdown has also prompted governments to promote teleworking as much as possible. Massive digitization of many systems has therefore ensued. Most of the workers thus found themselves working from home, leading them to train on numerous IT tools. As a result, the digital divide has shrunk considerably in these recent years which has allowed greater user confidence in electronic platforms, of which e-commerce is a part.

1.1.3 Issues of e-commerce

An issue of e-commerce, nowadays more than ever, might be its ecological impact. Defenders and detractors of e-commerce oppose each other on the issue of its environmental impact, with reports to back it up (Moustique, 2022). A report by the consulting firm Oliver Wyman, financed by Amazon, noted that "in-store purchases result in an average of two to three times more greenhouse gas emissions than online purchases.". The explanation lies in the fact that a dozen consumers who have driven to the store in their personal vehicles emits more CO2 than a van full of packages. This argument was however countered by the National Council of Shopping Centers, which, on the contrary, stated that customers buying in shopping malls and real shops buy an average of 6.3 products, which therefore has a better carbon footprint. The decisive point here relies on the frequency of deliveries. The current online sales model favors faster delivery time, which is put under pressure by Amazon and other companies. Multiple deliveries might be necessary for the same basket so that each item arrives as fast as possible at its destination. The other important factor in the environmental impact of e-commerce is the free return option, which is most of the time available to consumers. Greenpeace

Germany conducted a survey in which they discovered that among Germans under 30, a quarter of all packages ordered are returned to the sender (RTBF, 2019). Even more concerning is that more than half of these young consumers know they will return some of their orders when they buy them. For returned products, it is sometimes less expensive to destroy the products than to put them back in the supply chain. About 30% of the items returned by disappointed Amazon Germany customers end up in the trash, also revealed a first survey conducted in June by German public television (ZDF) and the business magazine WirtschaftWoche (RTBF, 2019).

Another issue in the development of e-commerce is the traffic it may cause on the roads. This problem directly concerns the city of Liege, as the neighboring airport, Liege Airport, recently welcomed Alibaba as its European hub. Both Liege Airport and Alibaba play a significant role in the field of e-commerce, Liege Airport being the eighth largest cargo airport in Europe (Peters & Schyns, 2020). According to Watching Alibaba (2020), a local activist group against the arrival of Alibaba at Liege Airport, the volume of lories on the road network around the airport and the city will undoubtedly increase tenfold. They claim that it is expected that around 1500 more trucks per day will be on the roads surrounding the city of Liège in a few years once Alibaba is fully developed in this region. This may cause a significant increase in traffic, both for truck drivers as well as citizens. The situation is even more problematic because the liege road network's capacity is already saturated.

1.2 Objectives and contribution of the thesis

This thesis aims to study a possible improvement of an essential step of the delivery process, more precisely, the filling of a container, or pallet, by an operator on the ground. The objective is to elaborate and develop a tool that would enable a better and faster loading process. Furthermore, this primary tool will be complemented with a range of modules that might help to increase its efficiency and make it easier to manage. Among these tools, we list a container loading packing algorithm, a Hololens 2 software, a box instances generator, a Unity visualization tool, and a web interface. More globally, this work aims to develop a proof of concept combining a box packing algorithm with an augmented reality device. To our knowledge, no such product has been developed yet.

Although such a tool might be used in a wide range of sectors, we decided to focus on e-commerce in the introduction because this sector is one of the prominent actors in the evolution of the number of packages transported worldwide. Moreover, unlike a classic company that sells only a few ranges of products, therefore only a few different box types, e-commerce is subject to a great diversity of boxes since each one buys what it wishes with different brands from all around the world. Consequently, observing containers or pallets filled with objects of different sizes is more common in this sector. The problem of multi packages dimensions is the one we want to face in this work. More details will come later on this particular point.

We believe it is possible to develop an algorithm that would consider all the parcels that need to be put in the container or pallet and place them so that we can store the most boxes in the determined space. This problem is quite similar to a three-dimensional game of Tetris. A fun way to better understand the problem is the board game TEAM UP, which can be played with several players. This small game's objective is quite similar to what we are trying to implement here. The goal is to fill a pallet with rectangular boxes. These boxes can have specific constraints, such as a fixed rotation or positions in the 3D environment. This game might help us determine an algorithm to load boxes onto the pallet. Such an algorithm could enable any transport company to reduce the number of transportations they make. This reduction in the number of journeys needed to transport the items would imply many advantages and partly respond to the various problems raised in the introduction. Thus, this would result in a reduction in the number of vehicles required and the number of goods transported. Thus, this would also have a positive impact on the ecological footprint of the company.

Furthermore, the more parcels that can be stored in a container, the faster customers can receive their delivery. A faster and more efficient loading process would increase customers' satisfaction, who, as mentioned above, are now more and more demanding concerning delivery waiting times. In order to test and improve the bin packing algorithm, a box instances generator will be developed to provide an unlimited number of datasets.

This algorithm is only part of the solution that we want to propose in this master thesis. Indeed, we believe that an augmented reality tool would allow faster assembly of pallets or containers. This tool would take the form of a computer program running on an augmented reality headset, the Hololens 2. This headset, developed and marketed by Microsoft, is a headset allowing numerous interactions with users by displaying holograms in front of their eyes. The headset assembles a circle placed on top of the user's head and a visor deployed on the user's front face. Holograms can be displayed on this visor, and it is possible to manipulate these different holograms, either by code or by gestual interactions. A complete computer is therefore present inside this device. Such software would also enable better pallet construction and packaging. A detailed description of the tools is given in the dedicated section.

The program that we wish to develop with this tool aims at helping the user during the assembly of a pallet or a container. It will therefore be possible for the user to quickly identify which boxes to take and where to place them precisely through holograms. To perform these actions, the user will be guided and see, for example, an arrow above the box to take, indicating that this box is the one to take in hand. Once the box is in hand, the user will only have to look at the pallet or the container and will see a hologram of the box at the precise location to place it. This hologram will perfectly match the dimensions of the box taken.

Finally, a web interface will be developed to easily test and use the provided modules. A study on the database is also part of the work. Indeed, the way of storing data, their location, but also their structure can significantly impact this project.

2 State of the art

The present chapter aims to review the literature for all fields of study involved in this master thesis. The first one is focusing generally around the Cutting and Packing (C&P) problem, and more specifically on the container loading problem. To better understand what this section is about, the typology published by Wäscher et al. (2007) is described and used throuhout all this research. A second part of this state of the art review focuses on the mixed reality. The following literature review is not intended to be exhaustive, but rather to give a general picture of the state of the art in this field.

2.1 Container loading problem

2.1.1 Cutting and packing problem: a general approach

The cutting and packing (C&P) problem is a field of study for which the number of publications has increased considerably in recent decades (Wäscher et al., 2007). However, the container loading problem, which is the problem we want to study in this work, is, in fact, only a specific part of the C&P problem.

In 2007, Wäscher et al. published a paper providing a comprehensive typology of the study of cutting and packing algorithms. Their paper is based on the following premise: a comprehensive typology, especially on operations research (OR) problems, can provide the basis for structural analysis of the underlying problems, but also the identification and definition of standard problems, models, and algorithms development, etc. Therefore, standardization of terms and notations can facilitate research in this vast field of study, cutting and packing (C&P). Communication between researchers is also facilitated in this way, and access to relevant information is a benefit of the work provided by the author. Consequently, the terms used in this chapter, and more generally in this master thesis, are primarily taken from this article, which is intended as a base of terms to be used in this field. Defining a typology in a sector is not an easy task. Indeed, (Dyckhoff, 1990) tried to introduce a basis, but this was not necessarily accepted internationally (Wäscher et al., 2007). Moreover, many studies have appeared on the subject in the meantime and the typology proposed at the time became progressively insufficient in view of recent research. Thus, the typology proposed by Wäscher et al. is more up-to-date, more comprehensive, and also takes into consideration the "blank spots", i.e., areas for which there has been no or very little research done of the corresponding literature. Furthermore, this improved typology provides a consistent system of problem types, allowing compartmentalization of the sub-problems. It is, therefore, possible to categorize all known C&P problems and their corresponding literature.

According to Wäscher et al. (2007), Cutting and Packing problems have an identical structure in common :

Given are two sets of elements, namely

- *a set of large objects (input, supply) and*
- *a set of small items (output, demand),*

which are defined exhaustively in one, two, three or an even larger number (n) of geometric dimensions. Select some or all small items, group them into one or more subsets and assign each of the resulting subsets to one of the large objects such that the geometric condition holds, i.e., the small items of each subset have to be laid out on the corresponding large object such that

- *all small items of the subset lie entirely within the large object and*
- *the small items do not overlap,*

and a given (single-dimensional or multi-dimensional) objective function is optimized

C&P problems are, indeed, characterized by the same input (supply) and the same output (demand). The input corresponds to a set of large rectangles (for the two-dimensions problem), also called stock-sheet. In our application context, this input might be a container, bin, or pallet on which boxes must be placed. It is defined by a width and a height (2D problem). The output coincides with the small rectangles, also called items. In our application context, this corresponds to the boxes. A solution to the problem might only use part of the input or part of the output.

In order to classify the problem faced, Wäscher et al. (2007) defines five criteria to categorize the problem. These crtieria are an improvement of the four-criteria classification proposed by Dyckhoff (1990). If only part of the criteria are evaluated in a problem, this is considered as a problem extension rather than a problem type.

Dimensionality corresponds to the distinction between a one-, two-, or three-dimensional problem. Some researchers (e.g., Lins, Lins, & Morabito, 2002) have studied more in detail the n-dimension problem with cases where n >3, but this is considered a variant by Wäscher et al. (2007).

Kind of assignment: two distinctions are made for a given problem for this criteria: output value maximization and input value minimization. In the output value maximization, the set of large items is of fixed size and is insufficient to accommodate the set of small items. Therefore all large items must be used, and only a set of the small items is used. The problem becomes a selection problem of the small item, with the objective of maximizing the value of these small items. Conversely, in the input value minimization, we are once again given a set of large objects and a set of small items, but we are not limited in the number of large objects. Therefore all items must fit in, and the objective is to assign all of them to a selection of the large objects of minimal value. It is consequently not a selection problem regarding the items.

The term “value” used in describing these two sub-categories is quite extensive. This general term can become more specific depending on the problem faced and can, for example, represent the cost, revenue, weight, or quantity of materials. It can also be proportional to the object's size, and the objective function consequently considers the length, area, or volume in this last case (for the one-, two-, or three-dimensional problem, respectively).

Assortment of small items: three cases are distinguished in the typology concerning the small items: all identical small items, a weakly heterogeneous assortment of small items, and a strongly heterogeneous assortment of small items. A weakly heterogeneous assortment can group items into relatively few classes for each element with a similar shape and size. In contrast, the strongly heterogeneous assortment is characterized by the fact that only a few to no elements have identical characteristics.

Assortment of large objects: The typology considers two distinct cases, the first where there is only one large object and the second where there are multiple large objects. For the second case, three sub-classes are considered, similar to the assortment of small items, with all identical large objects, a weakly heterogeneous assortment, and a strongly heterogeneous assortment.

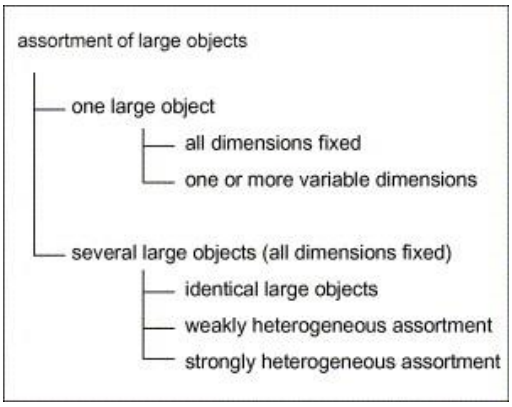


Figure 1 Assortment of large object criterion (from Wäscher et al., (2007)

Shape of small items: The typology distinguishes two types of items, the regular ones (rectangle, circle, etc.) and the irregular ones.

The decision of the criteria "kind of assignment" and "assortment of small items" defines the *basic problem type*. These two choices enable six different combinations, represented as a hierarchical structure as shown in Figure 2.

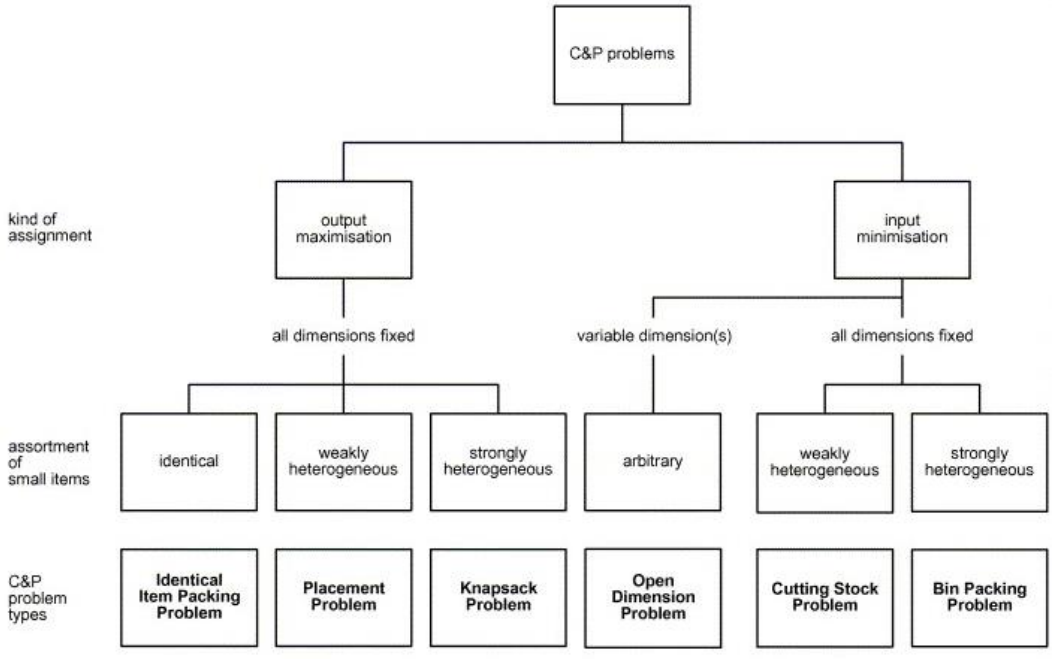


Figure 2 C&P problem types (from Wäscher et al., (2007)

Because we decided at the beginning of this work that the goal was to develop an algorithm able to provide a suitable solution for putting pre-encoded-size rectangular-shaped boxes into a single fixed-size rectangular-shaped large object, the corresponding problem type that we are facing appears to be the knapsack problem. Indeed, because we have a single large object that we can not expand, the objective is to select the most appropriate set of boxes so that the value of the accommodated boxes is maximized, i.e., the space used is maximized. This statement implies that we are facing an output maximization concerning the "kind of assignment" criterion. The "assortment of small items" is deducted from the problem we want to face in this work. As a reminder, the tool developed in this work aims to face issues in the e-commerce industry, where strongly heterogeneous boxes might be present. Therefore, the objective is to develop a greedy algorithm that can handle identical, weakly heterogeneous, and strongly heterogeneous boxes. However, from the start of the project development, the problem has been oriented to solve the most complex case, i.e., the strongly heterogeneous items assortment. Therefore, we conclude that the problem faced fits in the knapsack problem category.

The criteria "assortment of large objects" enables us to clarify the *intermediate problem type*. If we consider the five criteria, this constitutes the *refined problem type.*

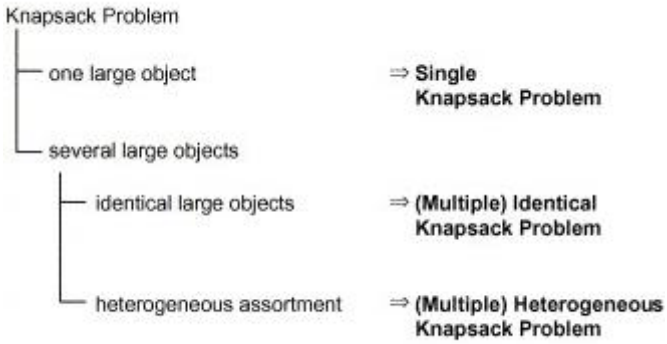


Figure 3 Intermediate types of the knapsack problem (from Wäscher et al., (2007))

Looking at the figure above (Figure 3), the intermediate problem type studied in this work is the Single Knapsack Problem. Considering the two last criteria, "assortment of large objects" and "dimensionality", this work's problem full name is the *three-dimensional rectangular single knapsack problem with one large object of fixed-size dimensions*.

According to Wäscher et al. (2007), if we replace some of the assumptions (e.g., multiple objectives, n-dimensions problem with n>3, etc.), this will lead to problems type that will be considered as *problem variants.* Details on each problem type (basic, intermediate, and refined) are provided in the typology written by Wäsher et al. (2007).

By studying only the terms used in this work, one could indeed have thought that the basic problem we are describing here is the bin packing problem. However, this one is not suitable because it imposes assigning the items in optimal ways in different large objects. This assignment problem is not studied in this work because this work assumes to have a single container, and this is why the bin packing problem, as defined by Wäsher et al. (2007), is not the basic problem in our case. Nevertheless, the term "bin packing problem" may still appear in this work, as the objective is still to pack different items into a bin, thus giving the name "bin packing problem". When this term is used in this work, it refers to the principle of assembling the boxes in a bin and not to the term defined in the topology. The term "container loading problem" might also be used as defined in (Bortfeldt & Wäscher, 2013). It refers to the same problem: small items are cargo, and large objects are containers. They stress, however, that

in a real-world application, the large object might just as well be a container or any loading space such as a truck or pallet, which may be loaded to a certain height.

2.1.1.1 Single Knapsack Problem

The Single Knapsack Problem (SKP) has been one of the most studied problems among all the ones presented in the Wäscher et al. (2007) typology. Indeed, at the time of publication, Wäscher et al. (2007) had identified no less than 87 scientific articles on this subject. Among these, 49 concerned the problem with one dimension, 25 with two, and 12 with three. In total, the SKP problem represents 19% of the research done on C&P problems.

The classic one-dimensional Knapsack Problem, also called the 0-1 Knapsack Problem (Martello et al., 2000), consists in packing a given set of items with weight and value as characteristics into a given knapsack which has a limited weight capacity. The objective is to maximize the total value of packed items. Variants exist, such as the Subset-Sum Problem (cf. Martello and Toth, 1990, pp. 105), where the weight of items corresponds to its value. This variant idea, assigning a characteristic of an item to its value, is interesting and will be addressed in the proposed solution. Another extension of the problem is the Multiconstraint (0-1) Knapsack Problem (Gavish & Pirkul, 1985; Drexl, 1988; Thiel & Voss, 1994). These scientific articles deal primarily with developing effective solution procedures for solving the multiconstraint knapsack problem with a single dimension using methods such as relaxation of the problem, simulated annealing concept, or genetic algorithms. Some of these concepts, such as genetic algorithms, will be discussed more in detail in the dedicated section.

When dealing with one more geometric dimension, the Knapsack Problem becomes the *Two Dimensional (Single Orthogonal) Knapsack Problem* (Caprara & Monaci, 2004; Healy & Moll, 1996), which aims at selecting and packing a maximum-profit subset of rectangles from a given set into another rectangle. These small rectangles can therefore be cut from the single large rectangle. When considering one more geometric dimension again, the problem gives rise to the *Three dimensional (Single Orthogonal) Knapsack Problem*, intending to pack rectangular-shaped boxes into a container. As mentioned before, this problem is the one addressed in this paper. We have, however, taken the liberty of developing the problem for lower dimensions above because the resolution methods have numerous similarities. This three-dimensional problem is sometimes addressed with a different name in the literature, such as *Knapsack Container Loading Problem (Pisinger, 2002)*, or simply *Container Loading Problem* (Bortfeldt & Gehring, 2001). It is sometimes even called the three-dimensional bin packing problem (Kang, Moon, & Wang, 2012), which is inacurate based on the typology proposed by Wäscher et al. (2007), the bin packing problem being another basic problem type of a C&P problem. Once again, we find in some of these articles the property that a value of an item is proportional to its size/volume. This property implies that the unused space of the container can be minimized. Finally, Fekete et al. (2007) developed an exact algorithm for higher-dimensional (n>3) orthogonal packing.

2.1.1.2 Related problem types inspiration

Among all the other problem types that Wäscher et al. (2007) define, some of them are actually close to the problem addressed in this paper, sometimes having only one criterion that differs, such as the number of large objects. We can, for example, take the *Single Stock-Size Cutting Stock Problem* (SSSCSP) or the *Single Bin Size Bin Packing Problem* (SBSBPP) (Lodi et al., 2002; Gzara et al., 2020)). These problems differ in the way that the "kind of assignment" criterion is set to "input minimization" while, in our case, it is set to "output maximization". This means that in the two problems mentioned here, all small items need to be packed into the large objects, and the goal is to minimize the number of large objects. In the SBSBPP problem, all containers have the same dimension.

This problem is, therefore, really close to the Single Knapsack Problem (SKP), and inspiration on algorithms and methods to handle this type of problem might just as well be helpful in the SKP problem.

2.1.2 Container loading constraints

More than 25 years ago, Bischoff and Ratcliff published the paper "*Issues in the development of approaches to container loading" (1995),* in which they argued that the container loading problem has so far encountered existing approaches that are each only applicable to a tiny part of the spectrum of situations encountered in practice (p. 377). They further claimed "… that a number of factors which are frequently of importance in practical situations have not received sufficient attention in the OR literature." (p. 378). Since then, many authors have studied problems including more and more constraints. A state-of-the-art review has been published by Bortfeldt & Wäscher in *Constraints in container loading – A state-of-the-art review* (2013) identifying factors that need to be considered when dealing with container loading problems.

Four years later, Celia Paquay's doctoral thesis (Paquay, 2017) again tackles the different constraints that can be addressed in the container loading problem. She brings not only a complete literature review but also an solution considering an impressive number of constraints in play. The following paragraphs describe the different constraints that can be considered when working on the container loading problem.

A first distinction is made between the geometric and the specific constraints. The typology from Wäscher et al. (2007) only considered the geometric constraints.

2.1.2.1 Geometric constraints

The geometric constraints involve the geometric assignment problem, in which small three-dimensional items (boxes) need to be assigned and packed into a large rectangular-shaped object (bin). If assigned in the bin, these items must lie entirely within the container and can not overlap. Furthermore, they are assumed to be placed orthogonally, such that the edges of the boxes are either parallel or perpendicular to those of the container. While Paquay's work focuses on filling a cargo plane and thus uses ULDs (Unit Load Devices) that are not perfectly cubic in shape, this work generally addresses the container loading problem, and we, therefore, assume that the bin is rectangular in shape.

2.1.2.2 Specific constraints

The specific constraints concern all the other constraints that can be considered when filling a container. It is these constraints that Bischoff and Ratcliff (1995) addressed by claiming that they were not sufficiently taken into account in the research conducted in this field.

Container weight limit: In real-life applications of this problem, it is not uncommon for the container to have a maximum authorized weight and therefore not be able to support items whose total weight exceeds this maximum authorized weight. Thus, each item has a fixed weight as a characteristic. Once the maximum allowed weight has been reached during loading, adding new items without removing some already assigned is no longer possible. Weight limit constraints can be easily modeled as linear Knapsack constraints, where the sum of the weights of the loaded items has to be lower or equal to a limit imposed by the container. Checking if a solution is feasible or not is therefore quite

straightforward. It is also one of the most commonly discussed constraint in the literature (e.g., Terno et al., (2000), Chan et al., (2006)). It is worth noting that the weight constraints are particularly involved in a similar C&P problem, namely the vehicle routing problem (Gendreau et al., 2006). This problem involves finding an optimal set of routes for a fleet of vehicles that must deliver items to geographically dispersed customers. Items are stored in cargo and are characterized by their weight, and each vehicle has a weight carrying capacity (weight limit). The objective is to find a set of routes that minimizes the total distance traveled by the fleet of vehicles while taking into account the weight carrying capacity of each vehicle.

In this work, each item is characterized by its weight, and the container has a maximum weight limit it can handle. Once the limit is reached, no more boxes can be added without already placed ones removed. More detail will be provided in the proposed solution.

Weight distribution constraints in the container: Also called load balance constraints (cf. Bortfeldt & Gehring, 2001), these constraints ensure that the weight distribution is evenly distributed as much as possible. A balanced-weight container is desirable, especially when the container is moving because it lowers the risk of collapsing and, as a result, the risk of an accident. According to Bortfeldt & Wäscher (2013), the weight distribution constraints are almost as frequently considered as weight constraints in the literature (11,7% for the weight distributions while 14.1% for the weight limit). Several authors have addressed these constraints in the way that the center of gravity of the load should be close to the geometrical center of the container (cf. Davies & Bischoff, (1999), Balakirsky et al., (2010)) or should at least not exceed a certain distance (Bortfeldt & Gehring, 2001).

Considering the three-dimensionality of this problem is not always required. For example, Eley (2002) considers the weight distribution constraints simply along the largest side of the container. Chen et al. (1995) similarly introduced a mixed linear optimization model for the input minimization and output maximization problem types in which he demonstrated how one-dimensional weight distribution constraints could be applied to this model. For real-world applications, we can take the example of aircraft loading, where the balance is much more critical in the longitudinal length than the lateral length (Davies & Bischoff, 1999, p. 509). However, this does not apply to all applications, and we sometimes need to involve more dimensions to generate acceptable results.

This master thesis does not handle the weight distribution constraints.

Loading priorities for the items: Sometimes, some items have a higher priority over others in the loading sequence. Indeed, in some cases, some items might be left behind if the container can not gather all items at once. This is typically the case for output maximization problems, where there is a limited number of large objects, and the objective is to decide which small items will be taken and which ones will be left behind. Therefore, this constraint only applies to output maximization problems, which is the case in this work. In practice, some items might have higher priority over others (Junqueira et al., 2012), for example, when dealing with limited shelf life products or items with delivery deadlines. However, according to Bortfeldt & Wäscher (2013), this constraint is rarely considered in the design of algorithms for container loading, having only 1.8% of the papers addressing this issue.

The priority can be characterized in different ways. Ren et al. (2011) introduced a tree-search algorithm based on a greedy heuristic in order to solve the container loading problem with shipment priority. This shipment priority divides the items into high- and low-priority, and each item must be in either one of these two categories. More categories would be possible if needed. Each item would be

assigned to a priority category, and the problem would become more complex. We have not found any article mentioning this path and providing an algorithm with more than two categories.

This paper considers part of the loading priority constraint as it provides several loading priority algorithms. More detail will be provided in section 3.1.

Orientation constraints for the items: In the three-dimensional version of the problem, items are characterized by width, depth, and height for the dimensions. These boxes are assumed to be placed orthogonally, i.e., the edges of the box are either perpendicular or parallel to the edges of the container. Considering these two pieces of information, it is possible to rotate an item into six different rotations. It however sometimes happens that an item cannot be rotated into one or multiple rotations due to the content of the item. For example, a fragile item might be allowed only to rotate horizontally but not vertically, its top side having to remain constantly on top. These constraints are called orientation constraints and limit some orientations for specific items. A list of possible orientations thus features each item it can take.

According to Bortfeldt & Wäscher (2013), the orientation constraints represent the most frequently addressed constraint type in the literature, with over more than 70 % (70.6% exactly) of papers dealing with this aspect. They further divide this constraint into five different cases. Case 1 considers that only a single orientation is permitted for each box, meaning that the box can not be rotated. Case 2 involved that only a single vertical orientation is allowed for each box, and no restriction is done regarding the horizontal direction. Case 3 assumes that there is no restriction regarding the vertical orientation of the box, but up to two vertical orientations may be prohibited for each box. The box is free in the horizontal direction. Then, case 4 applies no general restriction with respect to the orientation for both the horizontal and vertical directions. However, up to five orientations may be forbidden for each box. Finally, case 5 implies that there is no constraint with respect to the boxes' orientation, neither vertically nor horizontally. It is this case that enables to get the most possible combinations because boxes have the most extensive degree of freedom.

Case 5 will be used for this work, meaning that boxes can freely rotate in each direction. No constraints are applied to individual boxes.

Staking constraints: Also known as load-bearing constraints (cf. Junqueira et al., (2012)), staking constraints define the number of boxes that can be placed on top of each other. More precisely, it determines the maximum pressure that some boxes might handle on top of them such that no damage is caused. Sometimes, the strength of a box is not only measured as the strength of the top face but is instead determined by the strength of the box side walls (Bischoff & Ratcliff, 1995), which provides a more appropriate measure (Bortfeldt & Wäscher, 2013). The solidity of a box is also affected by the humidity, load duration, and the content of the box. Indeed, a box filled with solid content allows more stacking than a box that is only partially filled.

Several strategies have been developed in the literature. Terno et al. (2000) prohibit a particular box of type *i* from being placed on a particular box of type *j*. This way, it is, for example, possible to prohibit large boxes from being placed on top of small boxes. Junqueira et al. (2012) define a maximum number of boxes that can be placed on top of each box. Paquay (2017) takes a relatively different method to handle the staking constraints. The author defines boxes as either fragile or not, with fragile boxes being unable to support any box on top of it. Finally, a recent and interesting approach developed by Gzara et al. (2020) satisfies the load-bearing constraints through a graph representation to trace load distribution. Indeed, using a graph distribution enables the collection of perfect information about the

boxes that exert pressure on the boxes below. A better comprehension can be deducted from Figure 4.

These constraints are not considered in this work.

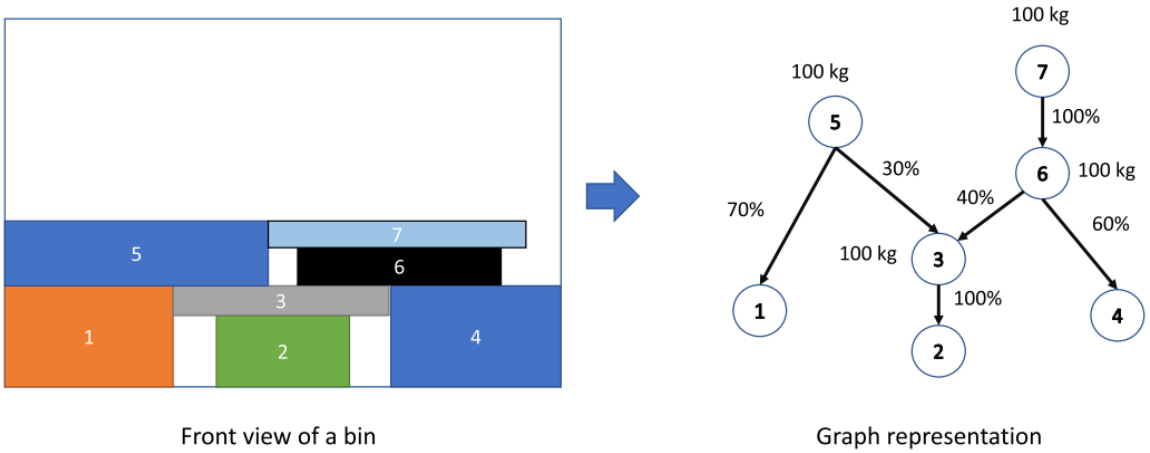


Figure 4 graph representation approach for staking constraints (from Gzara et al., (2020))

Complete shipment constraints: Sometimes, items can be split into subsets. For example, multiple pieces of furniture are packed separately but need to be sent to the same location and assembled on a customer's site. The complete shipment constraint implies that all boxes from a subset must either be all sent to their destination, or none. It implies that if one element of a subset is sent, then all the other elements of the same subset must also be sent. In the opposite case, if one box of a subset is not sent, then all the other boxes of the same subset must also not be sent. Like for the loading priority constraints, these constraints are only applicable for output maximization cutting and packing problems. Indeed, boxes might be left behind in the case of output maximization, while it is impossible in input minimization problems. According to Bortfeldt & Wäscher (2013), this constraint is rarely considered in the research field, having only 0.6% of the research involving this topic.

This constraint is not taken into consideration in this work.

Allocation constraints: In the case where several containers are available, the algorithm needs to assign each item to the container in order to determine which box goes into which container. These constraints are therefore only applicable when multiple containers are involved in the problem. These constraints might have some connexion with the previous constraints, the complete shipment constraints. Indeed, if items are grouped into subsets, this is sometimes because they need to be sent to the same destination. This implies that they probably need to be allocated into the same container. This specific case is called the connectivity constraints by Liu et al. (2011). In the reverse case, it is also possible that some items can not lie in the same container as other items. This case is called the separation constraints. Eley (2003) uses the example of food and perfume that should typically not be loaded into the same container.

This work considers the output maximization problem with a single container. The allocation constraints are thereby not relevant in this paper.

Positioning constraints: Sometimes called the placement condition (Terno et al., 2000), these constraints enforce the items' location in relative or absolute terms. If relative positioning constraints are used, this requires that some boxes may or may not be close to each other. Terno et al. (2000) give the example of items that are not allowed to be loaded on top of other pieces. Another example is that materials that might have chemical reactions with other material types must be far from each other. Concerning absolute positioning, Terno et al. (2000) provide examples of some pieces that must be loaded on the bottom of the container or reversely on the top. According to Bortfeldt & Wäscher (2013), only 2.5% of papers address the absolute positioning constraints and 4.3% for the relative positioning constraints.

A combination of absolute and relative positioning gives rise to multi-drop constraints (Bischoff & Ratcliff, 1995, p. 379). These constraints appear when a combination of the vehicle routing problem and the packing problem is made. As a reminder, the vehicle routing problem consists in delivering multiple items to different geographical locations in the most optimal way. The items should be arranged so that the first items removed from the vehicle are located closest to the door. This way, unnecessary unloading and reloading operations are avoided (Junqueira et al., 2012).

This type of constraint is not taken into account in this work.

Complexity constraints: This type of constraint refers to the maximum complexity of a solution. Complex loading patterns may, for example, not be accepted if the container loading is done manually (Bortfeldt & Wäscher, 2013). The human operator needs indeed to understand the instructions he is given. These instructions might be a visual indication and these need to be clear. The time required to build the solution is also relevant, and some real-world applications might prefer a fast-loading solution with fewer assurances regarding the stability or other constraints discussed above.

This type of constraint is not studied in this paper.

Stability constraints: Load stability is often considered in the literature as one of the most critical constraints in container packing problems after the geometric constraints (Eley, 2002; Bortfeldt & Wäscher, 2013). Indeed, finding a solution that is not even stable in a static position, i.e., no force applied on the structure except gravity, is not convenient for any real-world application. Furthermore, an unstable load might cause damage to the cargo and sometimes even injuries to personnel during the loading or unloading operations. Stability becomes even more critical if the pallet, container, or vehicle transporting the merchandise is moved and, as a result, lateral forces might be applied to the structure. This might as well consequently cause damage during transport and injuries to personnel.

However, some authors claim that the load stability constraint is not as important as it seems and does not consider it explicitly in their publication on container loading (Pisinger, 2002; Parreno et al., 2008). Instead, they claim that "… stability becomes an immediate consequence of the corresponding load compactness when high container space utilization can be guaranteed" (Bortfeldt & Wäscher, 2013). This is typically true in the output maximization problem type since the number of containers is limited, and only a subset of the items is picked. Practically, there also exists solutions to ensure stability, such as shrinking foil which enables the whole structure from not falling apart.

Two distinctions can be made concerning load stability: vertical and horizontal stability. Vertical stability ensures that items are not falling apart in the situation when the container is not moving and, consequently, only when the load is to withstand the gravity force (Junqueira et al., 2012). Horizontal stability deals with the fact that items do not significantly shift when moving the container (Bischoff &

Ratcliff, 1995). It, therefore, refers to the capacity of an item to remain at its position when inertia is exerted.

Vertical stability issues are usually handled by forcing the base of an item to be partially or fully supported by a flat base, either the floor of the container or the top face of one or several other items. According to Bortfeldt & Wäscher (2013), around 50% of all papers in which stability is a studied constraint stipulate that full support is required in their solution. This means that when a box is placed, it needs a flat base of at least the same size of the base of the box. In the other articles, i.e., when full support is not required, at least a minimum fraction of partial support is specified. For example, concerning the pallet loading problem, Hemminki et al. (1998, p2227) claim that complete support stability is desirable, but if the last stage of packing is the wrapping of the whole pallet with plastic folium, then only 70% of base support is generally sufficient in practice.

Alternatively, Mack et al. (2004) make other assumptions. Two restrictions must be respected to guarantee vertical stability. The first one is the same as the one mentioned above, which is a certain percentage of support that must be observed. The second one concerns the center of gravity of each box that must lay down on the support in order to avoid the box tipping over. The center of gravity of each box is assumed to be their geometric center. These conditions might solve the problem if only two boxes are stacked on each other, but if we get a pile of several boxes, the stability of this pile might not be guaranteed, as shown in Figure 5. To avoid these unstable box layers, Mack et al. (2004) add the condition that a box is stable only if the supported part of each box must also be calculated in relation to the container floor. In Figure 5, the grey box's center of gravity is supported by the underlying box but not by the container floor.

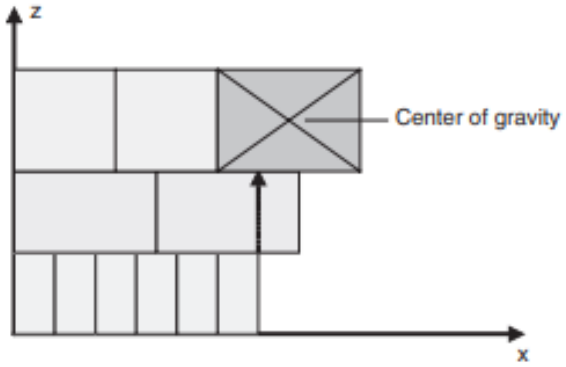


Figure 5. Stability constraints: avoiding unstable box layers (from Mack et al., (2004))

Another interesting approach to vertical stability is the one considered by De Castro Silva et al. (2003). In their paper *A greedy search for the three-dimensional bin packing problem: the packing static stability case*, they once again assume that the geometric center of each box coincides with the center of gravity. In addition, they assume that all boxes have the same density. Then, considering forces applied to each box by the other ones, the author formulates equilibrium conditions to retrieve each box's center of gravity. This approach enables the structure presented in Figure 6 to be considered stable, while this would have been considered unstable using the previous approach by Mack et al. (2004).

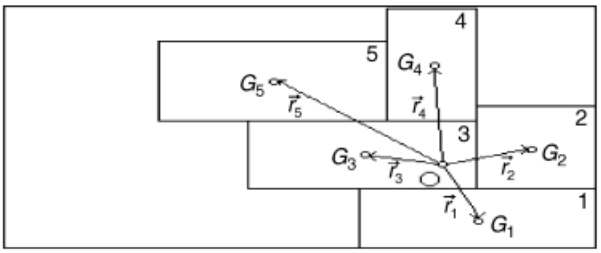


Figure 6. Gravity center of the packing, lateral view of the bin (from De Castro Silva et al., (2003))

Finally, a recent article focusing on developing an algorithm that can fully handle vertical stability is provided by Ramos et al. (2016). The study aims at developing a physical packing sequence algorithm for the container loading problem with static mechanical equilibrium conditions. In the article, an interesting approach worth mentioning consists of retrieving the polygon support when dropping a box on the load. This method is declined into three different cases:

1. The support polygon of the box to place is the container floor.

2. The center of gravity of the box to place lies in the interior of a support polygon (the top surface of the item below).

3. The center of gravity of the box to place lies in the interior of a support polygon which is a convex polygon defined by the convex hull of the vertices of the polygon generated by the intersection of the box to place with its supporting boxes (Figure 7).

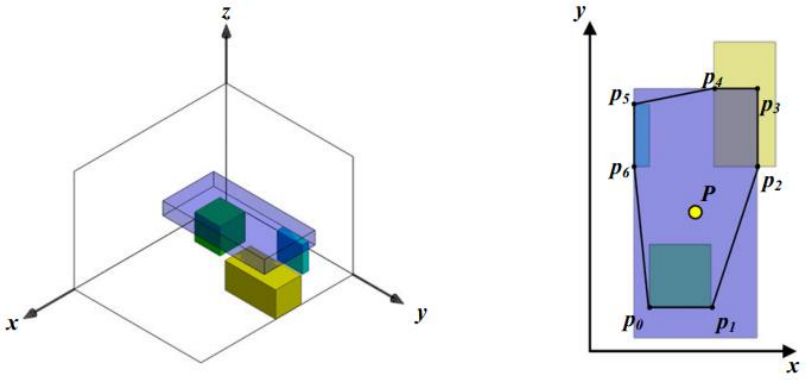


Figure 7 Example support polygon for case 3 (from Ramos et al., 2016)

Horizontal stability ensures that boxes do not shift significantly while the container moves. It can be ensured if each side of a box that is packed has either another box packed beside it or a container wall. To evaluate the lateral support of a load, Bischoff & Ratcliff (1995) retrieve the percentage of boxes that are not in contact with either the wall of the container or another item on at least three of the four sides. Liu et al. (2011) developed a slight variant where only one box surface is required to be in contact with either the wall of the container or another item.

Another solution consists of "interlocking" the various box layers. Carpenter & Dowsland (1985) wrote the paper *Practical considerations of the pallet-loading problem*, describing the interlocking principle in detail. Although this article was published almost 40 years ago, it remains a reference concerning this subject. In this paper, they introduced three criteria in order to ensure the interlock of the layers of a load (cf. Bischoff (1991), Bortfeldt & Wäscher, (2013)):

1. Supportive criterion: The bottom base of each box must be placed on the top face of at least two distinct boxes
2. Base contact criterion: A percentage of the base area of each box must at least be supported by the layer below (or the container floor).
3. Non-guillotine criterion: The length of a seam (guillotine cut) cannot cross the entire structure and exceed a specific maximum percentage of the length of the side.

Another approach to improve stability consists in using postprocessing procedures once the load is virtually completed (Bortfeldt & Wäscher, 2013). More concretely, this technique compacts the structure and removes the small unnecessary gaps that might occur between boxes.

Finally, Terno et al. (2000) emphasize that algorithms that tend to construct "tower structures", i.e., structures where boxes are placed on top of each other, and no interlocking is present, tend to be unstable for horizontal stability.

2.1.3 Container loading algorithms

This section aims to develop different types of algorithms used in the literature to solve the container loading problem in general. We, therefore, do not only explain algorithms for the single knapsack problem but for all types of algorithms where the objective is to place items into large objects.

Zhao et al. (2016) present a comparative review of 3D container loading problems that aims at being a complementary work to the literature review proposed by Bortfeldt & Wäscher (2013). This comparative review focuses on the design and implementation of solution methodologies for solving the container loading problem, where this term is used in its broadest sense. The authors define two solutions categories: placement & improvement heuristics and the exact methods.

2.1.3.1 Placement heuristics

Placement heuristic approaches, also known as construction heuristics, consist of defining a mechanism to decide where to load boxes into the container. They can be used either to generate a first solution or to solve the overall problem. This method can be declined in a wide range of approaches for the container loading problem. For example, when the set of boxes tends to be weakly heterogeneous, the favored algorithms are the *wall building* and the *layer building*. In the opposite case, i.e., when boxes are strongly heterogeneous, the authors (Zhao et al., 2016) advices that boxes are to be placed individually at a time.

Wall building and layer building algorithms arrange boxes of the same type in rows or columns to fill either one side of the container or the floor. Once a feasible solution is deducted from the empty space, i.e., the container can accommodate a wall or layer of boxes, the remaining space is considered as a new container of a smaller size. In both of these approaches, the objective is to group boxes into subsets to generate flat packing faces.

Another placement heuristic consists in identifying placement points, i.e., 3D coordinates where the next box will be placed. Martello et al. (The Three-Dimensional Bin Packing Problem, 2000) called these points corner points, while it was later adopted as extreme points by Crainic et al. (2008). These points are more precisely candidate placement positions for placing the boxes that are not packed yet. These points can be obtained by identifying the corners of the last box that has been placed. Figure 8 illustrates extreme points in 3D and 2D packings (from Crainic et al. (2008)).

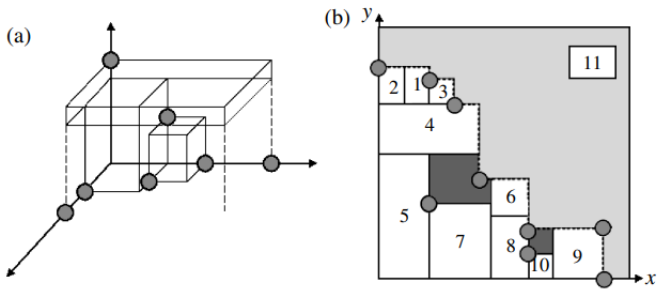


Figure 8: Extreme points in 3D (a) and 2D (b) (from Crainic et al. (2008))

The two approaches described above, wall- & layer-building and individual box placement, focus on how to pack boxes into the container. Along with determining the placement approach comes the decision of which box type to pack next. Standard techniques are called *pre-determined ordering* and *dynamic ordering*. The pre-determined ordering technique involves sorting boxes based on criteria such as the box volume, box base surface, box type, etc. This sorting is executed only once, and this execution comes before the placement of any box. The dynamic ordering can, on the contrary, change the order of boxes during the load process, and tend to be based on the usable space remaining after placing a box. Still according to Zhao et al. (2016), there is no definitive answer as to which decision technique is the best between static and dynamic ordering. However, the authors note that sorting by volume is a very common characteristic. In their paper, they then claim to have identified twelve static sequencing rules and five dynamic sequencing rules.

2.1.3.2 Improvement heuristics

Placement strategies such as the one described above will provide, in most cases, fast and suitable solutions with reasonable quality (Zhao et al., 2016). However, it is possible to broaden the search space using improvement heuristics, which can usually provide a significant gain. Usually, they make neighborhood moves to find better solutions and work with one or more solutions. These neighborhood moves can vary from simple neighborhood structures and acceptance criteria that only accept improving moves to complex and varying neighborhoods and acceptance criteria that are able to find many local optima. Therefore, improvement heuristics enhances the solutions provided by placement heuristics. The improvement heuristics described below are not intended to be exhaustive.

2.1.3.2.1 Genetic algorithms

Permutation of boxes is a technique that is present in many placement heuristics. These heuristics are therefore dependent on a sorted order. This type of representation is used quite often in what is called genetic algorithms. Genetic algorithms correspond to search heuristics inspired by Charles Darwin's theory (Mallawaarachchi, 2017). This theory establishes the process of natural selection, where individuals that are the most adapted to their environment are selected for reproduction to produce the next generation. The idea is similar to a genetic algorithm where the fittest individuals are selected. Natural selection starts with the selection of the fittest individuals from a population. They then produce offspring which inherit most of the characteristics of their parents. These child entities are part of a new generation. If the parents of a child have good characteristics, then the child has a reasonable probability of inheriting these properties, which gives him a better chance of surviving. This process of generating a new generation keeps on iterating, and progressively, generation after generation, the fittest individuals can be found.

The process starts with a set of solutions for the problem to solve. This set is called the initial *population*. A set of parameters characterize each individual. These parameters are called *genes*, and genes can be joined into a string to form what is called a *chromosome*, which corresponds to a solution. Binary values are often used for the genes in the simpler versions of the algorithm, but more complex values can be used.

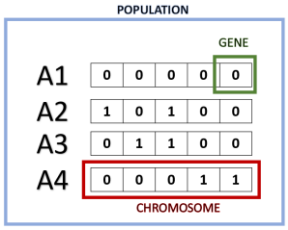


Figure 9 Genetic algorithm terms

A *fitness function* needs to be defined to test how fit an individual is. This function gives a fitness score for each chromosome, which is the criteria for determining the probability that an individual is selected for reproduction or not. Then, once the individuals are selected for reproduction, they are grouped into pairs (parents), and techniques such as crossover and mutations are executed, which basically define the genes of the children individuals.

Now that the most basic form of a genetic algorithm has been explained, it is time to develop how this type of algorithm is used in container loading problems. A first work that combines the container loading problem with a genetic algorithm was provided by Hemminki J. (1994, cited in Zhao et al., 2016). The author uses the wall building approach such as the one described above, and uses a genetic algorithm in order to determine the width of the walls. The author does not use boolean values to characterize the genes, as described in the basic definition above. Instead, each chromosome is characterized by a set of integers where each gene represents the packing strategy used in the corresponding wall, and the combination of all walls corresponds to the container. A random selection is carried on the parent chromosomes, and the fitness function directly impacts the selection chance of each chromosome. Crossover or mutation techniques are then applied using both parents and give, as a result, two child chromosomes. After several iterations, the best combinations of walls are obtained through the evolving process.

Several authors were subsequently interested in this type of algorithm to solve the container loading problem (see Thiel & Voss, 1994; Bortfeldt & Gehring, 2001; Kang et al., 2012). Various solutions exist in these works, sometimes using genes to represent boxes, walls, layers, placement strategies, etc. Compared to generating all solutions, the advantage of a genetic algorithm is the computational time. Genetic algorithms are indeed known to be highly efficient, at least if the fitness function is relatively fast to obtain the fitness score of a solution.

2.1.3.2.2 Tabu Search

Tabu search is another way to solve the 3D packing problem. It is one of the most popular metaheuristics in combinatorial optimization, and this popularity also applies to container loading problems. To understand tabu search, we must first define the term *local search*. Local (neighborhood) search starts with a solution for the problem and checks its immediate neighbors, i.e., solutions that are almost similar except for some minor details. The goal of looking at neighbors is to find an improved solution. The problem with local search methods is that it tends to become stuck in a local optimum. Tabu search improves this technique's performance by allowing a relaxation of the basic rule. At each step, it is allowed to take a worse solution if no improvement is possible (we are therefore at a local optimum). In addition, prohibitions (hence the term *tabu*) are added to avoid any search from looking

at previously-visited solutions. The implementation of a tabu search, therefore, stores already visited solutions which enables to look at unvisited solutions once we reach a local optimum.

Tabu search algorithm can be used for the container loading problem. Bortfeldt and Gehring (cited in Zhao et al., 2016) employed the wall building approach in combination with a tabu search algorithm for loading a weakly heterogeneous set of boxes into a single container. As described in the description of the tabu search algorithm, it is first required to start with a solution. Therefore, Bortfeldt and Gehring start with a solution that is generated using a basic heuristic. They then use tabu search on the packing sequence and fillable packing space. Neighborhood moves consist in deviating slightly from the fillable space order. They consequently use a tabu list that holds the packing sequence of the neighbor.

Like genetic algorithms, tabu search techniques can become more complex such as simulated annealing algorithms and hybrid search (Mack et al., 2004).

2.1.3.3 Exact methods

According to Zhao et al. (2016), the number of exact methods in 3D container loading is low compared to the number of available heuristics. The reason is the difficulty in representing possible patterns or considering practical packing constraints. Even if this is possible, the problem becomes the huge computationally expensive time required to generate such a solution when a large-scale problem is encountered. The 3D container loading problem is indeed an NP-hard problem. No algorithmic program is in a position to seek the precise solution for realistic size instances.

Exact methods approach can be modeled as mathematical formulations. Such techniques are, for example, the *mixed integer programming formulation*. In this approach, an objective function is formulated, aiming to minimize either the number of bins used or the empty space in a container. Each constraint that an author wants to include in the problem is also formulated mathematically, and then, with the use of a powerful solver, it is possible to obtain a feasible, optimal solution that handles all included constraints.

2.1.3.4 Guillotine cut algorithm

Another form of algorithm to solve the packing problem consists in using cutting patterns. The cutting problem consists in obtaining small rectangular pieces by cutting a rectangular-shaped object through a cut. Manufacturing businesses sometimes rely on cutting objects of multiple sizes in their production process (Martin et al., 2020). Indeed, the need to cut large objects into smaller ones is an industrial reality to satisfy internal and external demands. Examples of cutting operations are the cutting of wooden boards, steel bars, paper, etc. The glass industry is also particularly using this method. The approach is somewhat different from the ones explained in the placement heuristics section, where we started from an empty space and packed boxes progressively. With this approach, we start from a full bin-size object and cut it to obtain the required small items we need to pack. The objective is to minimize the total waste. Pioneering work on the subject was carried out by Gilmore and Gomory (1961; 1963; 1965). They have published numerous articles on this subject. Then, several publications have addressed some variants of the problems to handle a diverse application of packing and cutting algorithms.

In the cutting problem, the input corresponds to a large rectangle (for the two-dimensions problem), called a stock-sheet. The output coincides with the small rectangles, called items, which are the required objects to be obtained once all cuts are over. Once all cuts are carried out, there should exist

a configuration where all small items lie in the large rectangle, and the small items do not overlap with each other. To better understand the cutting problem, we will first focus only on the two-dimensional problem. In this problem, the original rectangle is a plain square or rectangle without any defect. The dimensions of the small shapes that need to be cut are already defined, and the challenge is to obtain these small rectangular shapes only through cuts, in an optimal manner in order to minimize the lost space. Several objectives can actually be relevant for this challenge, such as minimizing the lost space, maximizing the value of the items that are extracted, or minimizing the number of sheets required to get all the shapes needed. The objective can be defined with an objective function. Once again, these objectives differ depending on the problem addressed, which is, in our case, the Single Knapsack Problem, and, therefore, the objective is the output value maximization.

The cutting problem can be divided into sub-classes, known as the guillotine and non-guillotine. The difference lies in the cutting restriction, and these two sub-classes both lead to different solution methods. A guillotine cut (also known as edge-to-edge cut) corresponds to a cut from edge-to-edge in the object, meaning that there is a bisecting line going from one edge of the object to the opposite edge. With this problem, two parts are obtained every time a cut is made, and a new guillotine cut can be executed (Martin et al., 2020). In the non-guillotine class, such restrictions do not necessarily hold. In Figure 10a, it is indeed possible to get all the little rectangular shapes with guillotine cuts, while this is not possible in Figure 10b. A generally accepted assumption is that all cuts have zero width.

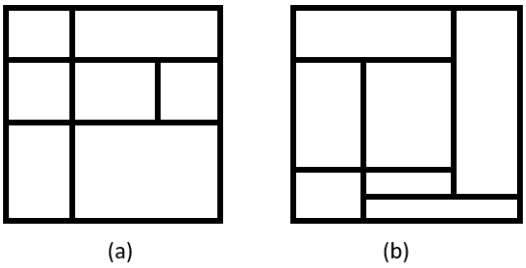


Figure 10 (a) Guillotine cut. (b) Non-guillotine cut

A considerable advantage of non-guillotine cut compared to guillotine cut is the horizontal stability. Indeed, having a slice that goes from one side to the other might conduct in an unstable load. This is not critical if the structure is not moved, as the only force applied is gravity, but it becomes critical when the container is moved, for example, on the road or in a cargo airplane. As a reminder, the use of non-guillotine cut is indeed refered in Carpenter & Dowsland (1985) as the third criterion in order to guarantee horizontal stability.

2.1.3.5 Offline and online algorithms

Two different versions of 3D packing algorithms exist, namely online and offline. The online version has the characteristic of not knowing the entire list of products to be stored and, therefore, only knows, for example, the first ten of the list. Progressively, as these first elements are placed, the following products will be known. On the other hand, the offline version requires knowing all the products to be stored in the container. The version implemented in this master thesis will be an offline version since most of the algorithms are based on this version first. It is later possible to implement online algorithms based on offline algorithms.

2.2 Mixed reality concept

The purpose of this section is to develop what exactly are augmented, virtual, and mixed reality, and how these technologies can be put to good use for industrial applications. We will more precisely study the impact on the supply chain and manufacturing sectors as these are the ones studied in this work. First, precise definitions are provided, followed by a brief history of the technology. A comparison is then conducted in order to distinguish virtual reality, augmented reality, and mixed reality, and a description of the potential of this technology to improve specific processes is made. A complete description of the Hololens tool is also provided.

2.2.1 Definitions

Augmented reality (AR) is an interactive experience that consists in adding elements such as sounds, 2D images, and 3D holograms on top of reality through the help of a computer system. It can enhance the real world with computer-generated perceptual information in real-time, sometimes across different sensory modalities, whether visual, haptic, somatosensory, or olfactory. AR can be described as the combination of three essential features: *a real and virtual world combination*, able to handle *real-time interactions* with both *3D and real objects*. The overlaid added elements can be either constructive, i.e., adding elements on top of reality, or destructive, i.e., masking the real environment. Augmented reality should not be confused with virtual and mixed reality terms.

Virtual reality (VR) consists of a computer-generated environment with which the user interacts. The virtual environment is made of scenes and objects that appear real from the user's perspective. This technology makes the immersion great because the user is in an entirely virtual world with which he can interact. The environment is perceived through a device known as a Virtual Reality headset or helmet.

Mixed reality can be defined as a mix of the real and virtual world (Microsoft, 2022). It is the merger in which physical and digital objects co-exist and interact. These interactions happen in real-time. Mixed reality, therefore, takes place not exclusively in neither the physical nor the virtual world and is a hybrid form of augmented and virtual reality. Several display technologies are available to experience mixed reality, and one of the most advanced tools is the Hololens 2 device. Augmented reality devices can be considered as holographic devices, while virtual reality devices as immersive devices. The difference is the visibility, the user being able to see through the display with a mixed reality device, while it is an opaque display for virtual reality devices. The Hololens is a mixed reality device in the way that it combines the holographic aspect of AR with the immersion experienced with VR. Practical applications are numerous with mixed reality, including design, entertainment, military, training, and remote work.

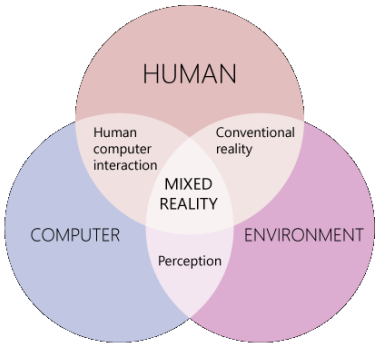


Figure 11 Mixed reality definition (From Microsoft, 2022)

2.2.2 Brief augmented/mixed reality history

The birth of such an innovation dates back to 1957, when Morton Heilig, a cinematographer, invented the Sensorama, a massive device that can deliver visuals, sounds, vibration, and smell to the viewer (Pesce, 2021). At the time, this was not controlled by a computer, but it was the first attempt to add

additional experiences. Then, in 1966, Ivan Sutherland, an American scientist, developed the first head-mounted display (HDM) (Rejeb et al., 2020). This device had the objective of being a window to a virtual world. Unfortunately, the technology was impractical for mass use. The innovation in this sector has kept growing and growing over the years but was rarely commercialized due once again to the difficulty of developing a device for mass use. In 1994, Paul Milgram and Fumio Kishino introduced the term mixed reality in their paper "A Taxonomy of Mixed Reality Visual Displays" (cited in Microsoft, 2022). This paper explored the concept of virtuality continuum and the taxonomy of visual display. With the emergence of Industry 4.0, the potential of AR and mixed technologies is rising again as it can now address several challenges in existing business models (Rejeb et al., 2020). The technology that is now becoming more mature has the ability to support a company's business processes. Advances in computer vision, graphical processing, display technologies, input system, and cloud computing have opened up new opportunities (Microsoft, 2022). In 2013, Google announced the Google Glass project, which later turned out to be a mediocre success. In 2015, Microsoft introduced the Hololens device, which is still marketed today and used in industrial applications.

2.2.3 Hololens 2 description

The Hololens 2 is an untethered holographic computer headset developed by Microsoft. It is the second version of the product, the Hololens 1 being the first version. It provides a more comfortable and immersive experience than the first-generation device. The device contains high-level components such as a visor with sensor and display features. The sensors enable eye and head tracking, cameras, depth measurement, and inertial measurement units such as accelerometer, gyroscope, and magnetometer. The display is able to render holograms. The Hololens 2 also has the ability to emit audio sounds and has a speech recognition feature. In addition, it has the capability of hand and eye tracking and voice recognition. It also has a spatial mapping feature and can make videos or photos with mixed holograms that appear in the physical environment.

Holograms are virtual objects made of light and sound that appear like real objects in the real world around the user. Holograms from the Hololens device can respond to gaze, gesture, voice command, or even interact with real-world surfaces (Microsoft, 2022). The Hololens can render holograms in the holographic frame directly in front of the user's eyes. Several appearances and behaviors are possible for holograms. They can either be realistic and solid, or cartoonish and ethereal. They can also be used to highlight elements in the environment or as user interface elements for the app. Holograms can be buttons, interactable objects, web pages, videos, texts, dynamic visuals, etc.

2.2.4 Augmented/mixed reality use in industrial applications

This section describes augmented and mixed reality potential in the industry sector, specifically in supply chain management and the manufacturing industry. An interesting study case with the Airbus company is then given.

Rejeb et al. (2020) provided a state-of-the-art review of the potential of augmented reality in supply chain management. The study's findings revealed that AR technologies could add value to five different parts of the supply chain, namely manufacturing, warehousing, sales & outdoor logistics, planning & design, and finally, human resource management. They highlight numerous practical applications where AR might be recommended in supply chain management, and claim that this technology can be a potential solution for enhancing business processes, improving operational efficiency, and increasing overall competitiveness.

The potential does not entirely lie in the production processes but also in terms of market size. According to BIS Research (cited in Rejeb et al., 2020), the augmented reality for businesses market was worth US 3.5 billion dollars in 2018 and is expected to reach more than US 200 billion dollars by 2025. This corresponds to a compound annual growth of around 65% between 2018 and 2025. the report also dedicates part of its content to the sale of smart glasses, which are a particular type of AR wearable device. This market has an estimated value of US 52.9 million dollars, with an approximate 25% of sales in the logistic industry. Furthermore, the technology analyst Gartner expected in 2018 that the sales of the head-mounted display would increase by roughly 19% through 2022 (cited in Rejeb et al., 2020).

Again according to Rejeb et al. (2020), AR refers to all activities wherein the main goal is to overlay virtual objects on top of the real-world environment with the objective of enhancing human senses and abilities. AR technologies can thus provide support to businesses throughout the entire product lifecycle. The first stage of this lifecycle is often the design process, and AR techniques can be applied to enable faster and more effective design processes (Elia et al., 2016). Indeed, research and development teams can manipulate, test, and evaluate prototypes using advanced simulation-based AR visualization. With this advanced visualization and manipulating tool, companies are able to reduce both time and cost in the engineering design task as the manufacturing of these prototypes sometimes becomes unnecessary. As AR and mixed reality combine virtual and real-world environments, designers can create and visualize, but most importantly, contextualize new creations.

In the production sector, AR can be practically used on assembly lines where the operators can visualize each part's location and display logistics and manufacturing information in front of them in their field of sight. This leads to least failures and better quality control (Rüßmann et al., 2015, cited in Rejeb et al., 2020).

An interesting study case of using mixed reality in the industry relates to the Airbus company. Airbus is a company designing, manufacturing, and servicing commercial aircrafts, helicopters, military aircrafts, satellites, and launch vehicles. In the coming years, their challenges are enormous. They delivered around 10 000 aircrafts in the last 40 years and expect to deliver twice that amount in the next 20 years (Microsoft, 2019). This challenge requires the company to adopt cutting-edge innovation. To face this challenge, they intend to use mixed reality intensely, which is why they have started partnering with Microsoft. Airbus believes mixed reality can help them increase quality, safety, and security. Indeed, the level of human error is significantly reduced when having the correct information in front of the eyes, and, in the aerospace sector, an increased quality is an increased safety. An essential factor of the success of such technology implementation is that, once implemented, the workers have the right data with the right rendering accompanied by a quality of visualization. The first place where Hololens 2 is used is in the design process, more specifically, mainly in the validation phase. In this phase, designers need to know whether their design is ready for industrialization or not, and Airbus thinks that mixed reality can accelerate the process by around 80 percent. On the manufacturing site, Hololens 2 is used to enable the workers to achieve complex tasks or tasks that require an intense reading of the documentation. Using Hololens also enables to work hands-free and access complex areas while still being able to read the documentation or having to ability to interact with the application. They therefore can handle heavy parts while having all the information they need right in front of their eyes. In total, Airbus has identified more than 300 use cases where mixed reality could add value to the process. To conclude this case, Airbus strongly believes that mixed reality can bring improvement up to 30 percent of some of their industrial tasks.

2.2.5 AR for training

Another field in which AR might be highly relevant is the training sector. Indeed, all AR, VR, and mixed reality techologies can bring improvements in this sector as the right information is displayed in front of the user's eyes, and interactive experience can help the person who is learning. This section details two studies conducted in the field of training with AR devices on industrial maintenance and assembly tasks.

Gavish et al. (2015) carried out a study that aims at evaluating the benefits of virtual and augmented reality for training in industrial maintenance and assembly tasks (IMA tasks). To empirically assess these technologies' efficiency and effectiveness for training compared to traditional training methods, they split a forty expert technicians group into four different groups. The goal was first to be trained on a particular electronic actuator assembly task and then execute this task. The difference between the groups lies in training, the first group having VR training, the second watching a filmed demonstration, the third having AR training, and finally, the last group was trained with the real actuator. Results show that both VR and AR training require more time than the other groups, but AR training enables fewer errors. VR training did not show a significant improvement compared to the control-VR group (the second group), but this is due to what the author calls a ceiling effect. Indeed, two training trials in the selected task were done, and the participants were all expert technicians. The study results show that the use of AR technologies for training, IMA tasks in the case of the study, should be encouraged as it seems to provide good results. VR training should also be encouraged and, more importantly, evaluated further.

More recently, Kolla et al. (2021) conducted a similar study. The goal was to provide empirical evidence comparing two groups, one having paper-based instruction and the other AR training with two AR systems, a Hololens device, and a mobile device. The study's metrics were the task completion time, the number of errors, and the workload index. The task that was asked of participants was to assemble a planetary gearbox. A qualitative interview was then conducted with the participants in order to get more insights about their overall experience. The authors are explicitly studying a possible improvement of training with AR because the workers in the industry are usually provided instructions on paper concerning assembly tasks. As the demand for product variants increases, providing these instructions on paper becomes increasingly challenging. Compared to VR for training, the advantage of AR is that it overlays digital information in the real world without compromising the awareness of reality, and there is a potential to provide interactive worker assistance. The study results show that AR instructions are superior to paper-based instructions concerning the number of errors and usability. However, the authors did not find a statistically significant improvement between the two groups concerning the completion time and overall workload. However, they stress that a smoother experience and enhanced user perception of reality using AR instructions systems will significantly improve the lead times as people get used to these systems.

3 Proposed solution

This section describes concretely the solution brought by this work in order to develop the solution mentioned in the objectives. As a reminder, the objective is to create a proof of concept of a combination of a 3D container loading algorithm with a mixed reality program using the Hololens 2 device. We believe that such software would enable faster and better loading efficiency. Therefore, the objective is to demonstrate with the development of the software that it is possible to combine these two fields of study.

First, a 3D container loading algorithm is developed (section 3.1). A MySQL database is designed to structure and store the data obtained by the algorithm (section 3.2). An API (Application Programming Interface) is then created in order to communicate the solution obtained by the 3D packing algorithm to the software present in the Hololens 2 device (section 0). With this communication enabled, the Hololens 2 is ready to receive information regarding the location of the boxes and other relevant data, and to indicate the user's actions (section 3.4). To visualize the solution obtained by the algorithm on a configuration, the Hololens 2 software described in section 3.4 might not be the best approach to simply and quickly visualize the solution provided. A software is thus developed in Unity to visualize the solution and be able to rotate the structure to fully visualize the solution of the packing algorithm (section 0). A box generator is developed using a guillotine cutting algorithm (section 0), enabling a complete generation of virtual datasets of boxes. The goal of this generator is to generate datasets of boxes to test and improve the 3D container loading algorithm. Finally, a web interface is later programmed in order to manage the different configurations that a user might have encoded in order to use and practice with the tools developped from section 3.1 to 3.6 (section 3.7).

The scenario we wish to address is a logistics operator having to stack a series of boxes of highly heterogeneous sizes on a pallet. The boxes are located next to the pallet, on the floor, or on a conveyor belt. This scenario can typically be found in many logistics warehouses around the world. Airports can also be locations where this kind of scenario occurs.

3.1 3D container loading algorithm

First, a 3D container loading algorithm is developed. As explained in the literature review, many algorithms exist to solve this problem. This work does not aim to develop the most optimal 3D container loading algorithm but to focus on combining this optimization problem with a mixed reality program. Therefore, a greedy 3D packing algorithm is a good approach, as the purpose is to generate a feasible solution. Several constraints are nevertheless well taken into account, and an improvement heuristic is developed in order to provide better results.

3.1.1 Pallet loading problem

As explained in the scenario, the operator needs to pack boxes either in a container or on a pallet. The specific problem addressed in this work might consequently be the pallet loading problem which can be seen as a specific type of problem among the 3D container loading problems. This problem comprises certain specificities. First, the box packing must be executed from bottom to top. It is indeed impossible to rely upon the lateral walls as it is possible when filling a container. Boxes are therefore only stable with the container floor or the boxes below. Thus, it is desired that the final structure of the boxes is as low as possible, which allows for better compactness and stability. Once the pallet is entirely packed, it is possible to ensure stability with the help of shrinking foil, which enables the whole structure to not fall apart. As mentioned in the literature review, this technique is used practically in

many situations and improves both vertical and horizontal stability. Although a pallet does not have a maximum height and lateral borders, virtual borders are considered. Therefore, we might use the term « container » later, which implies the pallet with all its virtual borders. The pallet has a square or rectangular shape, so the container consequently has also a shape of a rectangular parallelepiped.

3.1.2 Assumptions

As with most of the literature review, we consider several assumptions. The first assumption is that the boxes are rectangular in shape. None of the boxes have any other particular shape, such as a cylinder or other. The second assumption is the one explained just above, i.e., that the container is also rectangular in shape. Moreover, since the boxes and the container are of the same shape (rectangular), it is deduced that the placement of the boxes is either parallel or perpendicular to the edges of the container. These assumptions already simplify relatively much the problem we want to address. Finally, we assume that boxes are strongly heterogeneous, i.e., most, boxes if not all, have different sizes.

3.1.3 Programming language

The Python programming language has been chosen for this section. This object-oriented language enables fast prototyping due to its simplicity concerning the syntax and its wide range of libraries and documentation. It was indeed one of the most popular programming languages in 2021, rating second with 11.87% in May 2021, according to Statistics & Data (2021). However, one drawback of this programming language is its slowness. Being an interpreted language, it is not as fast as compiled languages like C or Java in code execution.

3.1.4 Problem definition

First, we need to define the problem based on the assumptions above. The following statement fully characterizes the objective of this section:

Given a set of n rectangular-shaped boxes, each characterized by the parameters width w_j, height h_j, and depth d_j ($j \in J = \{1, \dots, n\}$), and a container of size width W, height H, and depth D, the objective is to fill the container with the boxes in order to minimize the empty space once the boxes are packed into the container.

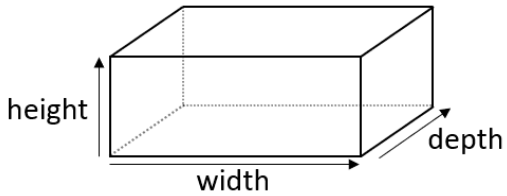


Figure 12 dimensions of a box or container

In the literature, the depth is sometimes called the length, such as in Paquay (2017). However, this term seems quite vague, as it might also describe an edge's length. Furthermore, we find the terms height, width, and depth in several papers, for example, in Martello et al. (2000), a scientific reference article concerning the 3D bin packing problem. Therefore, we decide to use these three terms in this work.

3.1.5 Axis definition

The axis chosen for this work in the 3D environment might seem contrary to what is usually done in the literature or industry. Indeed, in this work, the Y axis corresponds to the height, while the X and Z axes correspond to the bases, i.e., the width and depth. This decision is aligned with the Unity axis system, Unity being the software used to create the Hololens application. Furthermore, the 3D bin packing algorithm program has also adopted this axis system to keep uniformity. Finally, to be consistent throughout all this work, this paper makes the same assumptions.

3.1.6 Placement heuristic

As a reminder, the placement heuristic, or construction heuristic, defines the mechanism to decide where to pack the boxes to generate either the first or the overall solution. As explained in Zhao et al. (2016), placement strategies provide, in most cases, a fast and suitable solution with reasonable quality.

In the literature review, we have explained in detail the wall- and building-approaches, also called more generally the layered approach. However, this method must be favored when boxes are weakly heterogeneous, and, as explained in the assumptions, this work considers boxes as strongly heterogeneous. A better approach, in this case, is to place boxes one at a time individually. Such a technique can consist in identifying placement points. Placement points are 3D coordinates in the space that can match the position of a box. A box position (coordinate) is defined by its bottom left corner coordinates, as seen in Figure 13.

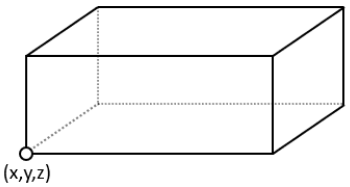


Figure 13 box coordinates position

The placement points technique is used in Martello et al. (2000), who call it corner points, and Crainic et al. (2008), who call it extreme points. The term *extreme point* will be used in this work (or, less frequently, pivot point). Extreme points are coordinates for hypothetical new box locations. These points correspond to geometric locations where future boxes will potentially be placed if several constraints are satisfied. Extreme points can be retrieved by getting the bottom right corner of the box, the top left corner, and the back bottom left corner, as seen in Figure 14.

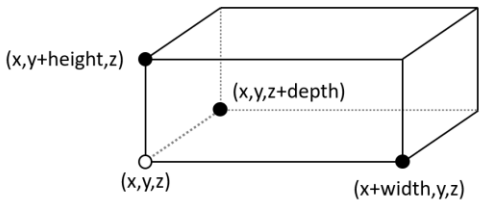


Figure 14: new extreme points (black dots)

At the start of the program, when no box is placed yet, a first extreme point is defined at the bottom left corner of the container. It corresponds to the (0,0,0) coordinates in the space. This extreme point corresponds to the location where the first box will be placed.

In order to define which box to take and position in the container, an ordering must be executed. This feature has been reviewed in the literature review, and the standard techniques discussed are *pre-determined ordering* and *dynamic ordering*. The conclusion made by Zhao et al. (2016) was that there is no definitive answer as to which decision technique is the best between the two. For this work, we chose the pre-determined ordering technique. As a reminder, this pre-determined ordering implies sorting boxes only once before the first box is even placed. However, we add an interesting feature in this work, i.e., the capacity for the human operator to indicate when a box is damaged. The process of handling this feature more precisely is explained in detail later, but it is essential to mention it here as it impacts the ordering. If a box is considered damaged, it is removed from the list. However, this does not imply that the sorting becomes dynamic, as no swap is performed, but it can be seen as an improved version of the pre-determined ordering. The most favored ordering in the literature is sorting the boxes by volume (i.e. Eley, 2002). Therefore, we decide that this sorting method is the default, but several other sorting algorithms are implemented as well, enabling a comparison between these techniques. Results are discussed in section 3.1.9. The other sorting algorithms are random sorting and the biggest surface sorting. This last pre-determined ordering sort boxes based on their biggest surface, i.e., therefore focusing on only the largest side.

3.1.7 General concept of the packing algorithm

As explained just above, the first step of the algorithm is first to sort all the boxes according to one characteristic, such as the volume of each box. The sorting algorithm used in this project is the Timsort algorithm used in the Python built-in function *sort()*. Tim Peters invented the Timsort algorithm in 2002 for use in the Python programming language. The algorithm is a combination of the merge sort and insertion sort algorithms and is designed to perform well on real-world data. The main idea of the Timsort algorithm is that it finds a subset of data that is already ordered and uses these subsets to sort the data more efficiently. Its time complexity is O (n log(n)).

Once the boxes are sorted, a second step consists in orienting the boxes in a specific direction. Each box is indeed oriented in the same way as the container. The longest side of the container will therefore orient the longest side of each box in the same direction. The same process applies to the second and third longest sides. Each box now has a determined rotation, but this is not fixed, as the box might be rotated during the placement process, further explained.

The iteration process can now begin. As mentioned previously, the first and only extreme point available is the origin (0,0,0). The first box of the list is taken and matched at the location of the extreme point currently tested. A series of tests are executed with several functions to determine if the box placement corresponds to a feasible solution or not. If it does, the box is placed, and the extreme point used is removed. Three new extreme points are generated at the three locations explained in section 3.1.6 in Figure 14. In the opposite case, the box does not fit at the location of the extreme point, i.e., it does not pass all tests performed. A rotation of the box is then conducted, and all tests are performed again. If the box does not fit with any of the rotations, then the next box in the list is tested, and the process continues until it finds a suitable box that can fit at the location of the extreme point. The orientation constraint is developed in more detail in section 3.1.8.1.2. The iteration process keeps going until all box are placed or no extreme points are available anymore.

A point worth mentioning is that the extreme point tested might be floating in the air, with no base below. This situation might occur in the case when a box is placed on top of another box with a slightly

longer edge than the box below. As a result, the extreme points added once the box is placed might be standing in the air. This problem might cause some trouble in the way that new boxes added at this point might as well be standing in the air and will therefore not pass the tests conducted. The points suspended in the air are thus condemned to not match any box. A solution has been found to solve this issue. When this issue occurs, a vertical projection is made, and the point is lowered until it reaches either a box or the floor of the container.

The pseudo-code of the general concept of the packing algorithm can be viewed below.

Algorithm 1: General concept of the packing algorithm

```
1  Function: packing_algorithm
       Data: boxes ⟵ the list of boxes to pack
             container ⟵ the container to fill
             sorting_algorithm ⟵ the algorithm to sort boxes
             extreme_points_manager ⟵ manage extreme points
       Result: Pack the boxes into the container. If not enough space is
                available, some boxes might be not find a location
2  begin
3      sort_boxes(boxes, sorting_algorithm) ;      // Timsort algorithm
4      for box ∈ boxes do
5          set_orientation(box, container)
6      end for
7      while still unplaced boxes AND still extreme points:
8          position ⟵extreme_points_manager.get_next_extreme_point()
9          if position.is_standing_in_the_air() then
10             position.lower_until_contact()
11         end if
12         for box ∈ boxes do
13             for rotation ∈ six_possible_rotations do
14                 if box_fit(box, rotation, position) then
15                     box.move(position)
16                     boxes.remove(box)
17                     extreme_points_manager.add(box.angle_1)
18                     extreme_points_manager.add(box.angle_2)
19                     extreme_points_manager.add(box.angle_3)
20                     break                    // get out of nested loop
21                 end if
22             end for
23         end for
24         extreme_points_manager.delete(position)
25     end while
26 end
```

3.1.8 Considered constraints

Despite the objective not providing the optimal algorithm to solve the 3D container loading problem, we nevertheless consider several constraints to come as close as possible to the scenario established above.

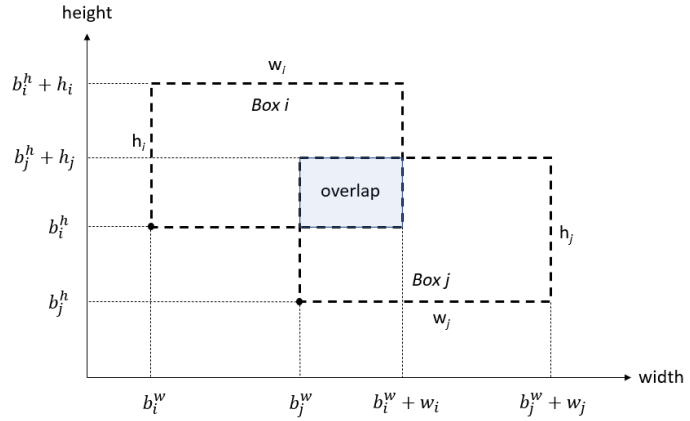
3.1.8.1 Geometric constraints

The primary constraints that are considered are the geometric constraints. As a reminder, these constraints involve the geometric assignment problem, in which small three-dimensional boxes need to be assigned and packed into a large container. As a result, overlapping boxes cannot occur, and boxes need to remain entirely inside the container. This last statement implies that boxes can not cross the limit of the pallet if a pallet is used as the support.

3.1.8.1.1 Overlapping constraints

Every time a box is placed, a check is executed to test if this newly added box does not overlap any other already placed box.

the two-dimensional schema below (Figure 15) has been created to visualize the overlapping constraint.



D

Figure 15 Overlap between two boxes (visualization on two axis)

The above scheme only considers two axes. The code of the project evidently considers the problem on the three axes. To solve this algorithmic problem, three functions are implemented. The first one is based on the scheme above, i.e., it considers the problem only on two dimensions. The second function applies the first one three times, once for each axis combination. Therefore, the algorithm considers the possible overlap between two separate boxes by verifying the overlapping with the axis width-height, height-depth, and width-depth. A third function enables to check the overlapping constraint between the newly placed box with all already placed boxes.

If the overlapping check returns a negative response, then the combination of the box, its rotation, and the extreme point determines that it is not a valid place to pack the box at this location with this specific rotation. If the value false is not returned, then the next check is executed, which is explained in the following constraint. The complete algorithm pseudo-code for the box overlap constraint can be viewed in Appendix 1.

3.1.8.1.2 Orientation constraints

The orientation constraints in the literature involve first the orthogonal position of boxes inside the container. This aspect has already been discussed in the assumptions. These constraints also involve possible rotations of boxes. A rectangular-shaped box has six possible rotations with an orthogonal position among the container (Figure 16). If we refer to the literature by Bortfeldt & Wäscher (2013), case 5 is considered in this work, meaning that boxes can freely rotate in each direction. No constraints are applied to individual boxes. For example, we do not impose on some boxes to remain at a fixed rotation.

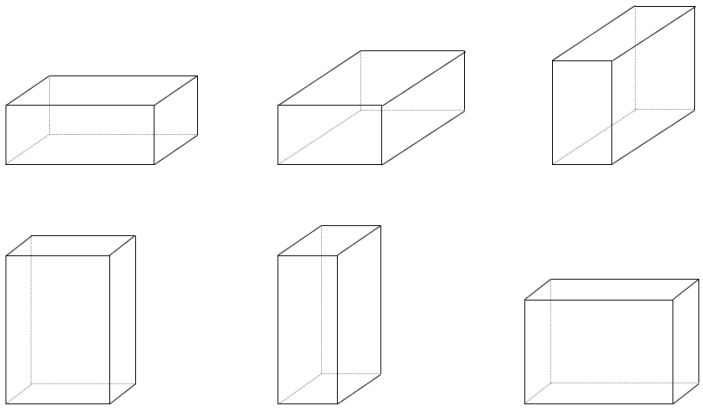


Figure 16 : 6 possible rotations of a rectangular-shaped box

3.1.8.1.3 Container limits

The verification of the container limits is also part of the geometric constraints. It is probably the easiest one to implement. It ensures that new boxes that are placed do not cross the borders of the container. This verification is executed on all sides of the container, including the top face. A bin is indeed characterized by a maximum height, no matter whether it is a pallet or an actual container.

3.1.8.2 Stability constraint

The stability constraint is probably one of the most difficult to implement in an algorithm. However, it is considered in this work because of its importance regarding the scenario. it is indeed unthinkable to provide a solution where boxes are, for example, suspended in the air. Moreover, given the practicality of this work with an implementation of a manipulation tool with mixed reality (the section with the Hololens 2), an algorithm with no consideration for stability would provide an infeasible solution for the operator packing the boxes.

As explained in the state-of-the-art review, two types of stability can be considered in the container loading problem, namely vertical and horizontal stability. This work only considers vertical stability, as most papers, because horizontal stability is much harder to implement. We, therefore, need to ensure that the items do not fall apart in the situation when the container is not moving, i.e., only when the load is to withstand the gravity force (Junqueira et al., 2012).

The solution implemented in this work is inspired by several papers discussed in the literature review on vertical stability. The solution brought by Mack et al. (2004) is a bit too restrictive as it enforces a box to be stable only if the center of gravity is supported by either the floor or a box below. For this work, we wanted from the beginning a stability algorithm that enables a box center to be suspended in the air while still being able to be considered stable if it, for example, lay down on multiple boxes on the sides. Therefore, we do not take this criterion into account in this work, but we can take inspiration from the other criterion, which imposes that a certain percentage of support must be observed for the box to be considered stable. We also make, as in most papers, the assumption that the center of gravity of a box corresponds to its geometric center. In addition, we assume that all boxes have the same density, which is the same assumption made by De Castro Silva et al. (2003).

Let us now discuss how the stability algorithm is implemented. We have at our disposal the function used to check if an extreme point is standing in the air. This function can be used again for the stability

check. As a reminder, the function makes a vertical projection from the point and keeps lowering until it reaches another box or the floor of the container. The stability check performed to check if a box is stable or not computes for each new box nine points on their base, as shown in Figure 17. We apply the function among these nine points to check if support exists at these precise locations. If six or more of these points are supported, we consider the box to be stable. Six is not a number that was chosen arbitrarily. Using this way, we enforce the box to have at least approximately 67% of base support, which is close to the base support recommanded by Hemminki et al.(1998, p2227) that was 70%. The word *approximatively* is quite important here because we do not compute the actual percentage of base support using the boxes below. We only retrieve data from the nine points used and can compute approximate base support.

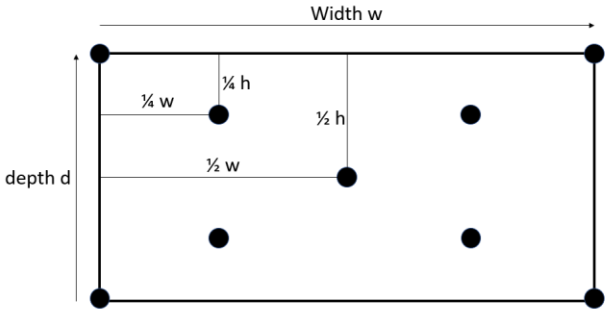


Figure 17 Stability base points

This algorithm also enables the box to be stable even if the center point (geometric center) does not lay down on a support. Asking more than 50 percent of base support was also a requirement. Indeed, if only 50 percent was required, the risk is that the boxes placed one on top of the other progressively form a staircase shape, each one being placed only 50 percent on the one below. The third box could cause the structure to collapse. Imposing a rate of 67 percent does not entirely diminish the risk of this type of problem, but it does reduce it considerably. Tests using the Unity 3D software were made. Each box was applied a Rigidbody component, giving it consequently a mass and gravity characteristic. Although these tests do not formally demonstrate the stability of the structure, the results obtained were good, and each box remained in its position.

Imposing a 67% minimum contact rate with boxes or floor below also makes the friction between the box and its supportive base relatively high. This is relevant in the case of horizontal stability. Despite the fact that this work does not aim to handle the horizontal stability, avoiding complete unstable horizontal load was our intention. When conducting experimentations of our algorithm, we realized that when forcing a 100% base support tend to construct tower structures with all boxes. As mentioned in the state-of-the-art review, Terno et al. (2000) emphasized that algorithms that construct 'tower structures' tend to be unstable for horizontal stability. Therefore, forcing only approximately 67% of base enables a form of interlocking principle and avoids tower structures.

3.1.8.3 Container weight limit

The weight limit is the last constraint included in this work. Each box is characterized by a weight and the container by a maximum weight limit. When the maximum weight limit is reached, new boxes can no longer be added to the solution. This mechanism might seem like it does not include the weight as an optimization problem, but it is actually of great relevance when using the improvement heuristic described below.

3.1.9 First results

The first results of the packing algorithm indicate that the placement heuristic developed provides reasonable solutions. At first sight, a majority of boxes are placed when the number of boxes seems reasonable compared to the size of the container. To test the algorithm's efficiency, a function first needs to be developed to provide a result, which could be considered a score. This score corresponds to the filled space inside the container by the boxes. The function computes the volume of the container, then computes the total volume of each box placed inside the container. It does not take into account the boxes that were finally not chosen for the solution. We indeed face the Single Knapsack problem, which includes one container with multiple boxes. It is consequently expected that some boxes will not suit the solution.

The following table provides the score, i.e., the filling rate of the container, where the total volume of boxes corresponds to twice the volume of the container. Therefore, it is impossible to put all boxes inside the container, but as we face the Single Knapsack problem, the role of the algorithm is to select the most appropriate boxes. Three different pre-determined ordering have been implemented. The scores among the *random* column do not bring much information as the experience can be repeated with any other random ordering, providing completely different results. However, the column *volume* and *biggest surface* provide meaningful results, as these would provide the exact same score if the experience is executed again with the same boxes. These results also demonstrate that volume ordering provides a significant gain compared to the two other ordering types.

Scores of the container loading algorithm with different dataset of boxes and pre-determined ordering

boxes	*Volume*	*Biggest Surface*	*Random*
10	94.0%	74.96%	67.0%
15	82.42%	66.88%	88.91%
20	94.0%	72.68%	74.98%
25	88.8%	74.5%	79.34%
30	88.75%	77.52%	66.77%
35	92.09%	71.86%	77.3%
40	87.99%	75.12%	72.05%
45	92.77%	84.64%	75.3%
50	90.28%	75.6%	68.54%
55	88.27%	72.92%	60.76%

3.1.10 Improvement heuristic

As explained in the literature review, placement strategies such as the one developed in this project provide fast and suitable solutions with reasonable quality (Zhao et al., 2016). We decided to broaden the search space using an improvement heuristic, providing a reasonable gain in terms of the quality of the solution provided by the packing algorithm. Using neighborhood moves, this new algorithm can find improving moves to converge towards a local optimum progressively. The packing algorithm described above has not changed at all. The new algorithm uses the packing algorithm multiple times and applies changes only to the pre-determined ordering. This ordering sequence is the only variable that helps us improve the solution. Our improvement heuristic is therefore not as complex as genetic algorithms or tabu search methods as explained in the literature review, but it nevertheless provides correct results. The objective of this improvement heuristic is to enhance the solutions that are provided by our placement heuristic.

To determine whether a solution is better than the other, the function providing the score described in the first results section is used. As a reminder, this function calculates the total space filled by boxes inside the container. The improvement heuristic uses multiple pre-determined ordering, applies the packing algorithm on each of them, and retrieves a score for every run. Improvement heuristics first need a placement strategy to be executed, and then are able to provide gain. The packing algorithm is thus first run using a pre-determined ordering such as the volume, biggest surface, or simply a random sort. The improvement heuristic then swaps two boxes in the list and applies the complete packing algorithm again. The score of this new result is compared to the first one, and if improvement occurs, then the new ordering is considered the best and default one. This new ordering will consequently be the basis for new swaps if only it is the best among all the other tests. The process keeps iterating over and over until a criterion is met, such as a maximum iteration number. Another criterion might be to stop the iteration process if no improvement has occurred in the last hundred iterations (or any other number).

In order to test the improvement heuristic, boxes were generated by the box generator, which will be explained in detail in section 0. To quickly explain, 70 boxes were generated in such a way that they represent two times the total volume of the container. We have, in this case, more boxes than it is possible to store in the container, but as we face the Single Knapsack Problem, this situation might happen. The goal is not to pack all boxes but to pack the most appropriate boxes in order to fill as much as possible in the only container at our disposal. The first run using the placement heuristic already provides a good result with 82.96 percent of total space filled. As shown in Figure 18, the best score keeps improving throughout the iterations and reaches the score of 89.86 percent at iteration n°59. We, therefore, have approximately a 7 percent improvement in around 280 seconds, corresponding to 4 minutes and 40 seconds. However, the algorithm's performance deeply depends on the processor used and the total number of boxes. This test has been executed using an Intel Core i5 9thGen. Some authors run their programs for hours or days in the scientific literature to get optimized results. For industrial applications, it actually depends on the time when the data of boxes is obtained and when these need to be packed. This time might greatly differ from one industry to another. Advices for future work and improvement will be given at the end of this work, including techniques to improve the performances of the algorithm.

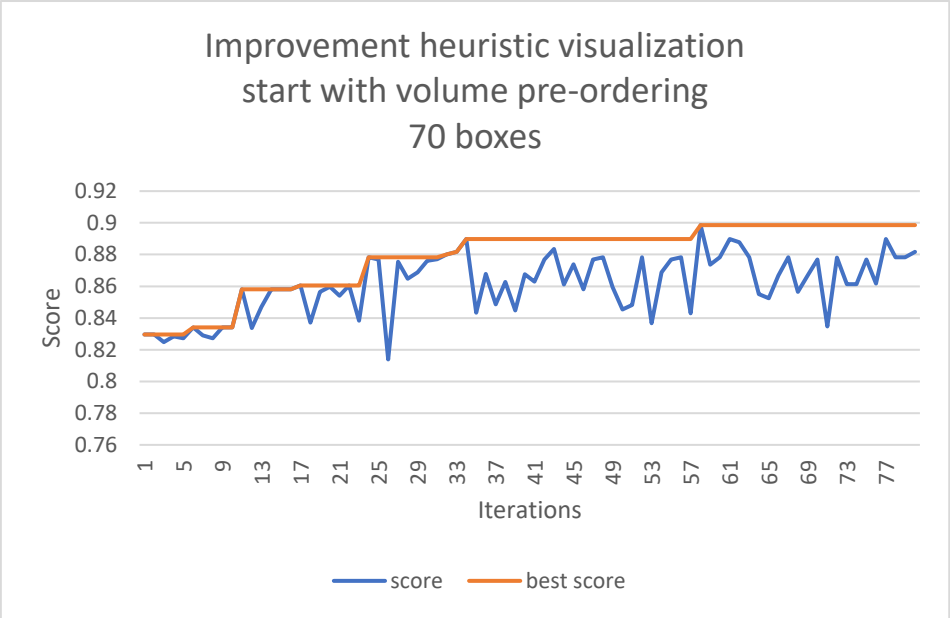


Figure 18: Improvement heuristic visualization (start with a volume pre-determined ordering)

Another test has been executed using a random pre-determined ordering for the first iteration. Again, the results seem quite similar to the ones obtained above, as the improvement is around 7 percent for the first 80 iterations. The starting point is however much lower than with a volume strategy, starting at only 74.58 percent.

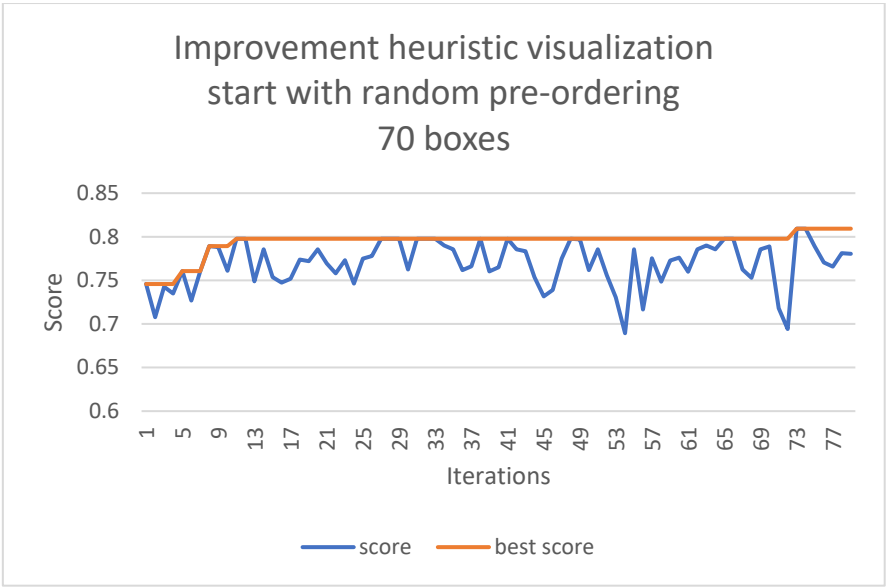


Figure 19: improvement heuristic visualization (start with a random pre-determined ordering)

3.1.11 Good programming practices

In this project, particular care was taken to apply good programming practices, whether it be optimization techniques, object-oriented principles, or good file management. This section describes several good practices that have been used in the project. Each individual part might be completely independent from the others.

3.1.11.1 Virtual environment

In the Python programming language, a virtual environment is a Python environment that includes the Python interpreter with a specific version of Python, the libraries, as well as the scripts installed. All these components are isolated from those installed in other virtual environments or the default libraries installed on the system. A different version of Python might indeed already be installed in the computer system or server which is used as the operating system. More precisely, the virtual environment is a directory tree that contains Python executable files. It also includes files indicating to the system that it is a virtual environment.

The advantage of using a virtual environment is that it isolates all scripts and languages used for the project. We can therefore transfer the whole project, including the virtual environment, to someone else, and this person would be able to run the program by activating the virtual environment. Therefore, this person does not need to install either the programming language or the libraries used in the project. Another advantage is that the isolation prevents any update of the programming language or libraries from affecting the code running. The project was indeed developed using particular versions of the language and libraries, and the developers of these components might release new versions. Being isolated enables the code to use the same version as it was developed.

3.1.11.2 Object-oriented programming

Object-oriented programming (OOP) is a programming paradigm with which the code is under the form of objects and classes. It is an excellent way to structure a software program into reusable blueprints. This programming paradigm has been used throughout this project as a way to organize and structure the code.

3.1.11.3 Extreme points sorting

Extreme points are organized in an object called *Extreme Points Manager*. This object provides the following extreme point when the packing algorithm runs and receives the three new ones each time a box is placed. The way the *Extreme Points Manager* works is optimal as it provides the next extreme point in O(1) time complexity. The insertion is however O(n) and sorts each new extreme based on their Y-X-Z location, Y being the first priority, followed by X, and then Z.

3.1.11.4 General good practices

The Python programming language enables good programming practices, such as using comprehension lists. The variable typing is also not compulsory in Python but recommanded as a good practice. Such practical approaches have been applied throughout the project.

3.2 The database

To store and retrieve the data of the packing algorithm, a first version of a database has been developed. This database consisted of a JSON file (JavaScript Object Notation). The advantage of this system is that JSON objects are very similar to Python objects. A series of more than 20 functions have been implemented to manipulate this database, i.e., to store, update, retrieve and delete information. The JSON file took the following form:


```
{
    "parameters": {
      "sortingAlgorithm": "volume"
    },
    "currentBox": {},
    "undeterminedBoxes": [],
    "placedBoxes": [
      { "name": "box1", "width": 27, "depth": 7, "height": 20, "start_x": -49, "start_y": -66,
        "start_z": 186, "end_x": 0, "end_y": 0, "end_z": 37, "id": 19, "rotation_type": 2, "weight": 1
      },
      {
        "name": "box2", "width": 30, "depth": 30, "height": 7, "start_x": -82, "start_y": -66,
        "start_z": 168, "end_x": 0,  "end_y": 7, "end_z": 0, "id": 16, "rotation_type": 0, "weight": 1
      }
    ],
    "unplacedBoxes": [],
    "noFitBoxes": [],
    "pallet": {
      "width": 50, "depth": 50, "height": 26, "x": 0, "y": 0, "z": 0, "id": 1, "max_weight": 200
    },
    "pivotPoints": [
      { "x": 0, "y": 21, "z": 0 },
      { "x": 0, "y": 21, "z": 30 },
      { "x": 38, "y": 24, "z": 27 },
      { "x": 30, "y": 26, "z": 0 }
    ]
  }
```


Figure 20: Version 1 of the database using JSON file

After discussion with the supervisor of the thesis, a proper relational database has been developed. The relational database has many advantages in terms of speed, simplicity, security, accessibility, accuracy, etc. The relational database management system (RDBMS) used is MySQL. The database has been developed using the tool phpMyAdmin, a free software tool written in PHP that is intended to handle the administration of MySQL databases using a web interface. This tool supports most of the operations available with MySQL databases and also provides the possibility to execute any SQL statements. With no server at our disposal, we use the tool WampServer, which automatically installs all the necessary components to run a web application, and is very intuitive to use. It is a Windows web development environment that allows the creation of web applications with Apache2, PHP, and MySQL databases. The latter is the reason for the use of this software. WampServer also allows tuning a server without touching any of the setting files.

The packing problem faced in this work is the *Three-dimensional Single Orthogonal Knapsack Problem*. The database structure's design must be conducted to provide and store relevant data for this specific problem. We consequently have a single container that can be connected to multiple boxes. In order to be able to make data analysis to improve the packing algorithm, an idea that emerged was to store multiple container-boxes assignments, called *configurations*. Such a design would also enable the user to store several container-boxes groups in the case where he personally has sets of boxes he wants to pack on a pallet or in a container.

A table *Config* is thus created, with several attributes. We find in this table the attribute *id* that determines the primary key, making this attribute unique in the whole database and enabling the software to differentiate one configuration over the others. An id might be quite difficult to remember and differentiate for a user, a name has thus been given to a configuration. Characteristics of the container or pallet are also necessary and can be attributes of the table Config since we only have a single container in the problem. We thus add both the coordinates of the container as well as the dimensions. Referring to the pre-determined ordering that happens once at the beginning of the packing sequence, the sorting algorithm also needs to be registered in this table. This sorting algorithm can be, for example, volume, biggest surface, or random. Finally, three more attributes still need to be defined. The first one determines which configuration the user is currently using. This one, as we will see later in the API section, helps to identify which configuration is to be transmitted to the Hololens 2 software. Another attribute determine the maximum allowed weight a container or pallet can handle. Finally, the last attribute indicates the current box, which is the box that the user with the Hololens is currently taking and packing. This final attribute is a foreign key, referencing the primary key of the next table described below.

The second table created in this database is the *Box* table. It is quite evident that this main component in the project is included in the database. An *id* also defines a box for the same reason as the id in the configuration. A box is also linked to a configuration instance through a foreign key. This connection using the configuration's id makes the container-boxes assignment possible. It is possible to give a name to a box, although this characteristic is less relevant here than for a configuration. As an example, this attribute is helpful in the case when practising with board games. Dimensions of the box also need to be stored. Concerning the box's location, we can store two different positions, namely the start and the end. Start refers to the position of the box when it is not loaded onto the pallet, while end refers to the box's location when it is packed. A box is also defined by a rotation type, an integer representing one of the six possible rotations a box box can have. To be able to handle the container weight limit constraint, a weight needs to be assigned to a box. Finally, a status is given to each box. This important attribute can be of 5 different options: undetermined, unfit, unplaced, placed, or damaged. *Undetermined* means the packing algorithm has not run yet, and there is no indication of where the box should be placed. *Unplaced* means the algorithm has found a feasible location for the box in the container, but the human logistic operator has not yet physically packed the box. Therefore, the box is only virtually placed. Conversely, the *placed* status indicates that the box is both virtually and physically placed in the container. *Unfit* means that the algorithm has not found any available location for the box, which is thus rejected from the solution. Finally, the *damaged* status means the box can not be included in the solution. This final status introduces the next paragraph dedicated to the damaged box functionality.

A functionality that we want to add to this project is the ability for the user to indicate that a box is damaged when he uses the Hololens 2 software. The database structure needs to be able to handle such cases. When a user indicates to the software that the box he is handling at the moment is damaged, it means this box can not be put in the container and therefore needs to be removed from the solution. As we cannot leave an empty hole in the middle of the structure, a recalculation of the packing algorithm is necessary. However, we do not want the user to take back all the boxes he has already placed. Therefore, it is relevant to store extreme points, which enables retrieval of possible future locations of boxes that will follow.

The following figure illustrates the database structure. This was taken from the phpMyAdmin conceptor, which gives a representation of the Entity-Association schema.

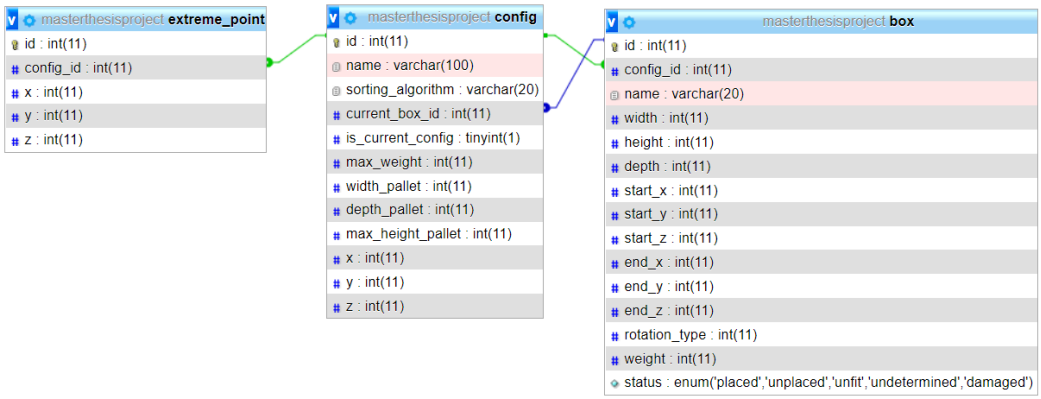


Figure 21 : Entity-Association schema (from phpMyAdmin conceptor)

3.2.1 Object-Relational-Mapper

The use of an Object-Relational-Mapper (ORM) is conducted in this work. An ORM enables to write queries to the database using an object-oriented programming language. It enables first to query the database using the language we desire, e.g., the programming language used in the project. As we master this language better than the SQL language, we are able to leverage that fluency. Secondly, and most importantly, the use of an ORM enables to directly retrieve Python objects from the database, onto which we can then apply functions of this specific object. For this project, SQLAlchemy has been used as the ORM.

3.2.2 Other possible solution

The database could have used other database types such as SQLite or PostgreSQL instead of a MySQL database. The advantage of SQLite is its simplicity, as the installation is relatively fast and easy. The drawback is that this type of database is not entirely fit for a production environment. In order to make this project the most professional as possible, this choice has not been kept. Concerning PostgreSQL, it is a much heavier database type than MySQL and offers much more features. However, the need for such features was irrelevant for the project, and a MySQL database has thus been chosen.

3.2.3 Problem encountered

An interesting problem encountered during the implementation of the project is worth mentioning. A first version of the implementation of the packing algorithm implied that the information was transmitted to the database every time a box was moved inside the main loop. Therefore, the algorithm relied entirely on the data stored in the database as they were constantly up to date. However, this process made the algorithm relatively low, much lower than when a JSON file was used. The idea to resolve this issue was to retrieve all necessary data from the database at the beginning of the packing sequence and then work locally only with Python data. Once the algorithm has finished packing, it transmits all data back to the database. This improvement has made it possible to reduce considerably the execution time required to run the packing algorithm. For example, it needed 54.9 seconds to run the algorithm for 25 boxes with the first implementation, while it only now requires 1.9 seconds for the same data with this improvement. Thanks to this upgrade, the improvement heuristic solution has been implemented and was able to provide good results. The time to run the improvement heuristic with the first version would have been exceptionally long in terms of time processing.

3.3 The API

In order to combine the bin packing algorithm developed and the Hololens 2 software, communication must happen between these two entities. As explained before, the programming language used to develop the bin packing algorithm is Python. The code must be executed on a server, and communication must be carried out with the Hololens 2 headset to transmit all the information Hololens needs to display the holograms correctly. More details about the Hololens 2 software will be provided in section 3.4. This communication must be done via a network; building an API is a solution. The word API refers to an application programming interface, which is a mechanism that allows two software components to communicate with each other using a set of definitions and protocols (Amazon Web Service, 2022). The word application refers more specifically to any software with a distinct function. The interface, on its part, can be viewed as the contract between the two applications. This contract defines precisely how communication between the two entities uses requests and responses. An API architecture usually consists of a client and a server. The client is the application sending the requests, while the server is in charge of sending the response.

3.3.1 Requirements of the project

Concerning the project, we have two specific applications: the container loading algorithm with the database, i.e., the server, and the Hololens 2 software, i.e., the client. The Hololens 2 device must first transmit requests to the server to indicate that a user will pack the boxes onto the pallet. Therefore, it asks the server to reset the configuration that might have previously been computed. Another possible request might be to place the pallet hologram at a specific location in the physical world. Finally, other requests might be to indicate that a box is damaged or that a box is placed at the proper location. Each request will be explained in more detail later on. The vital aspect to emphasize at this point is that the server does not need to transmit data to the client unless the client asks for it. This point will help us to choose the proper API type.

3.3.2 Different types of APIs

Four types of API exist, each being different by the time they were created and their use (Amazon Web Service, 2022).

- *SOAP APIs* use Simple Object Access Protocol. The exchange language between the two entities is XML, and this API is less flexible and was used more often in the past.
- *RPC APIs* are another form of API that stands for Remote Procedure Calls. It is one of the simplest and earliest forms of API, in which a code block is executed on another server. Once the output is computed, it is sent back to the client.
- *Websocket APIs* is a modern web API that uses JSON objects to pass data. This type of API supports a both-side connection between client apps and the server. The advantage of this system is that the server can send callback messages to connected clients. This form of API is thus more efficient than REST API.
- *REST APIs* are the most flexible and popular forms of API. The client sends requests to the server as data. The server then uses this data as input and runs internal functions to return the response in the form of data to the client. Rest API stands for Representational State Transfer, and this API type defines several functions, such as GET, PUT, DELETE, etc. which client apps can use to access server data. The data exchange between the client and the server is done using HTTP.

During the first semester of the year, the course "Digital Business Capstone Project" asked us to develop a robot car using a Raspberry Pi communicating with a program running on a server. A Websocket API was used in this project because the server sometimes needed to send requests to the car to perform some action. However, it is not the case for this present work. Indeed, given the description of the different types of API mentioned above, the most appropriate form of API for this project is a REST API. As mentioned in the project requirements, the client is the one that initiates the communication with the server, and it asks for a response once the request is sent.

The main feature of a REST API is the statelessness property. It means that the server does not keep client data between requests. Client requests are similar to URLs that can be entered into a browser. However, the server's response is very different, as it contains plain data. More specifically, this project will use a web API. Modern web APIs are REST APIs and consist of an application processing interface between a web server and a web browser. REST API is a particular type of web API that uses the standard architectural style explained above in the REST API section.

3.3.3 Programming language and framework

Now that the API type has been defined and chosen, we need to look at the possibilities of building a Web API. Because the packing algorithm has been implemented using the Python programming language, it makes sense to use this language again to program the web API. Another language could, however, have been chosen as well.

Programming an API with Python can be done in several ways. Three different frameworks that can fulfill the requirements have been identified: Flask, Django, and FastAPI. *Django* is a relatively heavy framework, including from the start many files and has its own structure that has to be learned to master before getting any result. This framework was not chosen since Django might be too heavy for a project like this. Django can indeed be used to handle large-scale applications with thousands of users. We personally hope for this project that some people, such as Ph.D. students, will use par of what has been implemented and improve it or use it the way they desire. Django requires practice if the person has no experience with this framework, which is why it has not been chosen. FastAPI is a recent framework, with the particularity of being more efficient than the others. It was not chosen to realize the API because its documentation, as well as its community, is still relatively weak. Finally, Flask is a framework, known for its simplicity, allowing to realize APIs in only a few lines of code. Its community and documentation are also consequent, which makes it possible to obtain help and to find solutions easily on forums or websites. As shown in Figure 22, the community sees interest in both Django and Flask. According to Google, numbers on this chart can be interpreted as follows: “Numbers represent search interest relative to the highest point on the chart for the given region and time (Worldwide in this case). A value of 100 is the peak popularity for the term. A value of 50 means that the term is half as popular. A score of 0 means there was not enough data for this term”

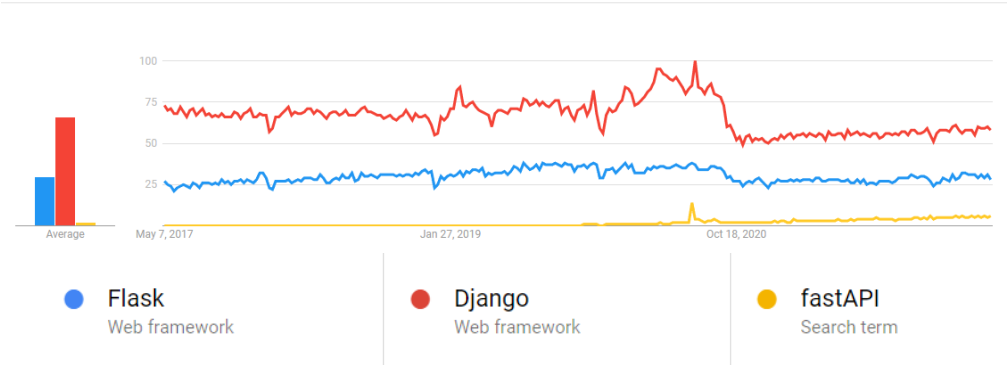


Figure 22 Interest over time. Source: Google Trends (07/06/2022)

3.3.4 Implementing the solution

Now that the programming language and framework have been chosen, the implementation of the solution can start. As mentioned in the above section, the API needs to be run from a web server to be able to respond to the client. Flask allows running the server on a local address, which is only applicable if both the client and the server run on the same computer. This functionality is helpful for local tests, but this solution is not feasible as the Hololens software is located on another computer. Fortunately, Flask enables us to simply change the host IP address to a private IP address, which starts with 192.168, which can thus be accessed from any device currently connected on the same network. Finally, the last solution is to host the program on a server that can handle requests from any network.

In the project, the code on the Hololens 2 headset comes from the Unity software, which uses the C# language. The Unity code is first programmed on a computer in the C# language and must then be compiled in C++ to be used on the Hololens device. The communication from the Hololens software for the request to the server will consist of a web request, such as one that can be typed in the URL section of a browser. The C# language enables to make web requests quite simply. The response part must be adequately defined to be understood from the Hololens software. As the JSON object is one of the most frequent forms of transmission between APIs, it is coherent to use this method for this project. The server, written in Python, must be able to transmit the response in JSON, while the client, written in C#, must correctly interpret the JSON objects received in the response. Fortunately, both languages have libraries that can transform JSON objects into their object style, both Python and C# being object-oriented programming languages.

The first request a user might send to the server is resetting the current configuration. As a reminder, the current configuration is the one currently used by the user. There can only be one current configuration among all at a time. Resetting a configuration implies setting the coordinates of each box assigned in this configuration to a null value and indicating that the user has no more box in his hand and currently placing it. The status of each box is also modified to indicate it is no longer virtually placed according to the packing algorithm. The server sends code 200 as the response, which is the standard response of an API when everything goes fine in the execution of the code.

A second possible request is to indicate where the pallet or container is placed in the physical world. It simply consists of 3D coordinates, the origin being the location where the Hololens 2 software was started from. Knowing the position of the pallet enables the packing algorithm to place the boxes on top of the pallet. The response from the server can once again be the number 200 if everything works correctly. Otherwise, an exception might be raised and sent.

A third request asks the server to run the packing algorithm with the data provided by the database. It defines the first extreme point to the pallet location and then runs the algorithm as explained in the previous sections. The number of iterations can be added in the parameters of the requests to enable the improvement heuristic if more than one iterations is asked. The server's response indicates that the algorithm's execution has been completed.

A fourth request asks for information from the server about the following box to take. The server, in this case, sends actual data to the client, not simply the code 200. The information transmitted includes the location of the box before it is packed, the location of the box on the pallet, as well as the dimensions of the box. If the box has a name, it is also transmitted. When the data of a box is asked from the client to the server, this implies that it is this box that the user will take into his hand and then place it at the location indicated in the data transmitted. The order of the box transmitted is not necessarily the same as the packing algorithm packed them onto the pallet. The code here looks for the lowest box; if several boxes are at the same level, it chooses one randomly. If no boxes are to be packed anymore, the message clearly indicates this information to the user.

A fifth request from the client to the server indicates to the server that the box that was previously asked has been positioned onto the pallet. This way, the human operator indicates he has done his job and is ready to receive the data of the following box to pack. Without making this request, the "next box" request (the fourth request) keeps sending the same information.

Finally, the sixth and last possible request that the Hololens software can send to the server is the box damaged request. The request indicates that the box that the user is currently packing is damaged and cannot be added onto the pallet. This results in a recalculation of the packing algorithm, and the new next box to pack is sent.

A sequence diagram of the happy path has been created and is shown in Figure 23. The happy path implies that all functions work correctly, and that no exception is raised. The user uses the tool as he is supposed to, and no box is damaged. The API described throughout this section is used to enable the interactions between the Hololens entity and the server entity, meaning the central part of the diagram. The left part, i.e., the interactions between the user and the Hololens software, has not yet been developed and will be in section 3.4.

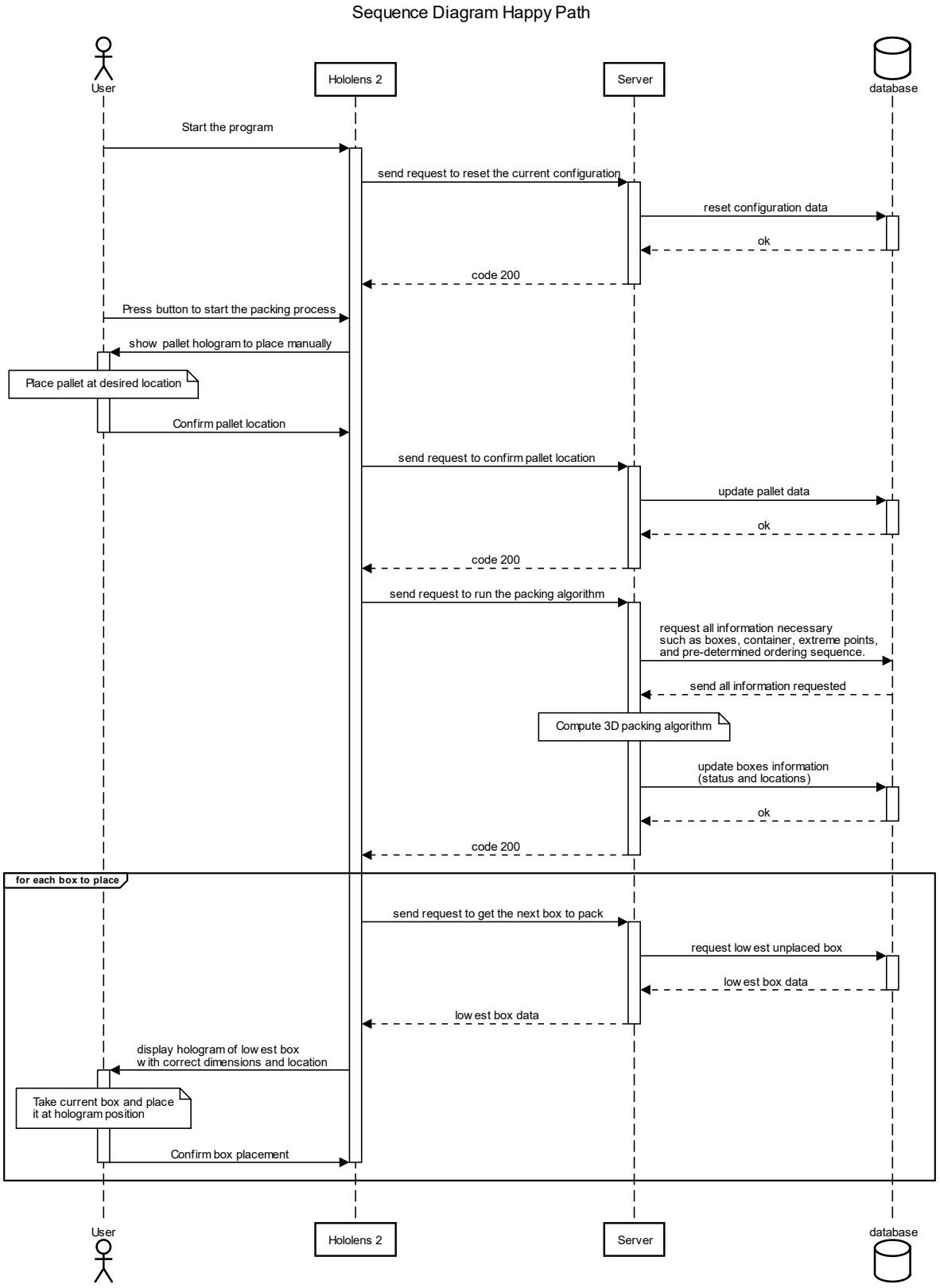


Figure 23 : Happy path sequence diagram

3.4 Hololens software

3.4.1 Objective

The main objective of this master thesis is to develop a proof of concept that combines a 3D bin packing algorithm with a mixed reality application. This section focuses on the mixed reality solution using the Hololens 2 device. Using this device, the goal is to develop an application enabling the user to load a container or pallet with the help of mixed reality. Furthermore, such technology will provide the user information such as the box to take in hand and the precise location of where to pack it on the pallet.

3.4.2 Assumptions

Numerous assumptions are made concerning the problem. The first assumption is that the user is located in a room where a container or pallet is empty, and several boxes are located beside it in an unordered manner. These boxes might, for example, be located on a conveyer belt.

The second assumption done in this project is that the start location and the dimensions of the boxes are already known to the program. It is therefore not the role of the user or the software to determine such information. For industrial applications, this type of assumption might be applicable thanks to third-party devices, such as sensors or cameras screening the environment, looking for boxes, and determining both their location and dimensions. For example, a device determining the dimensions of boxes has been developed by other students at HEC Liege for a class project using a lidar. As for the camera detecting the location of boxes in the room or on the conveyer belt, an interesting article is "Learning practically feasible policies for online 3D bin packing" provided by Zhao et al. (2021). In this article, they use a recognition tool to determine the precise location of boxes on a conveyer belt. A robotic arm then uses these locations to take the boxes and place them on the pallet (Figure 24). A similar idea is thus conducted in our work, assuming that the location of boxes in the physical world is transmitted by network to our software. Furthermore, working with distinct devices, each having its own role, defines a more coherent project for industrial application.

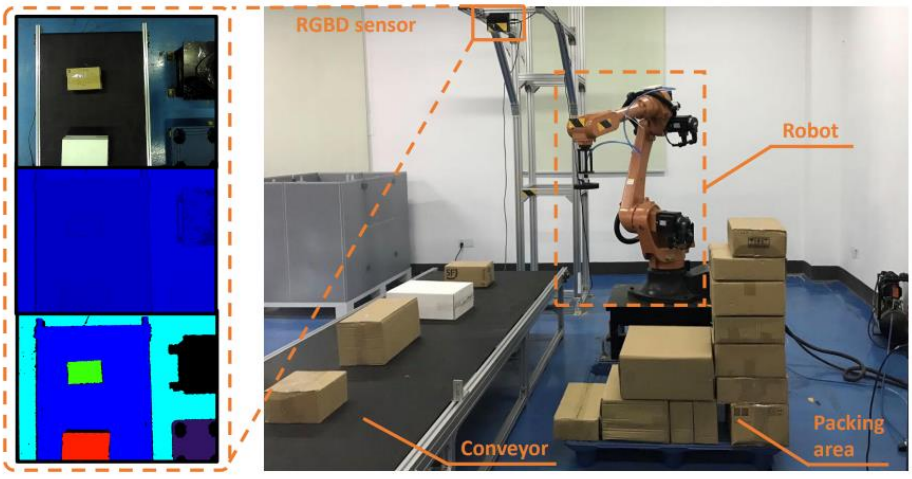


Figure 24: box dimensions and location recognition tool (from Zhao et al. (2021))

Finally, the last assumption we make for this work is that the user correctly places the boxes at their location. The user is indeed asked to confirm that he has correctly placed the box at the indicated location; therefore, no verification is done afterward.

3.4.3 Implementation of the software

To implement the solution, the Unity software must be used. Another possible software might have been the Unreal software. Both have similar ways of working and performances, but because the Unity software is used in the Digital Business Laboratory, this one has been chosen. The software uses the programming language C#, but if the Hololens software is the device for which the code is developed, it will be converted into the programming language C as it is this one that is used on Hololens. Some extensions must first be installed on Unity to develop the project for Hololens. A non-required but still handy extension is the MRTK2. It is a well-known extension that stands for Mixed Reality Toolkit and is a Microsoft-driven project that provides components and features for mixed reality apps developed on Unity. It provides powerful components such as hand-tracking systems, UI controls, a multi-scene manager, and much more.

Once the user starts the software using the Hololens 2 device, he is first invited to press the start button. This button takes the form of a hologram. When the user presses this button, the UI gives the perfect impression of pressing a physical button, with both sound and button-pressed movement. A message is sent with the API to the server by pressing the button. This communication is possible using a web request from Hololens. The message sent corresponds to the first request developed in the API section (section 3.3.4), which asks the server to reset the current configuration. It implies that the end locations of boxes and the pallet location are set to null. To better understand this explanation and the following ones, we advise looking at the same time the sequence diagram provided above in this work (Figure 23).

Now that the start button has been pressed, a hologram appears in front of the eyes of the user. This hologram corresponds to the pallet with the correct dimensions. The pallet takes the form of a thin rectangle having the sizes of the width and depth of the data stored in the database. The height is not visible because it would be challenging to visualize and place the whole container correctly. The user can move this pallet hologram with his hand by pinching it. He can also move it remotely by targeting it from a certain distance with his finger. Then, by pinching it, he can move it remotely. When the user is handling the pallet move, he can also alter the rotation of the pallet by rotating his hand. Performing such action can seem complicated, but once the user has a little bit of practice, it becomes quite intuitive to manage.

The user also has the ability to press another button, enabling the pallet hologram to stabilize on the horizontal axis. Once the user is satisfied with both the location and rotation, the user press the confirm button, which transmits the location to the server. This feature corresponds to the second request explained in the API section. The server handles this request and updates the location of the pallet. Then, the following request is directly sent through the API to the server, asking the server to run the packing algorithm. Again, this request does not expect any specific response except the code 200, indicating that everything worked properly. Finally, a third request is sent, the fourth in the API section. The three requests described here occur one after the other in a short time. The user does not have to do anything during this time. The last request asks the server to send the data of the first box to pack. The server response corresponds to the box's name, dimensions, start location if there is one, and end location on the pallet. Other irrelevant data might as well be sent, such as the weight and the status. The data received must be first understood by the Unity script. These data are in JSON format and must be converted into C# objects. Once converted, it is possible to apply functions to these objects and make them appear in the form of holograms.

Creating a hologram material in Unity can be done in only a few steps to generate this blue transparent aspect that usually comes to mind when addressing the subject of holograms. Then, we need to apply this material to every Unity object we desire to look like this. Figure 25 gives a representation of a hologram.

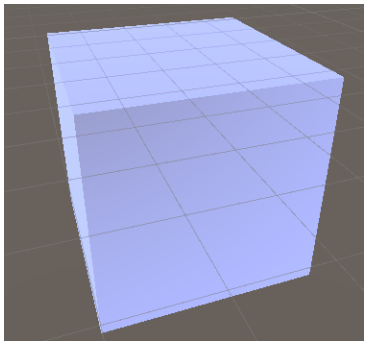


Figure 25 Picture of a box Hologram in Unity

A script uses the data retrieved from the last web request to create a box with the correct dimensions. The hologram material is applied to this box, and the rotations are deducted from the Unity software developed. The data concerning the rotation is indeed held locally in the Unity software and not on the server and the database. When the user looks at the pallet positioned a few moments before, a hologram is visible with the size of the box to place. When the user looks at all the boxes that have not been placed yet, an arrow is present on top of the box to take. Indeed, as mentioned in the assumption section, we assume to know the box's location before it must be placed on the pallet, which corresponds to what we call the start location. Using this location, an arrow is present on top of the box that the user must take into his hands. This arrow is slightly moving up and down to feel more immersed and clearly indicates to the user which box to take. This animation has been created using a sinus periodical function instead of an animation developed by unity.

Once the user has placed the box at the hologram's location, the user needs to push the button confirming the installation of the box at the correct location. This action sends the request to the API confirming the installation and sends a second request asking the next box to pack. Therefore, the user always needs to press only one button between each box. The process keeps going until all boxes are placed. The algorithm that sends the next box to place always takes boxes that are the lowest as possible. This mechanism enables to build the packing of boxes from bottom to top, which is necessary for stability constraints. The user has to move inside the logistic warehouse to take and pack boxes, and it can be convenient to make the menu with the buttons following the user. This assistance is possible by pressing a simple button that will enable the menu to follow the user's position.

During the process of loading the pallet, an indicator is visible on top of the pallet. This indicator takes the form of a circle with a progress bar. The background is transparent, while the progress bar is green. It indicates the progress of the filling of the palette. When looking at the circle's center, it is possible to visualize the number in percent. This functionality is helpful for the operator to estimate his remaining workload to finish filling the pallet. The data determining this percentage is determined by the server and transmitted to the Hololens when the request is made to confirm that the box has been placed.

3.4.4 Issues overcome

An issue during the program's development was handling the box rotation. Indeed, the server and the database do not store this information. To solve this issue, the solution imagined was to always put the pallet's location to the origin position (0,0,0) in the database and to manage locally (into the Hololens software) both the location of the pallet in the physical environment as well as the box rotation. To enable the boxes to be ideally rotated the same way as the pallet rotation, all boxes added have been assigned as a child object of the pallet in the Unity software. Therefore, the python

algorithm on the server does not manage any of these constraints and only computes the relative location of these boxes with regard to the pallet.

Another issue that occurred was that the geometric position of an object in Unity corresponds to its geometric center. This is in contrast to the position established by the packing algorithm that states that the position of any box corresponds to its bottom left corner, as shown in Figure 13. The solution found to resolve this issue was to create a parent empty object for each box and place this parent object at the correct coordinates.

Finally, a last interesting issue worth mentioning occurred when we wanted to set the length of the edges of a box. Indeed, when increasing or decreasing the length of a side of an object, Unity enlarges both sides of the object, making the object's geometric location erroneous. This issue was handled similarly to the previous issue described above.

To view a demonstration of the Hololens software, a video uploaded on Youtube can be seen with the following link: https://youtu.be/VeYUz13ihXw. In this video, all steps described in this section are shown. Another short video shows the final result, demonstrating that it is possible to pack several boxes with this tool. The link is the following : https://youtu.be/iV5wB1-y--E

Appendix 3 gives a user's manual with instructions to upload the software developed on Unity to the Hololens. There are multiple ways to do this, but some of them have been unsuccessful for us. Giving these instruction will possibly help a future person working on a similar project.

3.5 Unity visualization tool

The Hololens 2 software developed above perfectly fits the needs stated at the beginning of this work. However, it is sometimes necessary to visualize the solution proposed by the packing algorithm. This is important first as a verification tool to check if the boxes do not collide and check if the solution proposed by the algorithm looks good visually. The Hololens 2 software enables the visualization of the boxes through the form of holograms, but a more straightforward tool would be preferable in some cases. Such a tool will be even more relevant for the next section, the boxes generations.

Several authors in the literature have used multiple tools to visualize the packing algorithm. Therefore, there does not exist only a single solution. We want to visualize both the pallet and the boxes positioned on it. Other indicators such as current load rate might be relevant to visualize too.

As Unity is the software used for the Hololens software, it seems natural to use it again for this smaller project. The idea will be similar to the one applied in the Hololens software, as we desire to visualize quite the same data. The API will again be used as the communication tool between the server and this program.

An independent similar Unity project is thus created and uses part of the code developed for the Hololens project. However, this newly created project does not include any form of augmented or mixed reality concepts. Using keyboard buttons, calls to the servers through the API are made. When starting the software, the user presses the space bar, which sends the 'reset current configuration' request to the server. The software immediately sends two other requests, the 'set pallet position' request as well as the 'run packing algorithm' request. Then, the user can press the N button to ask for the next box to take. Receiving the box's data, the software generates a rectangular form into the scene and applies a random color to it, enabling the user to differentiate all boxes. Then, pressing the P button enables to confirm the box location, i.e., send the 'confirm box placed' request to the server. The user can press N and P iteratively until all boxes are visible. Finally, to rotate around the structure to visualize the whole solution, the user can go back to the scene view and manipulate the environment. We see that the solution brought by this visualization tool uses the same calls to the API that the Hololens app.

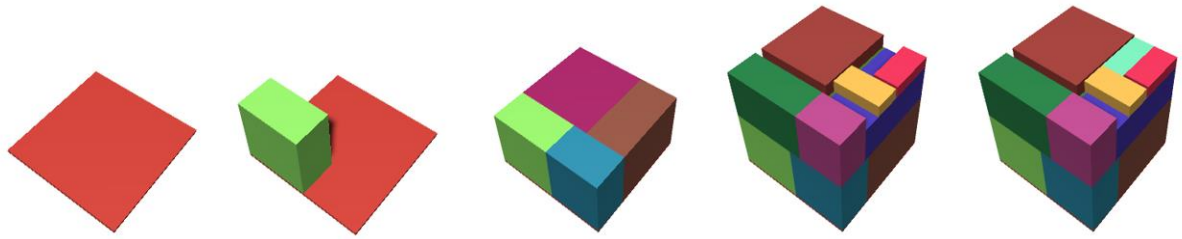


Figure 26 pallet progressively filled

3.6 Boxes generation

3.6.1 Objectives

In order to test our 3D bin packing algorithm that we developed, a good idea might be to obtain a dataset of items with their dimensions. The ideal dataset would even consist of an industrial database with authentic boxes packed on pallets or containers. With real-world problem data, metrics such as the total space filled with the items inside the container would become highly relevant, and we would obtain an accurate measure of the algorithm's performance. Unfortunately, such data are quite hard to find. To compensate for this issue, it is, in fact, possible to generate a series of item instances that can then be used in the algorithm.

Thus, a simple program can generate boxes of variable dimensions. A minimum and maximum length can be required in order to avoid getting boxes that are too small or too big. The main drawback of such an approach is that it is almost certain that there is no optimal solution where all these generated boxes fit perfectly in a container. However, the solution to such a problem can be found thanks to a well-known problem in operations research, the cutting problem. The cutting problem was explained in detail in the literature review, giving the distinction between the guillotine and non-guillotine approaches.

This tool aims to generate a whole dataset of boxes quickly and easily, each with different dimensions, as we face the Single Knapsack Problem in this work, which assumes boxes to be strongly heterogeneous in size. The problem faced is actually quite different from the one in the literature review. Indeed, the goal of cutting in this project is to generate a database of items in order to test our 3D packing algorithm. It is indeed not an optimization problem, and the dimensions of the items are not predetermined. However, the idea behind this new algorithm that is developed shares some key elements with the ones from the cutting and packing problem.

We would like to highlight again that this tool is not relevant for the Hololens solution, as the Hololens software aims to help pack real instance boxes that exist in the physical world. The box generator aims only at providing a tool to quickly generate virtual boxes that are used to improve the 3D packing algorithm, for example, in the improvement heuristic (section 3.1.10).

3.6.2 The proposed solution

The general idea to generate a perfect possible solution is to start from a large box having the size of the container. The whole space is thus occupied. A cut is then executed on the box, using a total guillotine cut. The box is thus cut entirely into two distinct parts. Each of the resulting boxes will be cut again, each one independently. Therefore, the proposed algorithm does not cut the whole structure every time but cuts only a section in two parts. As a result, each of the resulting parts of a cut is itself cut independently from its sister box. This approach still corresponds to what is called the guillotine cut approach but is, however, better than a total-guillotine cut approach which would cut everytimes the whole structure. A non-guillotine cut approach would nevertheless have been an even better solution but is harder to implement. As the objective in our case is to generate non-existing boxes, this selected approach is enough to solve our problem.

An algorithm is partly defined by the parameters it can take into account. A vital choice then arises. Two parameters are decisive, in our opinion. The first one directly concerns the number of boxes that will be generated. It is thus possible for the user to define this number precisely. The second main argument defines the minimum size that a box side should respect.

If among all boxes already obtained through cuts, we take the next one to cut randomly and again randomly choose a side of this box to cut, we might obtain very irregular shapes for the boxes at the end of all cuts. Indeed, some boxes would be really long on one side and extremely short on other sides. Moreover, some boxes might never be cut, while others might be cut several times, making them very small compared to others. Although we want strongly heterogeneous boxes, we would want the variance between the size of boxes to remain decent. Therefore, two distinct issues arise, namely the box volume difference that we want to keep reasonable, and the box sides that are sometimes too different between each other, making some sides of boxes too long or too short. To avoid the first issue, the choice of the next box to divide is crucial. The invented solution makes the algorithm first computes a ratio for each box based on its volume compared to the total volume of the container. It enables the algorithm to have a high probability of choosing a box with a more considerable volume and, conversely, a small probability of choosing a small box. This way of doing still lets a chance to choose small boxes, but the probability is relatively low. Now that the next box to divide has been chosen, we want to avoid the second issue. The same technique is used, i.e., use ratios with, this time, the length of the sides of the box. Therefore, there is a high probability of choosing the side with the highest length. If all sides have similar lengths, then the chances are similar. It is proportional to their ratio compared to the other sides. Finally, once the box and the side are chosen, the last decision consists of defining the exact location of the cut on the side. We call it the cut length. This cut location is obtained randomly between 1/6 and 5/6 of the length of the side. A uniform distribution is used here, giving the same probability to every integer (centimeter) located in the defined range. We now have the three elements required, i.e., the box, the side, and the location of the cut on this side. The cut is then executed, and two separate boxes are obtained.

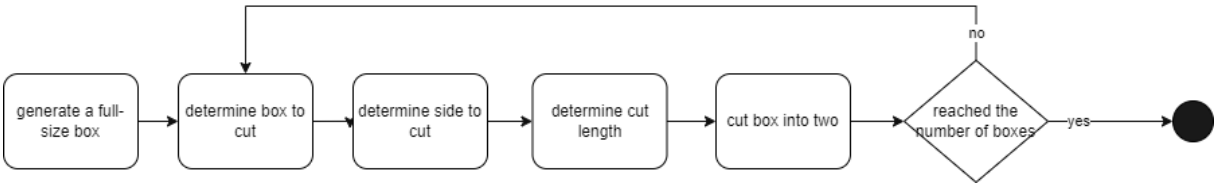


Figure 27: Box generator algorithm scheme

Using the Unity visualization tool, we are able to visualize the solution provided by the box generator tool. We can clearly see the perfect solution provided. Boxes seem to be of quite a reasonable size. For the examples provided in Figure 28, the container size is 50 centimeters for both the width and depth and 100 centimeters for the height. One hundred fifty boxes were generated with a recommended minimum of 10 centimeters for each side. The time for the software to generate all these boxes does not exceed 10 seconds.

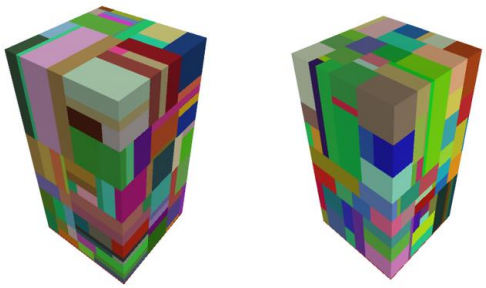


Figure 28: Two distinct examples of the box generator

3.6.3 Tests

As mentioned in the objectives, the goal is to obtain a dataset of boxes in order to test the packing algorithm. With this box generator, we are confident that there exists a solution where all boxes fit in the container. Of course, we do not expect our greedy algorithm to provide the perfect solution. Even in the literature, very few authors would be able to provide the exact solution. We would like to mention that the graphs obtained in the improvement heuristic (Figure 18, Figure 19) were obtained using two times the box generator tool, therefore providing twice the number of boxes that could provide a complete solution. Providing more boxes makes it possible to obtain a better score. We are now going to test the algorithm with only one call of the box generator tool, making it possible to visualize the full solution, as well as the solution provided by the packing algorithm only with the provided boxes. The score is expected to be lower than in the improvement heuristic section, which used the box generator tool twice. The score is indeed 84.27% in Figure 29, and we can see that the score is 94.05% in Figure 30, where two calls of the box generator tool have been made, providing more boxes than it is possible to fit. Figure 30 shows that generating two times smaller amount of boxes generates bigger boxes and also enables the packing algorithm to provide a better score because the algorithm can take boxes of both box generator instances.

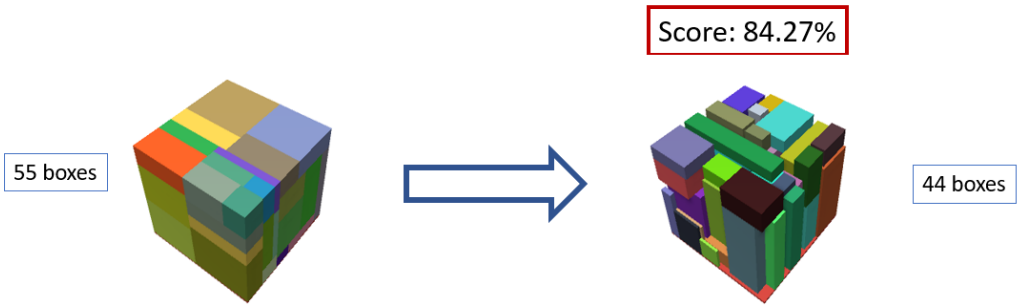


Figure 29: Applying the 3D packing algorithm with a single call of the boxes generator tool

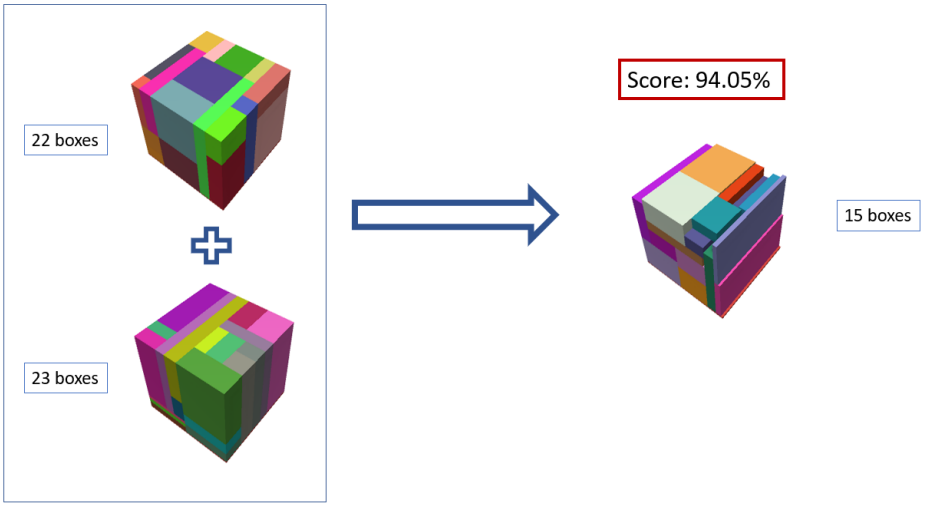


Figure 30 Applying 3D packing algorithm on double call instances of boxes generator tool

3.7 The web interface

One of the requirements for the project was to develop an interface in order to provide the ability for the user to manage the entire database, i.e., create and manage configurations, execute tests on specific configurations, and define which configuration is the current one. Nowadays, the most modern form of an interface for managing a computer program is a web interface. No installation or updates are required, and it is the simplest form for the user. Fortunately, Flask, the framework used to create the API, can also be used to create web interfaces. A formal structure needs to be implemented to manage the HTML files and routes to transfer the user from one page to the others properly. Once again, the key aspect of Flask is its relatively simple use, and this will enable any other person who decides to work on the same project to understand all that has been developed in this project quickly. Again, the IP address needs to be defined, and we can switch from a localhost address, private network IP address, or public IP address, which will differ in the access process.

When the user arrives on the home page, a list of the configurations currently in the database is displayed (Figure 30). For each configuration, the number of boxes assigned to this configuration is shown, as well as other metrics such as the container dimensions, the number of boxes, the sorting algorithm, and if the configuration is the current one used or not. For each of them, the user can click on three different buttons, the first indicating that the selected configuration becomes the current one, the second to edit it, and the last to delete it. When deleting it, a confirmation message is popped on the screen to prevent the user from unintentional click and consequently accidentally deleting a configuration. Finally, a button enables to create a new configuration.

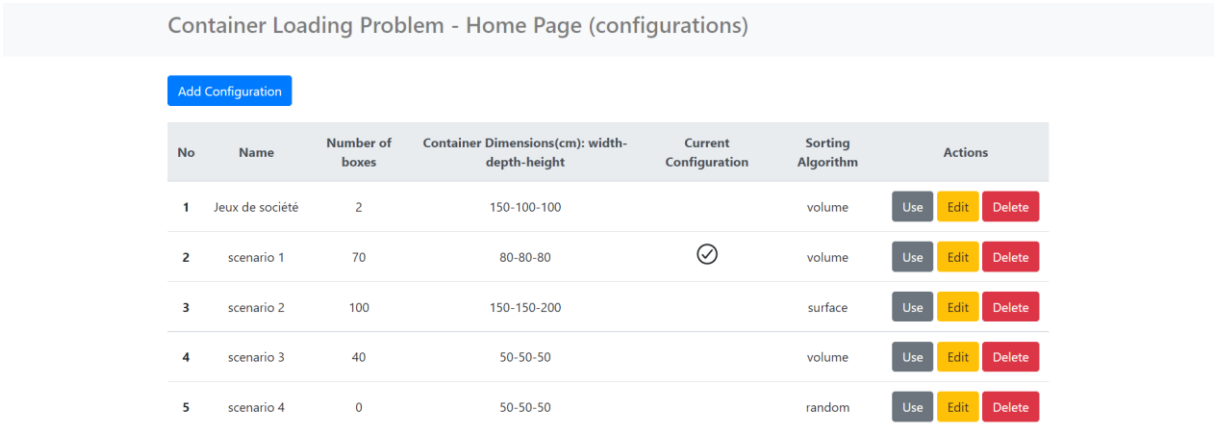


Figure 31: web interface: home page

Pushing the button to create a new configuration transfers the user to the creation page of a new configuration (Figure 31). On this page, a form is displayed, and the user is invited to fill it out. If some fields are not correctly filled and the user presses the following button, a warning indicates the user did not fill in the information. The form involves information such as the name of the configuration, the container dimensions, and the sorting algorithm used to define the pre-determined ordering of boxes. A check box is also pre-selected to indicate that the newly created configuration becomes the new current configuration. If the user had pressed the "edit" button on the home page, he would have arrived on a similar page, but the questionnaire fields would be already filled in.

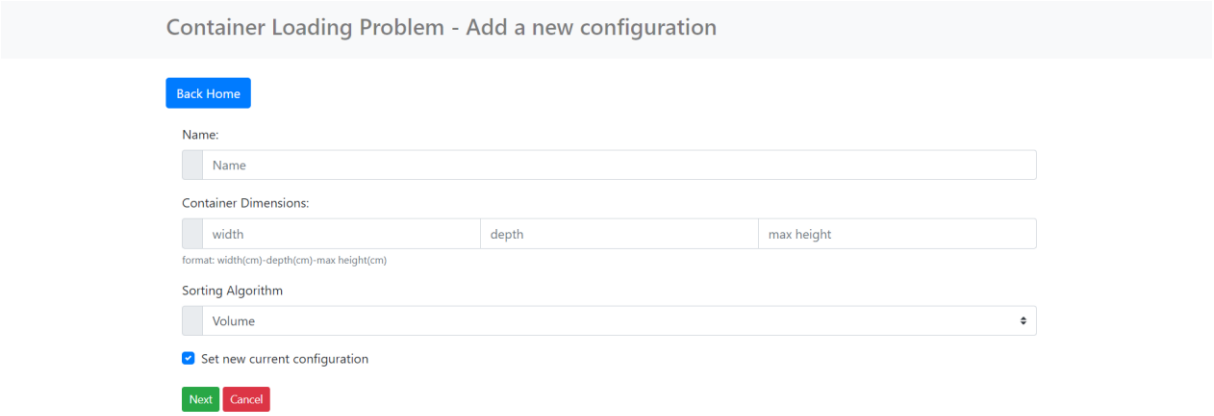


Figure 32: web interface: Add new configuration

Pressing the next button activates the code to create the new configuration in the database. No box is attached to this configuration, which is why the user is transferred to the second page of editing a configuration (Figure 32). This page is the most important one to manage and test a configuration. The first feature available on this page is manually encoding boxes that will be added to the configuration. Once all fields are filled in, pressing the insert button adds a new box to this configuration, and this box is visible in the first place on the list of boxes below.

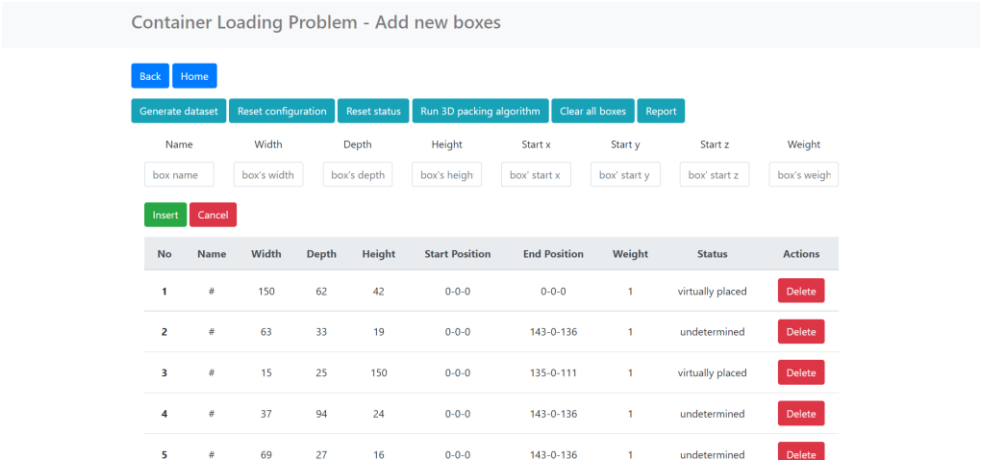


Figure 33: editing a configuration page 2

A list of buttons is also present at the top of the page. Each button corresponds to a particular feature.

- *Generate dataset:* Transfers the user to another page where he can set generator parameters such as the desired number of boxes or the minimal size of a box. Once these fields are filled in, the submit button reroutes the user to the previous page and generates a list of boxes using the box generator described in section 3.6.

- *Reset configuration*: Reset the status of all boxes of this configuration to 'undetermined' and set their final position to a null value. The configuration also no longer has a current box. The container position is also reset to the origin.
- *Reset status*: For all boxes having the status 'manually placed' ('placed' status in the database), their new status becomes 'virtually placed' ('unplaced' status in the database). These features enable the user to do multiple practices with a configuration. Indeed, whenever the user uses either the Hololens software (section 3.4) or the Unity visualization tool (section 3.5), the status of boxes placed is set to 'manually placed'. Therefore, this button enables to restart the exercise.
- *Run 3D packing algorithm*: *:* Transfers the user to another page where he can run the packing algorithm described in section 3.1. The user can encode the number of iterations, enabling the improvement heuristic to be applied if more than one iterations is encoded.
- *Clear all boxes*: Delete all boxes from the configuration
- *Report*: Transfer the user to another page where he can look at relevant information such as the configuration score, the number of boxes placed in the container, and the number of boxes that the packing algorithm has rejected.

To understand the action behind each of these button without looking at the documentation provided just above, the user can hover is cursor on top of these button, which will make appear a small text describing the actions if the user press the buttons.

A user flow chart has been created using the Overflow software. This user flow chart can be visualized by clicking on the following link: https://overflow.io/s/RX9S1TIY. Pressing key one will orient the user to the start page. Then the user is invited to play with the buttons to visualize the user flow chart. This chart represents the pages that a user can reach from each page. However, it does not show the actions executed by the code behind every button. The goal is to focus entirely on the flow of the user.

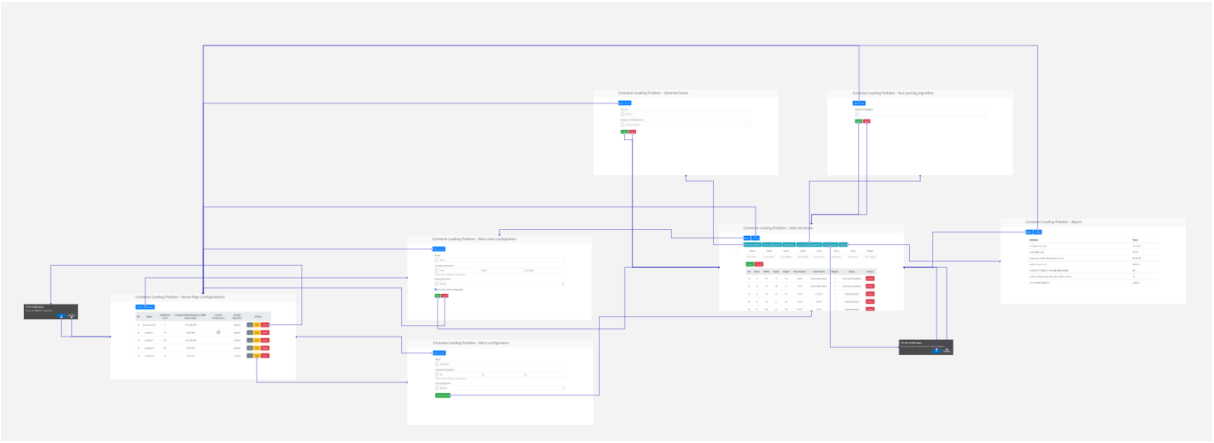


Figure 34: User flow diagram (using Overflow)

Appendix 2 gives a user's manual with instruction to run the program, start the server, and interact with the web interface.

4 Solution analysis

4.1 Utility of the solution

This section aims to provide a reflection on the product's utility. This critical analysis will emphasize the advantages and drawbacks of the solution.

4.1.1 Strengths of the solutions

From an operational point of view, the product is not ready yet to be used industrially. However, it is a proof of concept demonstrating the possible combination of augmented and mixed reality with a 3D packing algorithm. During our research to write the literature review, we did not find at any time a product combining these two fields of study. The closest work that we have been able to find is the one proposed by Peters & Schyns (2020), a short paper that discusses a possible future similar product. The product enables packaging a pallet or filling a container with boxes. Although it is not ready yet for industrial purposes, it still provides a good idea of a possible final solution. If the tool was to be improved, we believe it would enable better container filling than the more straightforward scenario where an operator fills a container by hand without instruction or instructions on paper. However, this statement is hypothetical since no qualitative or quantitative analysis has been performed to evaluate the product's effectiveness. However, we can rely on the literature review that has been conducted, which has pointed out that augmented reality can allow for greater efficiency, particularly a reduction in the number of errors.

We also believe this product can be adapted to sectors such as training or entertainment. For example, concerning the training of new employees, an adequately explained tutorial would allow training of the operators who have to fill the containers. This tutorial in augmented reality would allow a total interaction with the operator's movements. The tutorial would provide clear indications of where to take the boxes, which boxes to choose, where to place them in priority, what to do in case of a damaged box, what to do in case of a different problem, and what to do once all the boxes have been filled, etc.

4.1.2 Drawbacks of the solutions

The proposed solution to facilitate the filling of a container has, however, some negative points. This section aims to explain some of them.

The first essential point is the battery life of the Hololens. Indeed, this one is relatively weak, with an autonomy of about 2 to 3 hours at most. If the tool was used industrially, the operator should have at his disposal several Hololens in order to be able to work continuously.

A second important point is the comfort of use. The Hololens headset has some features that might not be very pleasant at some point during the use of the device. The first point is the weight of Hololens. This one has greatly improved during the transition from Hololens 1 to Hololens 2. However, this weight can be unpleasant, although we think this problem is not the most restrictive for users.

The next point is the field of view that the Hololens provides. Although the visor is quite large, the holograms can only appear on a small portion of the visor. Therefore, the field of view is relatively small, and it is sometimes difficult to observe a hologram entirely if it is close to the user. It becomes then necessary to move and turn the head constantly to perceive the environment as a whole. This

negative point is the main one raised by the people who tested the Hololens during the time it was at our disposal.

In the case where the operator performs a physical task, he will potentially sweat. The headset can then become particularly annoying and scratchy.

Finally, the last point concerning physical comfort is that of headaches. We experienced several times slight headaches during the tests of the product. Having let several acquaintances try the Hololens during the months when the Hololens was at our disposal, these users have also reported some headaches after use.

A possible drawback of using Hololens in industrial sectors occurs when working in a noisy environment. Indeed, it becomes impossible to use both oral instructions as well as receiving vocal instructions. Therefore, the system's design must handle such cases by providing an intuitive and pleasant user experience, even when working with complicated conditions.

We also believe that the efficiency of the solution, although potentially positive in the above-mentioned advantages, remains inferior to the use of a robot to pack the boxes. Figure 24 (from Zhao et al., 2021) showed an example of such a solution with an robotic arm. A comparison study is however necessary to make any comparison statement. The cost of a robot is however much higher than a Hololens.

Finally, a little ethical reflection is necessary in our case. Indeed, if the objective is to deploy this kind of software in an industrial context, the risk is to reduce the human being to a pair of arms obeying the algorithm. Add to that all the little concerns about comfort mentioned above, which would make the operators' work very unpleasant. For a beginner employee, this may seem fun and rewarding in terms of learning the job, but once the employee becomes experienced, we do not think this type of tool has a place, at least in the way it has been established in this work.

4.2 Futur works and improvements

One way to overcome many of the disadvantages mentioned above is to potentially use another device than the Hololens. The Hololens is however one of the best tools on the market today. Augmented, virtual, and mixed reality is a future market, and many big companies are investing massively in this field, like Meta with the metaverse. These technologies are therefore bound to evolve rapidly in a growing market. We believe that in the future, there will be devices that will overcome the disadvantages mentioned. For example, some companies are working on augmented reality glasses, which would indeed allow a lighter weight, a better field of vision, and greater comfort. Once these new tools are on the market, a possible future work would consist in reimplementing this kind of program and analyzing if the discussed negative points still persist or not.

More concretely, it is possible to improve the container loading algorithm in different ways. This algorithm is indeed far from being the most efficient or the fastest. Concerning the speed of execution, it is mainly a problem when many boxes are present. If, in addition to a high number of boxes, a large number of iterations is required for the improvement heuristic, the program can take a long time to determine the final solution. An improvement that could be quickly realized would be to segment the boxes according to their location on the pallet. By dividing all the boxes into segments, placing a box in a specific location would be much faster since the constraints would be quicker to perform. For example, to check if a box collides with its neighbors, it would be enough to compare the position of the new box with the boxes of the same segment, or at the very least, the adjacent segments. This would reduce the number of verifications made because, for the moment, the verification is done with

all the boxes of the problem considered. Segmenting the boxes could also be helpful to check the stability because, again, we go through all the boxes of the problem in the current algorithm.

A third possible improvement concerns the user experience (UX). This area may sometimes seem less important than program development, performance, or other areas, but it is still crucial for the program to be intuitive, enjoyable, and easy to use. The technology provided by Hololens is remarkable and provides new ways of thinking about UX. Some concepts, until now impossible to put into practice, are now easily implemented with Unity, especially the MRTK2 module. We are indeed able to create new combinations of real and virtual worlds. However, no matter how beautiful and immersive the environment created is, interactions are still essential to make the tool much more enjoyable and interesting. Much work has to be done at this level. An improvement of the Unity program on Hololens can therefore be conducted within the framework of this project. For example, voice commands or instructions received by audio messages could potentially be a good idea. Another element that could be interesting to add is what Microsoft calls the hand menu, which displays a personalized menu when the user opens his palm in front of him. The buttons are then clickable with the other hand. This mechanism is intuitive and could be applied when the user confirms the placement of a box in the right place on the palette. This way, he would not have to look for the menu around him and move to click on the button. Moreover, this mechanism avoids arm fatigue because the user does not have to lift his arm every time he has to press a button. Microsoft has done much research and innovation to avoid this kind of constraint. Finally, other elements could improve the user experience, such as a tutorial available with videos and an example of exercise.

Fourthly, the Unity program could check if the user is putting the box in the right place. Thus, once the user places the box, a check is performed on the structure to determine if the box is correctly placed. Such a feature requires significant development.

Finally, we would like to bring the last idea for improvement. So far, we assume that the location of the boxes before being picked up by the operator is known. This assumption allows us to place an arrow hologram above this location. Developing a program that automatically detects the boxes and determines their dimensions could be interesting. This program would be developed either directly with Unity to be applied in the Hololens software or by means of another device, as explained in this work with the example of a camera observing a conveyor belt and determining both the location and the dimensions of the boxes to be placed. This development would also be substantial.

5 Conclusion

Above, we have explained in detail the advantages and disadvantages of the developed product. The tests of the efficiency of the packing algorithm have also been performed and analyzed, as well as possible future works and improvements. Therefore, it would be redundant to address these topics again in the conclusion. This conclusion wishes to complement these pre-conclusions and address a more personal point of view of this work.

At the beginning of this work, we had no idea where to start, and it took several weeks of reflections to come up with a first version of the packing algorithm. In parallel, we studied the state of the art in this field and gradually discovered multiple ways to create such an algorithm. These researches allowed us to learn a lot about the development of a program but also about many optimization processes or algorithms that can be used in a multitude of fields.

Having no experience before this work with Unity and the programming language C#, the development of the Hololens part was the most complicated. We are therefore proud to have obtained this result which, although still having some problems, is the fruit of a single developer still very inexperienced.

The realization of an API was also a first and allowed us to improve our understanding considerably in this domain. Concerning the realization of a web interface, we had already realized in the past different web interfaces but using only vanilla HTML, CSS, and PHP. In this work, we had to adapt to a web framework, a first for us.

Finally, in order to finish this work positively, we would like to mention that this consequent work, started from nothing, played a key role in impressing a recruiter in an IT company which allowed us to get a job as a software developer. This was no easy task given the fact that our background mostly consists of economics and business knowledges.

6 Appendix

Appendix 1: Box overlap check pseudo code algorithm

Algorithm 2: Box Overlap Check

```
1  Function: rectangle_overlap
       Data: box1, box2, axis1, axis2
       Result: Return True if an overlap exists between the two boxes on
                the two axis space
2      begin
3          dim1 ⟵ box1.get_dimensions();
4          dim2 ⟵ box2.get_dimensions();
5          pos1 ⟵ box1.get_position();
6          pos2 ⟵ box2.get_position();
7          if (pos1[axis1] + dim1[axis1]) <= pos2[axis1]) or
             ((pos2[axis1] + dim2[axis1]) <= pos1[axis1]) or
             ((pos1[axis2] + dim1[axis2]) <= pos2[axis2]) or
             ((pos2[axis2] + dim2[axis2]) <= pos1[axis2])) then
8              Return True
9          end if
10         else
11             Return False
12         end if
13     end

14 Function: box_overlap
       Data: box1, box2
       Result: Return True if an overlap exists between the two boxes, i.e.
                the two boxes collide with eachother
15     if rectangle_overlap(box1, box2, Axis.width, Axis.depth) and
         rectangle_overlap(box1, box2, Axis.width, Axis.height) and
         rectangle_overlap(box1, box2, Axis.height, Axis.depth) then
16         Return True
17     end if
18     else
19         Return False
20     end if

21 Function: is_valid_position
       Data: box, extreme_point
       Result: Return True if the box can fit at the extreme point
                position with regard to the overlapping constraint with all
                other already placed boxes
22     placed_boxes ⟵
         all already placed boxes, either virtually or manually
23     for placed_box ∈ placed_boxes do
24         if box_overlap(box, placed_box) then
25             Return False
26         end if
27     end for
28     Return True
```

Appendix 2: User's manual of the program:

- Download the following code on GitHub: https://github.com/humi534/Master-Thesis-Project using the git clone command (the file may be too big to be downloaded on GitHub).
- Open two different terminals. Indeed, a problem persists in the code and the user interface still works only on a local address while the Hololens code must communicate with a private IP address (192.168).
- In the first terminal:
    - Go to the location of the downloaded file using the cd <PATH FILE> command (example: cd OneDrive/Documents/3DBP)
    - Activate the virtual environment with the following command: *VirtualEnv\scripts\activate*
    - Write the following command: *set FLASK_DEBUG=1*
    - Write the following command: *flask run*
- In the second terminal:
    - Go to the location of the file in the same way as with the first terminal
    - Activate the virtual environment in the same way as in the first terminal
    - Activate the virtual environment in the same way as with the first terminal
    - write the following command: *python app.py*
- To visualize the web interface, open a browser and go the address mentioned after the command *flask run*. It should be something like 127.0.0.1:5000
- To run the software on Hololens:
    - Open the Hololens and go to all apps
    - Select the app of the project
    - Press the start button

Appendix 3: Manual to upload a Unity program on the Hololens 2:

The following explanations show how we uploaded the software developed on Unity to the Hololens. There are multiple ways to do this, but some of them have been unsuccessful for us. So we give you the following instructions hoping that they will be useful if one day you want to do the same.

- Download the necessary extensions to develop an augmented reality program on Hololens
- Go to Builds Settings
- Select the Universal Windows Platform
    - Set *Target Device* to *Hololens*
    - Set *Architecture* to *ARM64*
    - Set *Minimum Platform Version* to *10.0.18362.0*
    - Set *Build and run on* to *USB Device*
    - Set *Build configuration* to *Release*
    - Press the *Build* button:
- A window opens to determine the location of the Build file
- Create a new Builds folder if it does not exist yet. For convenience, this folder can be placed in the same folder of the project
- Wait for the Build to be done, it should only take a few seconds
- Open the file explorer and go to the location of the Builds files in which you just placed the folder
- Right click on the file with the .sln extension and open this file with Visual Studio 2019
- Turn on the Hololens and connect to the same Wi-Fi network as the computer
- Click on the Project -> Properties -> Configuration Properties -> Debugging -> Machine Name
- Search for devices on the same network and select the Hololens that should appear
- Go back to the main interface
- Put *Release* in the top left slot
- Put *ARM64* to the right of the previously selected button
- Put *Remote Device* in the location just to the right of the previously selected button
- Go to the Debug tab
- Press *Start without debugging*
- Wait several minutes (the file will be compiled and deployed on Hololens)
- The project should be installed on the Hololens

Appendix 4:

The code of the project can be found with the following link:
https://drive.google.com/drive/folders/1uI2lO4ZzE41a3BDza5WbQk8fyAC61Sfw?usp=sharing

References

Amazon Web Service. (2022). *What is an API?* Retrieved July 07, 2022, from AWS: https://aws.amazon.com/what-is/api/?nc1=h_ls

Balakirsky, S., Proctor, F., Kramer, T., Kolhe, P., & Christensen, H. I. (2010). Using Simulation to Assess the Effectiveness of Pallet Stacking Methods. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 6472, pp. 336-349). Springer Berlin Heidelberg. doi:10.1007/978-3-642-17319-6_32

BigCommerce. (2022). *Ecommerce 101: The History and Future of Online Shopping*. Retrieved June 20, 2022, from BigCommerce: https://www.bigcommerce.com/articles/ecommerce/#types-of-ecommerce

Bischoff, E. E. (1991). Stability aspects of pallet loading. *OR Spektrum, 13*(4), 189-197. doi:10.1007/BF01719394

Bischoff, E., & Ratcliff, M. (1995). Issues in the development of approaches to container loading. *Omega (Oxford), 23*(4), 377-390. doi:10.1016/0305-0483(95)00015-G

Bortfeldt, A., & Gehring, H. (2001). A hybrid genetic algorithm for the container loading problem. *European journal of operational research, 131*(1), 143-161. doi:10.1016/S0377-2217(00)00055-2

Bortfeldt, A., & Wäscher, G. (2013). Constraints in container loading – A state-of-the-art review. *European journal of operational research, 229*, 1-20.

Bottani, E., & Vignali, G. (2019). Augmented reality technology in the manufacturing industry: A review of the last. *IIE transactions, 51*(3), 284-310. doi:10.1080/24725854.2018.1493244

Caprara, A., & Monaci, M. (2004). On the two-dimensional Knapsack Problem. *Operations research letters, 32*(1), 5-14. doi:10.1016/S0167-6377(03)00057-9

Carpenter, H., & Dowsland, W. (1985). Pracitcal considerations of the pallet-loading problem. *Journal of the Operational Research Society, 36*, 489-497.

Chan, F. T., Bhagwat, R., Kumar, N., Tiwari, M., & Lam, P. (2006). Development of a decision support system for air-cargo pallets loading problem: A case study. *Expert systems with applications, 31*(3), 472-485. doi:10.1016/j.eswa.2005.09.057

Chen, C., Lee, S., & Shen, Q. (1995). An analytical model for the container loading problem. *European journal of operational research, 80*(1), 68-76. doi:10.1016/0377-2217(94)00002-T

Coffman Jr, E. G., Csirik, J., Galambos, G., Martello, S., & Vigo, D. (2013). Bin Packing Approximation Algorithms: Survey and Classification. In D.-Z. Du, & P. Pardalos, *Handbook of Combinatorial Optimization* (Vols. 1-5, pp. 455-531). New York, NY: Springer New York.

Crainic, T. G., Perboli, G., & Tadei, R. (2008). Extreme Point-Based Heuristics for Three-Dimensional Bin Packing. *INFORMS journal on computing, 20*(3), 368-384. doi:10.1287/ijoc.1070.0250

Crainic, T. G., Perboli, G., & Tadei, R. (2009). TS2PACK: A two-level tabu search for the three-dimensional bin packing problem. *European journal of operational research, 195*(3), 744-760. doi:10.1016/j.ejor.2007.06.063

Davies, A., & Bischoff, E. E. (1999). Weight distribution considerations in container loading. *European journal of operational research, 114*(3), 509-527. doi:10.1016/S0377-2217(98)00139-8

De Castro Silva, J. L., Soma, N. Y., & Maculan, N. (2003). A greedy search for the three-dimensional bin packing problem: the packing static stability case. *International transactions in operational research, 10*(2), 141-153. doi:10.1111/1475-3995.00400

Drexl, A. (1988). A simulated annealing approach to the multiconstraint zero-one knapsack problem. *Computing, 40*(1), 1-8. doi:10.1007/BF02242185

Driessche, L. V. (2021, March 10). *Le covid a fait entrer les achats en ligne dans les mœurs*. Retrieved Mai 12, 2022, from L'Echo: https://www.lecho.be/entreprises/grande-distribution/le-covid-a-fait-entrer-les-achats-en-ligne-dans-les-m-urs/10290151.html

Dyckhoff, H. (1990). A typology of cutting and packing problems. *European Journal of Operational Research, 44*, 145–159.

Eley, M. (2002). Solving container loading problems by block arrangement. *European journal of operational research, 141*(2), 393-409. doi:10.1016/S0377-2217(02)00133-9

Eley, M. (2003). A bottleneck assignment approach to the multiple container loading problem. *OR Spectrum, 25*(1), 45-60. doi:10.1007/s002910200113

Elia, V., Gnoni, M. G., & Lanzilotto, A. (2016). Evaluating the application of augmented reality devices in manufacturing from a process point of view: An AHP based model. *Expert systems with applications, 63*, 187-197. doi:10.1016/j.eswa.2016.07.006

Fekete, S. P., Schepers, J., & van der Veen, J. C. (2007). An Exact Algorithm for Higher-Dimensional Orthogonal Packing. *Operations research, 55*(3), 569-587. doi:10.1287/opre.1060.0369

Gavish, B., & Pirkul, H. (1985). Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. *Mathematical programming, 31*(1), 78-105. doi:10.1007/BF02591863

Gavish, N., Gutiérrez, T. a., Rodríguez, J., Peveri, M., Bockholt, U., & Tecchia, F. (2015). Evaluating virtual reality and augmented reality training for industrial maintenance and assembly tasks. *Interactive learning environments, 23*(6), 778-798. doi:10.1080/10494820.2013.815221

Gendreau, M., Iori, M., Laporte, G., & Martello, S. (2006). A Tabu Search Algorithm for a Routing and Container Loading Problem. *Transportation science, 40*(3), 342-350. doi:10.1287/trsc.1050.0145

Gilmore, P. C., & Gomory, R. E. (1961). A Linear Programming Approach to the Cutting-Stock Problem. *Operations Research, 9*(6), 849-859. doi:10.1287/opre.9.6.849

Gilmore, P. C., & Gomory, R. E. (1963). A Linear Programming Approach to the Cutting Stock Problem: Part II. *Operations Research, 11*(6), 863-888. doi:10.1287/opre.11.6.863

Gilmore, P. C., & Gomory, R. E. (1965). Multistage Cutting Stock Problems of Two and More Dimensions. *Operations Research, 13*(1), 96-120. doi:10.1287/opre.13.1.94

Gilmore, P. C., & Gomory, R. E. (1966). The Theory and Computation of Knapsack Functions. *Operations Research, 14*(6), 1045-1074. doi:10.1287/opre.14.6.1045

Gzara, F., Elhedhli, S., & Yildiz, B. C. (2020). The Pallet Loading Problem: Three-dimensional bin packing with practical constraints. *European journal of operational research, 287*(3). doi:10.1016/j.ejor.2020.04.053

Healy, P., & Moll, R. (1996). A Local Optimization-based Solution to the Rectangle Layout Problem. *The Journal of the Operational Research Society, 47*(4), 523-537. doi:10.1057/jors.1996.58

Hemminki, J., Leipala, T., & Nevalainen, O. (1998). On-line packing with boxes of different sizes. *International journal of production research, 36*(8), 2225-2245. doi:10.1080/002075498192869

Junqueira, L., Morabito, R., & Sato Yamashita, D. (2012). Three-dimensional container loading models with cargo stability and load bearing constraints. *Computers & operations research, 39*(1), 74-85. doi:10.1016/j.cor.2010.07.017

Kang, K., Moon, I., & Wang, H. (2012). A hybrid genetic algorithm with a new packing strategy for the three-dimensional bin packing problem. *Applied mathematics and computation, 219*(3), 1287-1299. doi:10.1016/j.amc.2012.07.036

Kolla, S. S., Sanchez, A., & Plapper, P. (2021). Comparing effectiveness of paper based and Augmented Reality instructions for manual assembly and training tasks.

Lins, L., Lins, S., & Morabito, R. (2002). An n-tet graph approach for non-guillotine packings of n-dimensional boxes into an n-container. *European journal of operational research, 141*(2), 421-439. doi:10.1016/S0377-2217(02)00135-2

Liu, J., Yue, Y., Dong, Z., Maple, C., & Keech, M. (2011). A novel hybrid tabu search approach to container loading. *Computers & operations research, 38*(4), 797-807. doi:10.1016/j.cor.2010.09.002

Lodi, A., Martello, S., & Monaci, M. (2002). Two-dimensional packing problems: A survey. *European Journal of Operational Research, 141*(2), 241-252. doi:10.1016/S0377-2217(02)00123-6

Lodi, A., Martello, S., & Vigo, D. (2002). Heuristic algorithms for the three-dimensional bin packing problem. *European journal of operational research, 141*(2), 410-420. doi:10.1016/S0377-2217(02)00134-0

Mack, D., Bortfeldt, A., & Gehring, H. (2004). A parallel hybrid local search algorithm for the container loading problem. *International transactions in operational research, 11*(5), 511-533. doi:10.1111/j.1475-3995.2004.00474.x

Mallawaarachchi, V. (2017, July 8). *Introduction to Genetic Algorithms — Including Example Code*. Retrieved from Towards Data Science: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3#:~:text=A%20genetic%20algorithm%20is%20a,offspring%20of%20the%20next%20generation.

Martello, S., Pisinger, D., & Toth, P. (2000). New trends in exact algorithms for the 0–1 knapsack problem. *European journal of operational research, 123*(2), 325-332. doi:10.1016/S0377-2217(99)00260-X

Martello, S., Pisinger, D., & Vigo, D. (2000). The Three-Dimensional Bin Packing Problem. *Operations research, 48 (2)*, 256-267. doi:https://doi.org/10.1287/opre.48.2.256.12386

Martin, M., Hokama, P. H., Morabito, R., & Munari, P. (2020). The constrained two-dimensional guillotine cutting problem with defects: an ILP formulation, a Benders decomposition and a CP-based algorithm. *International journal of production research, 58*(9), 2712-2729. doi:https://doi.org/10.1080/00207543.2019.1630773

Microsoft. (2019, June 20). *Airbus drives innovation and accelerates production with Azure mixed reality and HoloLens 2*. Retrieved 06 28, 2022, from https://customers.microsoft.com/en-us/story/732143-airbus-manufacturing-azure-hololens2

Microsoft. (2022, May 25). *What is mixed reality?* Retrieved June 27, 2022, from Microsoft Documentation: https://docs.microsoft.com/en-us/windows/mixed-reality/discover/mixed-reality

Moustique. (2022, February 8). *E-commerce: quel impact sur l'environnement ?* Retrieved July 12, 2022, from Moustique: https://www.moustique.be/actu/environnement/2022/02/08/e-commerce-quel-impact-sur-lenvironnement-226979#:~:text=Multiplication%20des%20livraisons&text=Financ%C3%A9%20par%20Amazon%20(%20!)%2C%20un,que%20les%20achats%20en%20ligne%20%22.

Paquay, C. (2017). The three-dimensional rectangular Multiple Bin Size Bin Packing Problem with transportation constraints, A case study in the field of air transport.

Parreno, F., Alvarez-Valdes, R., Tamarit, J. M., & Oliveira, J. F. (2008). A Maximal-Space Algorithm for the Container Loading Problem. *INFORMS journal on computing, 20*(3), 412-422. doi:10.1287/ijoc.1070.0254

Pesce, M. (2021). *Augmented Reality – The Past, The Present and The Future*. Retrieved 06 27, 2022, from Interaction Design Foundation: https://www.interaction-design.org/literature/article/augmented-reality-the-past-the-present-and-the-future#:~:text=Augmented%20reality%20was%20first%20achieved,and%20smell%20to%20the%20viewer.

Peters, F., & Schyns, M. (2020). Improving e-commerce logistics with Augmented Reality and Machine Learning: The case of the 3D bin packing problem. *6th International AR VR Conference, April 30, 2020.*

Pisinger, D. (2002). Heuristics for the container loading problem. *European journal of operational research, 141*(2), 382-392. doi:10.1016/S0377-2217(02)00132-7

Ramos, A. G., Oliveira, J. F., & Lopes, M. P. (2016). A physical packing sequence algorithm for the container loading problem with static mechanical equilibrium conditions. *International transactions in operational research, 23(1-2)*, 215–238. doi:https://doi.org/10.1111/itor.12124

Rejeb, A., Keogh, J. G., Wamba, S. F., & Treiblmaier, H. (2020). The potentials of augmented reality in supply chain management: a state-of-the-art review. *Management review quarterly, 71*(4). doi:10.1007/s11301-020-00201-w

Ren, J., Tian, Y., & Sawaragi, T. (2011). A tree search method for the container loading problem with shipment priority. *European journal of operational research, 214*(3), 526-535. doi:10.1016/j.ejor.2011.04.025

RTBF. (2019, November 9). *Black Friday: achats impulsifs, retours, invendus et gros gâchis*. Retrieved June 20, 2022, from RTBF: https://www.rtbf.be/article/black-friday-achats-impulsifs-retours-invendus-et-gros-gachis-10078688?id=10078688

Schaus, P. (2009). Solving balancing and bin-packing problems with constraint programming.

Semerádová, T., & Weinlich, P. (2022). *Achieving business competitiveness in a digital environment : opportunities in e-commerce and online marketing.* (T. Semerádová, & P. Weinlich, Eds.) Cham, Switzerland: Springer.

Sheetz, M. (2019, 12 12). *Watch out, UPS. Morgan Stanley estimates Amazon is already delivering half of its packages*. Retrieved from CNBC: https://www.cnbc.com/2019/12/12/analyst-amazon-delivering-nearly-half-its-packages-instead-of-ups-fedex.html

Statistics & Data. (2021, May). *The Most Popular Programming Languages – 1965/2021 – New Update*. Retrieved June 20, 2022, from Statistics & Data: https://statisticsanddata.org/data/the-most-popular-programming-languages-1965-2021/#:~:text=The%20most%20popular%20programming%20language,third%20place%20with%20a%2010.31%25.

Suková, L., & Míková, L. (2022). The Relationship Between Product Placement and Shopping Intentions on Instagram. In T. Semerádová, & P. Weinlich, *Achieving Business Competitiveness in a Digital Environment* (pp. 177-206). Cham, Switzerland: Springer International Publishing. doi:https://doi.org/10.1007/978-3-030-93131-5_7

Terno, J., Scheithauer, G., Sommerweiß, U., & Riehme, J. (2000). An efficient approach for the multi-pallet loading problem. *European journal of operational research, 123*(2), 372-381. doi:10.1016/S0377-2217(99)00263-5

Thiel, J., & Voss, S. (1994). Some Experiences On Solving Multiconstraint Zero-One Knapsack Problems With Genetic Algorithms. *INFOR. Information systems and operational research, 32*(4), 226-242. doi:10.1080/03155986.1994.11732253

Wäscher, G., Haußner, H., & Schumann, H. (2007). An improved typology of cutting and packing problems. *European journal of operational research, 183*(3), 1109-1130. doi:10.1016/j.ejor.2005.12.047

Watching Alibaba. (2020). *No to Alibaba at Liege Airport!* Retrieved July 10, 2022, from Watching Alibaba: https://watchingalibaba.be/why/

Zhao, H., Zhu, C., Xu, X., Huang, H., & Xu, K. (2021). Learning practically feasible policies for online 3D bin packing. *Science China. Information sciences, 65*(1). doi:10.1007/s11432-021-3348-6

Zhao, X., Bennell, J. A., Bektaş, T., & Dowsland, K. (2016). A comparative review of 3D container loading algorithms. *International transactions in operational research, 23*(1-2), 287-320. doi:10.1111/itor.12094