КВАЛІФІКАЦІЙНА РОБОТА

МАГІСТРА

КРМ.АКСм - 11.00.00.000 ПЗ

група АКСм-24-1

Михайло КАРАЛАШ

Міністерство освіти і науки України
Івано-Франківський національний технічний університет нафти і газу
Факультет інформаційних технологій
Кафедра інформаційно-телекомунікаційних технологій та систем

Каралаш Михайло Романович
(прізвище, ім'я, по батькові)

УДК 658.7.07
(індекс)

МАГІСТЕРСЬКА РОБОТА

Проєктування автоматизованої системи оптимального розміщення вантажу при транспортуванні з оцінкою ефективності за просторовими, ваговими та маршрутними критеріями

(назва роботи)

Комп'ютеризовані системи управління та автоматика
(назва освітньої програми)

174 - Автоматизація, комп'ютерно – інтегровані технології та робототехніка
(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня М. Р. Каралаш
(підпис, ініціали та прізвище здобувача)

Науковий керівник к.т.н. доц. Л. О. Штаєр
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

д.т.н.,проф. Л.М. Заміховський
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

(повне найменування закладу вищої освіти)

Факультет Інформаційних технологій

Кафедра Інформаційно-телекомунікаційних технологій та систем

Освітній рівень магістр

Спеціальність 174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка

(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТТС

д.т.н. проф. Л.М. Заміховський

«____» _____2025 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ

Каралашу Михайлу Романовичу

(прізвище, ім'я, по батькові)

1. Тема роботи *Проєктування автоматизованої системи оптимального розміщення вантажу при транспортуванні з оцінкою ефективності за просторовими, ваговими та маршрутними критеріями*

керівник роботи *к.т.н.,доц. Штаєр Лідія Омелянівна*,

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від « 30 » *жовтня* 2025 року № 690 / 7

2. Строк подання здобувачем роботи *грудень 2025 р.*

3. Вихідні дані до роботи Матеріали переддипломної практики, аналіз науково-технічної літератури, програмний код Pack3D.dll

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз існуючих рішень та систем автоматизації процесу розміщення вантажу

2. Розробка евристичного алгоритму для автоматизованого розміщення вантажу. Вибір та обґрунтування апаратно-програмних засобів інструменту

3. Інтеграція та тестування системи (Client-Side Implementation). Експериментальна перевірка ефективності розробленої системи

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

1. МРК.АКСм – 11.00.00.001 - Дерево проекту. Схема функціональна.

2. МРК.АКСм – 11.00.00.002 - UML діаграма класів. Схема функціональна.

3. МРК.АКСм – 11.00.00.003 - Конфігурація стартової точки входу в ПЗ. Вигляд загальний.

4. МРК.АКСм – 11.00.00.004 - Схема UseCase Pack3D. Схема функціональна

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Штаєр Л. О.		

7. Дата видачі завдання: 30.11.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Термін виконання етапів роботи	Примітка
1	Аналіз алгоритмів та моделювання вимог: Формування математичної моделі та визначення багатофакторних критеріїв оптимальності.	30.08.2025 р.	Виконано
2	Проєктування алгоритму: Розробка евристики пакування, реалізація геометричного апарату (Guillotine Split) та механізму LIFO.	10.09.2025 р.	Виконано
3	Впровадження фізичних обмежень та скорингу: Налагодження логіки штабелювання, крихкості та корекція функції оцінки (Tandem Safety Check).	20.09.2025 р.	Виконано
4	Системна інтеграція та діагностика: Розробка SVG-візуалізатора і створення клієнта для загальної демонстрації	30.10.2025 р.	Виконано
5	Експериментальна валідація: Обґрунтування ефективності системи на тестових сценаріях та аналіз відповідності критеріям ТЗ.	05.11.2025 р	Виконано
6	Оформлення та фіналізація МР	01.12.2025 р.	Виконано

Здобувач _____ Каралаш М. Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Штаєр Л. О.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розроблено програмну бібліотеку автоматизованого розміщення гетерогенного вантажу в транспортних засобах. Проведено огляд сучасних евристичних алгоритмів тривимірного пакування. Зроблено аналіз переваг і недоліків існуючих підходів до оптимізації простору. Вдосконалення системи пакування полягає в розробці методу інкрементального розрахунку осьових навантажень з урахуванням черговості вивантаження (LIFO) та геометричних перешкод, а також розробленні на його базі програмного компонента на платформі .NET.

В результаті розробки система дозволить забезпечити надійне планування завантаження з дотриманням законодавчих вагових норм. Окрім цього передбачається впровадження функції пошарової візуалізації результатів для контролю процесу вантажних робіт.

ANNOTATION

A software library for automated placement of heterogeneous cargo in vehicles has been developed. A review of modern heuristic algorithms for three-dimensional packaging has been conducted. An analysis of the advantages and disadvantages of existing approaches to space optimization has been made. The improvement of the packaging system consists in developing a method for incremental calculation of axial loads taking into account the unloading order (LIFO) and geometric obstacles, as well as developing a software component on the .NET platform based on it.

As a result of the development, the system will allow for reliable loading planning in compliance with legislative weight standards. In addition, it is planned to implement a layered visualization function for monitoring the loading process.

РЕФЕРАТ

Розрахунково-пояснювальна записка: 69 сторінок, 28 рисунків, 6 таблиць, 28 посилань, 6 додатків.

Об’єктом дослідження є тривимірне розміщення гетерогенних вантажів у обмеженому об’ємі транспортного засобу.

Метою роботи є розробка та програмна реалізація автоматизованої системи оптимізації завантаження "Pack3D", що забезпечує високу щільність пакування з дотриманням фізичних обмежень (навантаження на осі, стабільність штабелювання) та логістичних критеріїв (черговість розвантаження LIFO).

У першому розділі проведено аналіз методів 3D-контейнеризації, формалізовано логістичні критерії та фізичні обмеження: геометрію, крихкість вантажів та ліміти навантаження на групи осей.

У другому розділі розроблено математичну модель розподілу навантаження на основі фізики важеля та запропоновано евристичний алгоритм розміщення з багатокритеріальним підходом.

У третьому розділі здійснено програмну реалізацію системи "Pack3D" на платформі .NET, описано архітектуру обчислювального ядра, розроблено модуль візуалізації SVG та модуль "UnloadingAnalyzer" для перевірки відповідності плану вимогам LIFO та оцінки часу розвантаження.

Ключові слова: ОПТИМІЗАЦІЯ ЗАВАНТАЖЕННЯ, ГЕТЕРОГЕННІ ВАНТАЖІ, ГІЛЬЙОТИННИЙ АЛГОРИТМ, НАВАНТАЖЕННЯ НА ОСІ, ВІЗУАЛІЗАЦІЯ ПЛАНІВ, LIFO, ЛОГІСТИЧНА ЕФЕКТИВНІСТЬ

ABSTRACT

Calculation and explanatory note: 69 pages, 28 figures, 6 tables, 28 references, 6 appendices.

The object of the study is the three-dimensional placement of heterogeneous cargo in a limited volume of a vehicle.

The purpose of the work is the development and software implementation of an automated loading optimization system "Pack3D", which provides high packing density while complying with physical constraints (axle load, stacking stability) and logistical criteria (LIFO unloading sequence).

The first section analyzes 3D containerization methods, formalizes logistical criteria and physical constraints: geometry, cargo fragility and load limits on axle groups.

The second section develops a mathematical model of load distribution based on lever physics and proposes a heuristic placement algorithm with a multi-criteria approach.

In the third section, the software implementation of the "Pack3D" system on the .NET platform is carried out, the architecture of the computing core is described, the SVG visualization module and the "UnloadingAnalyzer" module are developed to check the compliance of the plan with LIFO requirements and estimate the unloading time.

Keywords: LOADING OPTIMIZATION, HETEROGENEOUS LOADS, GUILLOTINE ALGORITHM, AXLE LOADING, PLAN VISUALIZATION, LIFO, LOGISTICS EFFICIENCY

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

3D-BPP – (Three-Dimensional Bin Packing Problem) тривимірна задача пакування;

API – (Application Programming Interface) інтерфейс прикладного програмування;

CLP – (Container Loading Problem) задача завантаження контейнера;

CoG – (Center of Gravity) центр мас;

CSG – (Constructive Solid Geometry) конструктивна блокова геометрія;

CSV – (Comma-Separated Values) значення, розділені комами;

DOM – (Document Object Model) об’єктна модель документа;

ERP – (Enterprise Resource Planning) планування ресурсів підприємства;

GC – (Garbage Collector) збирач сміття;

JSON – (JavaScript Object Notation) запис JavaScript об’єктів;

LIFO – (Last In, First Out) останнім прийшов - першим пішов;

LINQ – (Language Integrated Query) інтегрований у мову запит;

LTL – (Less Than Truckload) вантаж менше повної вантажопідйомності;

MER – (Maximal Empty Rectangles) максимальні порожні прямокутники;

NP – (Nondeterministic Polynomial) недетермінований поліноміальний час;

SVG – (Scalable Vector Graphics) масштабована векторна графіка;

TMS – (Transportation Management System) система управління транспортом;

UML – (Unified Modeling Language) уніфікована мова моделювання;

ВСТУП

Логістика сьогодні - це не просто допоміжна функція, а справжня кровоносна система бізнесу, від якої без перебільшення залежить виживання компанії. Якщо говорити прямо: те, наскільки грамотно ви завантажили фуру, безпосередньо б'є по кишені - визначає собівартість товару, чи доїде він цілим і як швидко опиниться у клієнта. Головний біль транспортної логістики - це запхати в обмежений простір кузова якомога більше товару. Але, як показує практика, просто грати в «Тетріс» вже недостатньо. Реальність підкидає купу фізичних проблем: треба думати, щоб не перевантажити конкретну вісь трака, щоб вантаж не полетів шкереберть на повороті, і щоб дотримувався порядок вивантаження (правило LIFO). А ще ж є холодильні установки, колісні арки та інші «сюрпризи» всередині контейнера, які з'їдають корисний простір.

Актуальність теми. Зазвичай процес завантаження тримається на плечах досвідчених комірників - вони роблять це "на око" і часто досить непогано. Але ситуація різко змінюється, коли приходить нестандартний, різнорідний вантаж. Тут людська інтуїція починає буксувати. В таких умовах ручне планування вже не працює ефективно: це довго, дорого і загрожує помилками. Дуже ймовірно, що саме тут людський фактор стає вузьким місцем, породжуючи ряд проблем:

- нераціональне використання внутрішнього об’єму транспортного засобу. Це призводить до фактичного перевезення невикористаного простору, що змушує збільшувати кількість рейсів і, відповідно, витрати на паливо;
- порушення вагових норм. Нерівномірний розподіл ваги - це пряме порушення законодавства будь-якої країни, що неминуче призводить до штрафів за перевантаження окремих осей кузова, навіть якщо загальна маса вантажу в межах норми;
- пошкодження вантажу. Ігнорування параметрів крихкості та допустимого окремої одиниці штабелювання спричиняє матеріальні збитки.

Метою і завданням дослідження є підвищення безпеки та економічної ефективності вантажних перевезень шляхом автоматизації процесу планування. Це

дозволяє мінімізувати вплив людського фактору. Для досягнення поставленої мети було вирішено низку наступних завдань:

- детально розглянути наявні підходи до розв'язання задачі пакування контейнерів, зосередившись при цьому на евристичних алгоритмах та традиційних методах керування вільним простором, які застосовуються під час формування планів завантаження;
- виконати формалізацію фізичних обмежень транспортного засобу, включаючи модель розподілу осьових навантажень;
- розробити математичну модель розміщення гетерогенного вантажу, що враховує геометричні перетини, штабелювання вантажу та заборонені зони в кузові трака;
- створити алгоритм пакування на базі підходу Best-Fit Decreasing із застосуванням модифікованого методу гільйотинного розрізу (Guillotine Split) та атомарної багатокритеріальної функції оцінки;
- реалізувати розроблений алгоритм у вигляді окремої бібліотеки на платформі .NET Core та розробити SVG інструментарій для візуалізації результатів з обов'язковим логуванням результатів.

Об'єктом дослідження є процес формування вантажних партій та їх розміщення у вантажних відсіках транспортних засобів.

Предметом дослідження є математичні моделі, методи оптимізації та класичні алгоритми тривимірного пакування, а також методи розрахунку фізичних характеристик завантаженого транспортного засобу.

Наукова новизна роботи полягає у доопрацюванні підходу до евристичного пакування за рахунок включення динамічного розрахунку моментів сил безпосередньо в цільову функцію алгоритму. Це дає змогу під час пошуку місця для розміщення вантажу орієнтуватися не лише на щільність укладання, а й на положення центру мас відносно оптимальної точки рівноваги. У результаті сформований план завантаження є не тільки компактним, а й безпечним з точки зору стійкості транспортного засобу під час руху.

Практична цінність. Все зводиться до реального застосування. Я розробив бібліотеку Pack3D не як простий прототип, а як інструмент, готовий до роботи. Її можна спокійно "підшити" до вже працюючих на підприємстві систем - чи то складських WMS, чи транспортних TMS. Головна фішка полягає в тому, що тій чи іншій компанії не доведеться ламати власну IT-архітектуру чи переписувати софт з нуля. Бібліотека інтегрується безболісно, як пазл, у вже налагоджені бізнес-процеси.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ФОРМУВАННЯ ТЕХНІЧНИХ ВИМОГ

Задача розміщення різногабаритних вантажів є цілком класичною проблемою комбінаторної оптимізації, яка має широке застосування в логістичній теорії. Ця задача класифікується як NP-важка задача, що обумовлюється експоненційним зростанням обчислювальної складності при збільшенні кількості пакунків. Це створює справді суттєвий розрив між теоретичними, здатними знайти ідеальне рішення для невеликої кількості елементів, математичними моделями, та реальними потребами транспортної логістики, де необхідно оперувати тисячами різноманітних вантажних одиниць у режимі реального часу, враховуючи не лише геометричні аспекти, а й фізичні. Тому вкрай важливим етапом проектування спеціалізованого програмного забезпечення є не тільки вибір того чи іншого методу оптимізації, а й глибоке розуміння аспектів природи середовища, в якому цей метод буде застосовано в подальшому [1].

Розділ присвячено формуванню теоретичної бази дослідження. Тут проаналізовано існуючі алгоритмічні рішення та обґрунтовано їхню обмеженість у контексті розроблюваної системи. Проте для створення дієвого інструменту недостатньо лише математичних даних - необхідний перехід до комплексних інженерних моделей.

До таких обмежень відносяться допустиме навантаження на осі транспортного засобу, забезпечення статичної стійкості вантажу під час руху, а також дотримання правильної послідовності розвантаження. Ігнорування цих факторів на етапі аналізу може призвести до появи планів завантаження, які математично виглядають правильними, але на практиці їх неможливо реалізувати. Основною метою розділу є формалізація вхідних даних, що слугуватимуть основою для побудови математичної моделі та алгоритмічних рішень у наступних розділах роботи. Також розглядаються підходи сучасних досліджень до інтеграції логістичних бізнес-правил безпосередньо в процес геометричного розміщення вантажу та визначаються ключові технічні характеристики, необхідні для

створення конкурентоспроможної системи автоматизованого планування завантаження.

1.1. Огляд методів 3D-контейнеризації та алгоритмів пакування

Задача 3D-пакування належить до класу NP-важких, У виробничих умовах значно доцільнішим є застосування підходів, здатних генерувати якісний результат у регламентовані часові рамки. Цей компроміс є цілком свідомим: потенційна втрата незначного відсотка щільності заповнення повністю компенсується оперативністю системи, що дозволяє уникати тривалих простоїв при розрахунках. Адже в динаміці логістичних процесів швидкість прийняття рішень часто має вищий пріоритет, ніж абстрактна математична досконалість плану.

Важливим аспектом є те, що алгоритм не може функціонувати у сам по соб. Він зобов'язаний працювати зконтекстом. Під «контекстом» маються на увазі ризики зміщення вантажу під дією інерції, обмеження щодо допустимих орієнтацій та специфічну геометрію контейнера. І хоч введення цих змінних суттєво ускладнює розрахункову модель, саме такий підхід перетворює її на потужний інструмент.

1.1.1 Аналіз евристичних підходів та їх застосування для гетерогенного вантажу

Ефективність вирішення задачі пакування (3D-BPP) для гетерогенного вантажу критично залежить від початкової обробки вхідних даних, оскільки всі евристичні алгоритми є чутливими до послідовності надходження об'єктів при подальшому обчисленні. Хаотичне завантаження коробок різного розміру так чи інакше призведе до фрагментації простору та утворення порожнин. На більш пізніх етапах пакування, ці порожнини неможливо заповнити оптимальними пакунками. Таким чином, фундаментом для будь-якого оптимального алгоритму є попереднє сортування даних. Найбільш корисною є відома практика - «Спадаюча послідовність» (Decreasing Sequence), яка передбачає впорядкування вантажу за ключовими параметрами [2].

Проте для логістичних задач просте геометричне сортування ніколи не було достатнім. Як показано на рис. 1.1, для забезпечення стабільності та дотримання маршруту необхідно застосовувати багаторівневе сортування. Скажімо, первинним ключем виступає індекс зупинки (Delivery Stop Index), а вторинним - вага (для розміщення важких об'єктів знизу), і лише потім враховується сама геометрія пакунка [3].

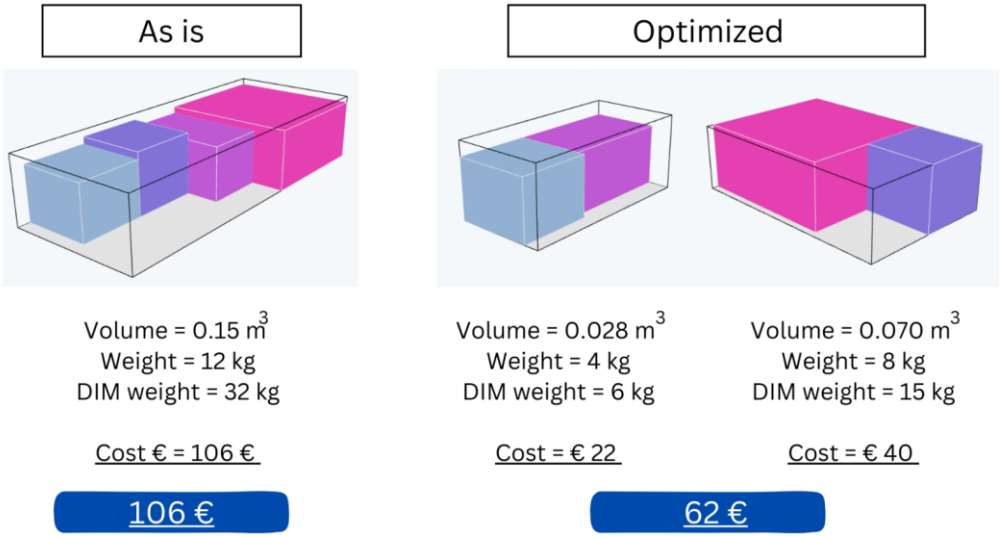

Рисунок 1.1 – Оптимізація через багатокритеріальне сортування вантажу [1]

Серед основних алгоритмів розміщення, базовим і найшвидшим представником є метод First-Fit (FF) або ж «Перший підхожий». Принцип його роботи полягає у розміщенні поточного об'єкта в найпершу знайдену позицію, яка задовольняє його геометричні обмеження. Тобто перебору всіх можливих варіантів не відбувається. Таким чином, головною перевагою FF є вкрай низька обчислювальна складність, що дозволяє обробляти тисячі або й десятки тисяч об'єктів за лічені секунди. Однак варто зауважити, що, як видно з порівняльної діаграми (рис. 1.2), цей метод дуже часто є малоефективним з точки зору результативності на фоні інших, більш складних алгоритмів.

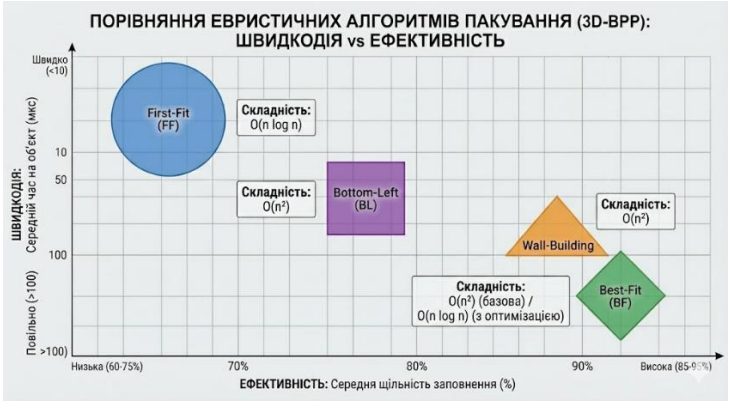

Рисунок 1.2 – Порівняльна характеристика ефективності до швидкодії основних евристичних алгоритмів

Більш досконалим підходом до вирішення проблеми є алгоритм Best-Fit (BF) або ж «Найкращий підхожий». На відміну від вищезгаданого FF, цей метод аналізує всі доступні вільні простори. В результаті, обирає лише той, де місце після розміщення об'єкта буде мінімальним. Така тактика дещо підвищує щільність пакування. Варто сказати, цей підхід також зменшує загальний об'єм порожнин. Проте Best-Fit має суттєвий недолік. Він схильний до сильного розбиття вільного простору на дрібні, часто непридатні для повторного використання ділянки, розпорошені по всьому контейнеру [5].

Третім класичним методом є Bottom-Left (BL) («Нижній лівий»). Його евристика побудована на геометричних координатах. Тобто алгоритм намагається розмістити об'єкт якнайближче до початку координат (умовно - в самий низ, вглиб і вліво траку). Цей підхід інтуїтивно зрозумілий і показує високі результати у двовимірних задачах розкрою. Однак, у тривимірному просторі "чистий" BL має обмеження. Він не враховує «гравітаційну стійкість» і може створювати ситуації, коли об'єкт просто "зависає" в повітрі, якщо під ним є вільний простір, який раніше виявився недоступним для заповнення через специфічну геометрію сусідніх коробок або коробок знизу (рис. 1.3). Тому для реальних вантажів цей метод потребує значних модифікацій з перевіркою опорної поверхні [6]. Використовувати BP в класичному вигляді не є оптимальним методом вирішення задачі.

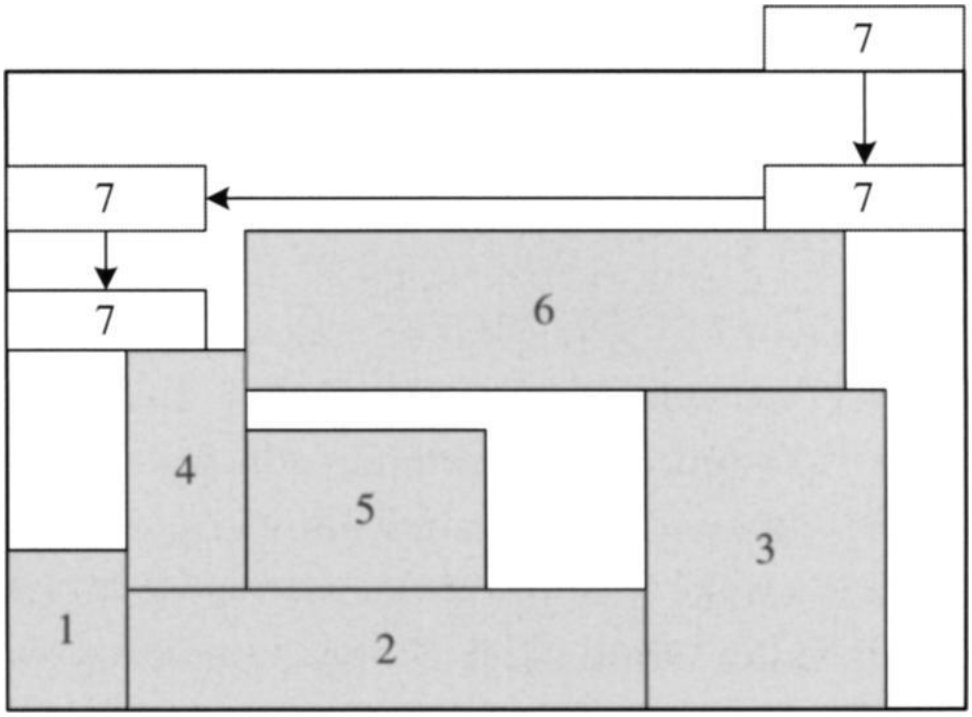

Рисунок 1.3 – Принцип роботи евристичного алгоритму Bottom-Left Fit

Для завантаження транспортних засобів (вантажівок, контейнерів) найбільш адаптованим є четвертий підхід - Wall-Building («Побудова стінок») або пошарове формування. На відміну від вищезгаданих методів, які розглядають простір як єдиний, неподільний об'єм, WB-алгоритм віртуально розрізає контейнер на вертикальні шари (стінки) певної глибини. Заповнення відбувається поетапно в декілька етапів. Спочатку формується одна цілісна стіна з коробок, і лише після її завершення починається формування наступної стінки (рис. 1.4). Це дозволяє досягти високої стабільності вантажу при русі або гальмуванні транспорту та спрощує процес розвантаження. Хоча цей метод може поступатися Best-Fit у щільності пакування дрібних коробок, але на цей час, він є стандартом для палетованих та коробкових вантажів у логістиці [7].

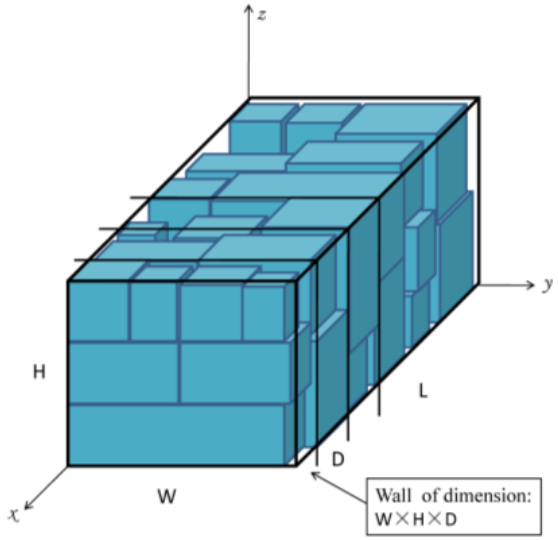

Рисунок 1.4 – Візуалізація роботи алгоритму Wall Builder

Порівняльний аналіз цих підходів показує, що не існує універсального алгоритму. First-Fit виграє у швидкості поки Best-Fit - у щільності для компактних наборів, Bottom-Left забезпечує геометричну впорядкованість, а Wall-Building гарантує фізичну стабільність вантажу (табл. 1.1)

Таблиця 1.1 – Порівняльна характеристика основних евристичних методів

Характеристика	First-Fit	Best-Fit	Bottom-Left (BL)	Wall-Building
Швидкодія	Висока	Середня	Середня	Низька
Щільність (однорідні)	Низька	Висока	Середня	Дуже висока
Щільність (гетерогенні)	Дуже низька	Середня	Середня	Висока
Фрагментація пам'яті	Низька	Середня	Висока	Низька
Складність реалізації	Низька	Середня	Середня	Висока
Підтримка обмежень	Низька	Середня	Низька	Висока

У контексті даної роботи, де вантаж є гетерогенним, використання одного "чистого" методу є неефективним. Тому було обрано використовувати гібридний підхід: модифікований Best-Fit із елементами перевірки опорної поверхні (як у BL), що працює в межах пріоритетних груп, сформованих стратегією Decreasing Sequence. Це дозволить поєднати щільність укладання з вимогами до безпеки транспортування [8]. Алгоритми, типу WB та FF не використовуються в силу своєї складності та простоти відповідно.

1.1.2 Огляд апарату геометричних розрізів (Maximal Rectangles, Guillotine Split) для ефективного управління вільним простором

Ефективне управління даними про доступний об'єм є чи не найскладнішою частиною програмної реалізації 3D-пакування. Фізичний простір контейнера є неподільним. Головне завдання алгоритму полягає у його дискретизації, тобто представленні у вигляді обмеженого набору доступних для розміщення зон. Від вибору методу декомпозиції вільного простору напряму залежить як теоретично досяжна щільність пакування, так і реальна швидкодія системи при обробці великих масивів даних [9]. В сучасній літературі виділяють два фундаментально різні підходи до вирішення цієї задачі.

Першим виступає Maximal Empty Rectangles. Його особливість та водночас відмінність полягає у принципі роботи з вільним простором. Замість хаотичного пошуку місця, MER-алгоритм динамічно підтримує структурований реєстр усіх доступних для завантаження зон. Причому зауважу, що мова йде не про випадкові фрагменти об'єму, натомість - про виключно «максимальні» паралелепіпеди. Маються на увазі, найбільші цілісні геометричні фігури, які можна вписати в незайнятий простір.

Коли новий об'єкт розміщується у вибраній позиції, він, як правило, перетинається одночасно з декількома існуючими максимальними прямокутниками. Алгоритм зобов’язаний виявляти всі такі перетини та видалити «пошкоджені» старі прямокутники. На їх місце повинен генеруватися набір нових,

менших прямокутників, що утворилися навколо розміщеного об'єкта. У тривимірному просторі розміщення навіть одного об'єкта може призвести до генерації від 1 до 6 нових потенційних зон (над, під, зліва, справа, спереду, позаду об'єкта), які часто значно перекривають одна одну, як показано на рисунку 1.5.

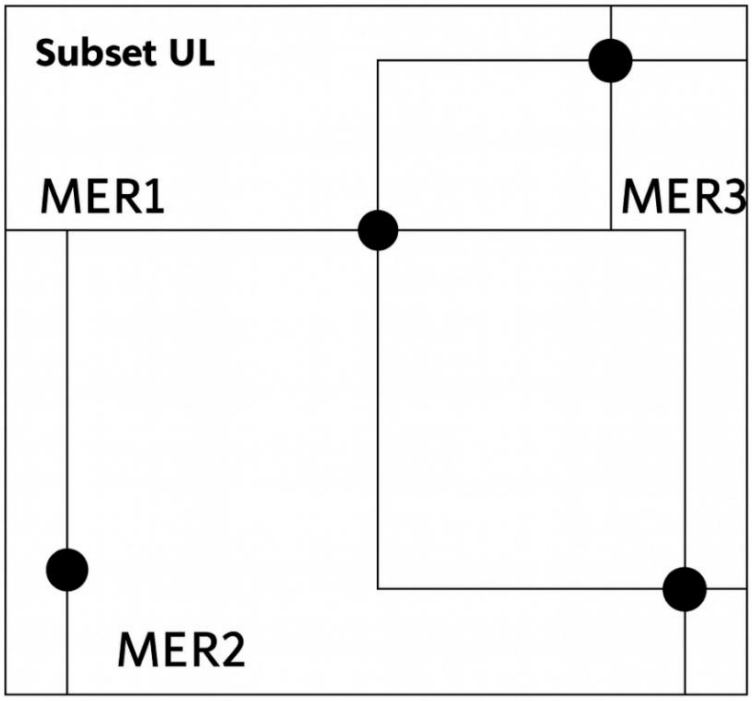

Рисунок 1.5 – Принцип роботи підходу Maximal Empty Rectangles

Отже Maximal Empty Rectangles (MER) демонструє як переваги так і значні недоліки:

- перевага: забезпечує найвищу теоретичну щільність пакування, оскільки зберігає інформацію про всі можливі варіанти розміщення, не відкидаючи жодної потенційно корисної ділянки простору;
- недолік: схильність до «комбінаторного вибуху». Кількість прямокутників у списку може зростати нелінійно відносно кількості розміщених об'єктів. У найгіршому випадку, буде отримано квадратичну залежність $O(N^2)$. Для великих завантажень (понад 500–1000 коробок чи більше) це критично сповільнює процес пошуку місця, роблячи метод непридатним для run-time систем [10].

Метод Guillotine Split базується на ідеї розділення простору ортогональними площинами. Ця сукупність площин проходить через грані розміщеного об'єкта. Основна відмінність від MER полягає в тому, що вільний простір завжди представлений набором підпросторів, що не перетинаються.

При розміщенні об'єкта у вибраному вільному прямокутнику (зазвичай у його куті), залишок цього простору розрізається на фіксований набір нових, менших прямокутників. До прикладу, при розміщенні коробки в нижньому лівому задньому куті доступного об'єму, простір, що залишився, зазвичай ділиться на три нові незалежні зони: простір праворуч від коробки, простір над нею та простір перед нею (рис. 1.6).

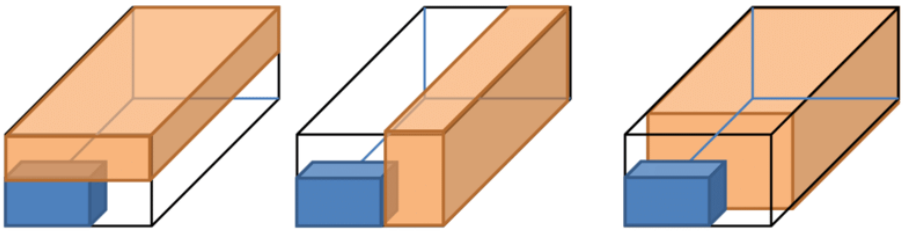

Рисунок 1.6 – Візуалізація принципу Guillotine Split

Загалом існує кілька евристик вибору порядку розрізів (наприклад, спочатку за найкоротшою віссю, або ж завжди спочатку горизонтальний розріз), що впливає на форму майбутніх вільних зон. Це має як і переваги, так і недоліки:

- перевага: Це гарантує лінійну складність оновлення простору O(N). Оскільки вільні зони ніколи не перетинаються, при розміщенні нового об'єкта потрібно оновити лише один вільний прямокутник, в який він був поміщений, замінивши його на декілька нових. Такий підхід значно спрощує перевірки перетину колізії та Зменшує час обчислення;
- недолік: Жорсткість розрізів. Цей метод може призвести до ситуації, коли дві сусідні вільні зони в теорії могли б вмістити великий об'єкт, але через наявність віртуальної площини розрізу між ними використання цього

об'єднаного простору стає неможливим без складних процедур. типу «злиття» (merging) [11].

Враховуючи специфічні вимоги даної магістерської роботи - високу швидкодію (час розрахунку до 10 секунд для 2000 коробок) та необхідність враховувати складні фізичні обмеження, метод Guillotine Split є значно привабливішим. Проте варто наголосити, що він може дещо поступатися MER у теоретичній щільності. Тим не менш, його передбачувана продуктивність та, що найважливіше, чітка структура неперетинних просторів є критичними перевагами. Така структура дозволяє легко реалізовувати «віртуальні стіни» для забезпечення LIFO (просто забороняючи доступ до певних зон до завершення поточного шару) та ефективно «вирізати» із загального об'єму заборонені зони трейлера (наприклад, колісні арки або холодильну установку). Більше того, модифікований принцип Guillotine Space можна використати для реалізації SVG шарів в майбутньому.

1.2. Аналіз та формалізація критичних логістичних критеріїв ефективності

Система пакування повинна не просто виконувати математичні алгоритмічні задачі, а й органічно вписуватися в загальну логіку роботи підприємства.

У цьому підрозділі розглядаються ключові аспекти, що визначають практичний контекст застосування такого програмного рішення. Вони окреслюють набір проблем, які система має вирішувати, але не заглиблюються у технічні деталі реалізації - конкретні механізми будуть описані в наступних розділах.

1.2.1 Формалізація критерію просторової ефективності (Volume Utilization)

Ключовим показником якості будь-якого рішення задачі 3D-пакування виступає коефіцієнт використання об'єму U_{vol}. У найбільш простому теоретичному випадку він визначається як відношення сумарного об'єму вантажу до повного об'єму контейнера. Однак, у реальних логістичних умовах такий підхід є неточним.

Він ігнорує конструктивні особливості транспортного засобу. Таким чином, критично важливо розрізняти «номінальний об'єм» траку (загальні внутрішні габарити) та «корисний об'єм» - простір, фізично доступний для розміщення вантажу. Формалізований критерій ефективності, що використовується в даній роботі, розраховується за наступною формулою:

$$U_{vol} = \frac{\sum V_{cargo}}{V_{trailer} - \sum V_{forbidden\_zones}}, \quad (1.1)$$

де $\sum V_{cargo}$ - сумарний об'єм усіх успішно завантажених об'єктів, $V_{trailer}$ - номінальний внутрішній об'єм трейлера, а $\sum V_{forbidden\_zones}$ - сумарний об'єм зон, недоступних для пакування.

Ігнорування $\sum V_{forbidden\_zones}$ призводить до генерації нереалістичних планів завантаження, які неможливо виконати на практиці. До таких зон у стандартних напівпричепах відносяться колісні арки, що розміщені всередині кузова або ж простір, зайнятий холодильною установкою (для рефрижераторів), механізми ролетних воріт біля стелі, а також технологічні відступи, необхідні для безпечного відкриття дверей. Ефективна система пакування повинна моделювати ці зони як віртуальні статичні перешкоди, віднімаючи їх з доступного пулу вільного простору. ще до початку роботи алгоритмів розміщення.

Важливо зазначити, що хоч високий інтегральний показник U_{vol} є необхідністю, але також він є недостатньою умовою якісного пакування. Цей критерій не враховує розподіл вантажу в просторі. Також можлива ситуація, коли вже досягнуто високого відсотка заповнення (наприклад, 90% чи більше), але весь вантаж розміщено щільним шаром на підлозі, залишаючи великі порожнечі («повітряні кишені») під стелею по всій довжині трейлера. Таке завантаження є неефективним з точки зору логістики та нестійким з точки зору безпеки в цілому. Тому просторова ефективність повинна додатково аналізуватися пошарово (або посекційно вздовж довжини трака) для забезпечення рівномірного заповнення

кузова по висоті, що є критичним фактором для забезпечення фізичної стабільності вантажу під час його транспортування [12].

1.2.2 Формалізація критерію вагової безпеки (Axle Load Distribution)

Першочергове дотримання вагових обмежень є найбільш важливою вимогою в контексті нашої задачі Її порушення тягне за собою не лише значні фінансові проблеми, але й загрозу безпеці дорожнього руху через погіршення керованості та збільшення гальмівного шляху. У Сполучених Штатах Америки регулювання здійснюється на федеральному рівні та рівні штатів, де ключовим стандартом є «Федеральна формула мостових ваг» (Federal Bridge Gross Weight Formula), закріплена у кодексі 23 CFR § 658. Для типової конфігурації американського п'ятивісного сідлового тягача (тривісний тягач 6x4 та двовісний напівпричіп типу «тандем») критичними є три точки контролю, показані на рисунку 1.7 одиночна кермова вісь тягача (Steer Axle) - 1, здвоєна ведуча група осей (Drive Tandem) - 4 та здвоєна група осей напівпричепа (Trailer Tandem) - 9. Стандартні федеральні ліміти для міжштатних магістралей складають 20 000 фунтів (≈9070 кг) на одиночну вісь та 34 000 фунтів (≈15420 кг) на тандемну групу, хоча технічні специфікації конкретних моделей тягачів (наприклад, Freightliner Cascadia або Kenworth T680) можуть мати інші конструктивні обмеження, які також необхідно враховувати [13].

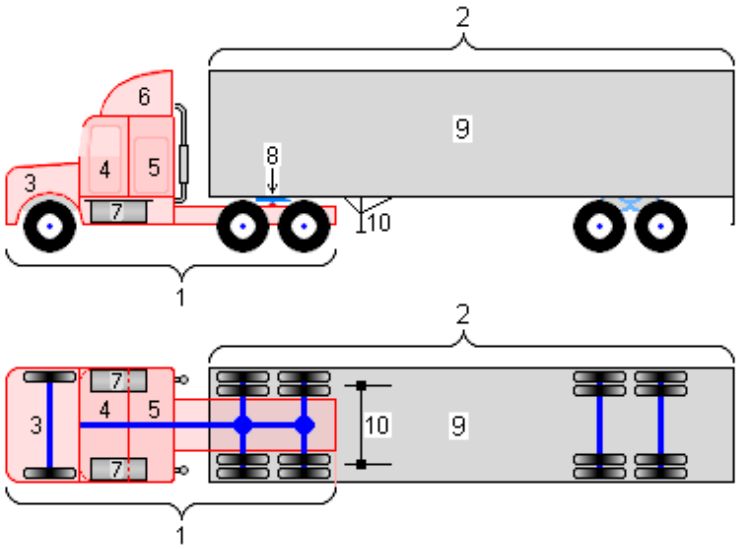

Рисунок 1.7 – Схема типового сідлового тягача

Розроблювана математична модель розподілу навантаження базується на статичній механіці твердого тіла. Вантаж у кузові не можна розглядати просто як сумарну масу. Критичне значення має розташування його спільного центру мас (Center of Gravity - CoG). Будь-яке зміщення CoG вантажу вздовж поздовжньої осі трейлера змінює баланс сил у системі важелів, якою є сам тягач. Виробники причіпної техніки (наприклад, Wabash National або Great Dane) надають спеціалізовані діаграми (рис. 1.8) розподілу навантаження, які визначають допустимий діапазон положення CoG для певної маси вантажу, вихід за межі якого гарантовано призводить до перевантаження однієї з груп осей [14]. Завдання алгоритму - утримати CoG у цьому безпечному «вікні» протягом усього процесу завантаження з подальшою валідацією результату.

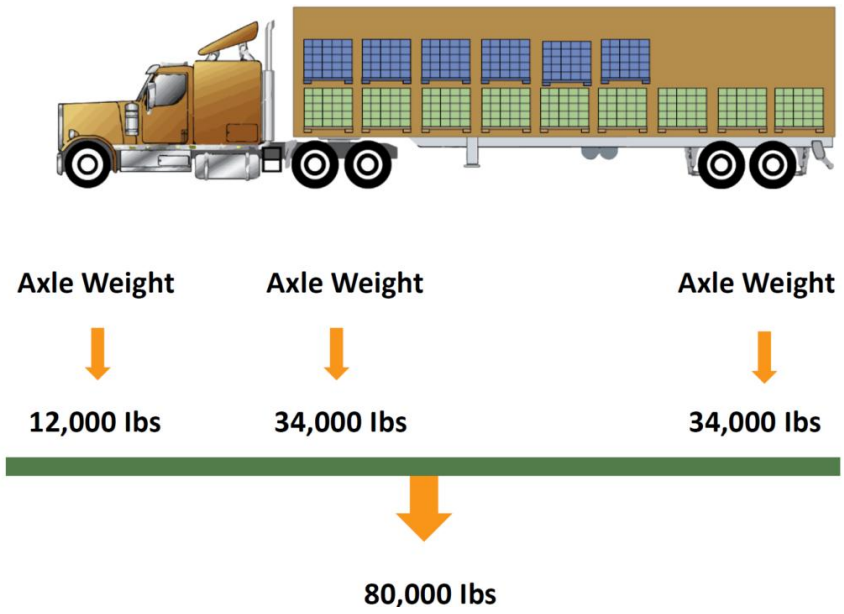

Рисунок 1.8 – Допустимий діапазон положення CoG

Розрахунок сил у системі будь-якої вантажівки відбувається у два взаємопов’язані, послідовні етапи, що пов'язано з наявністю шарнірного з'єднання - Kingpin).

На першому етапі напівпричіп розглядається як звичайна балка, що спирається на дві точки: плиту шворня спереду та геометричний центр колісного візка ззаду. Вага розміщеного вантажу створює силу тяжіння та обертальний

момент відносно цих точок опор. Використовуючи рівняння рівноваги моментів, система розраховує реакцію опори на тандемі причепа *(R_{TR})* та, що найважливіше, навантаження, яке передається на шворінь *(R_{KP})*.

На другому етапі розраховується розподіл навантаження всередині самого тягача. Сила R_{KP}, розрахована на попередньому кроці, діє як точкове навантаження на раму тягача. Оскільки «сідло», зазвичай, трохи зміщене відносно середини колісної бази тягача в бік ведучої групи осей, ця сила нерівномірно розподіляється між кермовою віссю (Steer) та ведучим тандемом (Drive Tandem) [15]. Критично важливим є те, що надмірне зміщення вантажу вперед у причепі може не лише перевантажити ведучі осі, але й небезпечно розвантажити кермову вісь, що призводить до втрати зчеплення передніх коліс з дорогою і втрати керованості. Це може привести до ДТП з подальшими економічними збитками.

Враховуючи високу ціну помилки, розроблювана система 3D-пакування не може перевіряти вагові обмеження лише в кінці розрахунку. Алгоритм повинен виконувати ці обчислення інкрементально. Тобто після віртуального розміщення кожної окремої коробки відбувається миттєвий перерахунок спільного CoG загального вантажу. Після цього відбувається перевірка реакцій опор на всіх трьох контрольних точках.

Будь-який варіант розміщення, який призводить до перевищення законодавчого або технічного ліміту на будь-якій групі осей, повинен негайно відсікатися евристикою як неприпустимий. Це має відбуватися, незалежно від того, наскільки він вигідний з точки зору геометричної щільності [16].

1.2.3 Формалізація маршрутного критерію (LIFO) та вимог до послідовності завантаження

Так званий, принцип LIFO (Last In, First Out - «останнім прийшов, першим пішов») диктує, що вантаж, призначений для першої точки маршруту, повинен бути фізично доступним для вивантаження відразу після відкриття дверей трейлера, без необхідності переміщення товарів для інших, більш пізніх зупинок.

Порушення цього правила призводить до виникнення так званих «блокувань», коли коробка *i* (для пізньої зупинки) перекриває доступ до коробки *j* (для поточної зупинки). Недотримання цього принципу, вимагає від водія виконання операцій з розвантаження та повторного завантаження зайвого товару на рампі, що критично збільшує час простою та ризик пошкодження вантажу [17].

Геометрично ця вимога трансформується у нескладну задачу зонування простору вздовж поздовжньої осі трейлера (X). На відміну від статичних складів, у кузові вантажівки зони не є фіксованими метрично. Тобто їхні межі (L_i) визначаються динамічно, виходячи з сумарного об'єму вантажу для кожної зупинки. Якщо прийняти початок координат $(X=0)$ біля передньої стінки (кабіни), а кінець ($X=L_{total}$) біля дверей, то для маршруту із зупинками 1, 2, ..., N (де 1 - перша зупинка) розподіл простору має бути строго інверсним до порядку відвідування:

- вантаж для останньої зупинки N розміщується у «глибокій зоні»: $x \in [0, L1]$;
- ... ;
- вантаж для першої зупинки 1 розміщується у «зоні біля дверей»: $x \in [L_{N-1}, L_{total}]$.

Візуалізація цього принципу наведена на схемі зонування (рис. 1.9), де кольорами позначено приналежність вантажу до конкретних зон розвантаження.

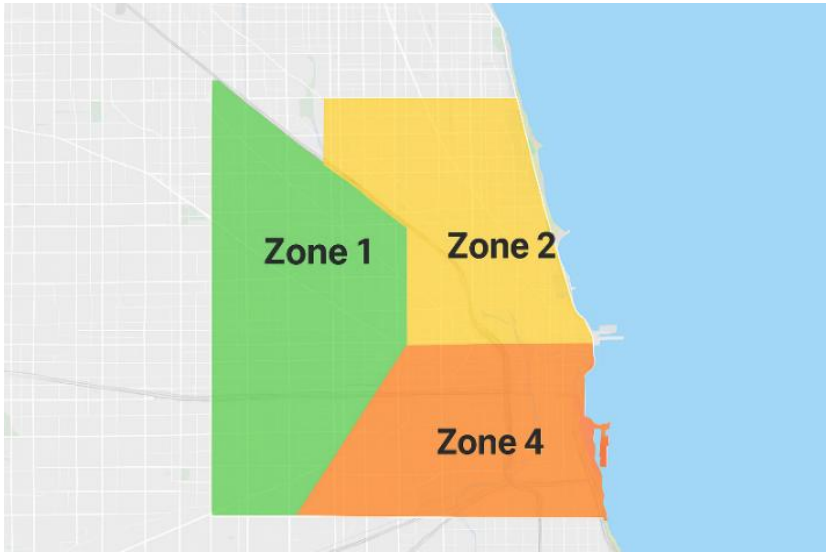

Рисунок 1.9 – Схема зонування на приклади США, штат Іллінойс

Математично умова невиникнення блокувань може бути сформульована наступним чином: для будь-якої пари об'єктів, скажімо, *a* та *b*, якщо пріоритет вивантаження *rank(a) < rank(b)* (тобто а вивантажується раніше), то його координата по осі довжини повинна задовольняти умову $x_{min}(a) \geq x_{max}(b)$. Це означає, що товари для ранніх зупинок завжди повинні мати більшу координату *X* (бути ближче до дверей), ніж товари для пізніх зупинок. Сучасні алгоритми часто використовують підхід «віртуальних стін». Його суть полягає в тому, що коли після завершення пакування всіх об'єктів для зупинки *N*, встановлюється уявна вертикальна площина, за яку не можуть заступати об'єкти наступної за пріоритетом зупинки *N-1* [18].

1.3. Оцінка технічних та фізичних обмежень

У теоретичних моделях контейнер часто уявляють як ідеальний прямокутний паралелепіпед. Але в реальному житті внутрішня геометрія транспортних засобів набагато складніша: там є різні конструктивні елементи та технологічні виступи. Ігнорування цих деталей може призвести до того, що план завантаження, який з точки зору геометрії виглядає правильним, на практиці просто не вдасться реалізувати.

Окремо виділяється атрибут крихкості (Fragility), який задається як логічна ознака та забороняє розміщення будь-яких інших об’єктів поверх відповідного вантажу. Дія цього атрибута не залежить від маси або габаритів елементів, що потенційно могли б бути розміщені зверху. У результаті планувальник змушений розташовувати такі об’єкти виключно у верхньому шарі завантаження. Це обмежує можливості щільного заповнення об’єму, зокрема при формуванні завантаження «під стелю», проте дозволяє уникнути пошкодження крихкого вантажу під час транспортування.

Окрім ваги, критичну роль відіграє геометрія взаємодії поверхонь. Для безпечного транспортування недостатньо, щоб центр маси верхньої коробки просто знаходився в межах периметра нижньої коробки. Необхідно гарантувати

достатню площу контакту (Support Ratio), яка зазвичай становить не менше 70–80% від площі основи верхнього об'єкта (рис. 1.10). Порушення цієї вимоги призводить до ефекту «звисання»). Цей ефект створює ризик перекидання при поворотах, або до деформації країв нижньої коробки, яка не розрахована на надто сильне точкове навантаження.

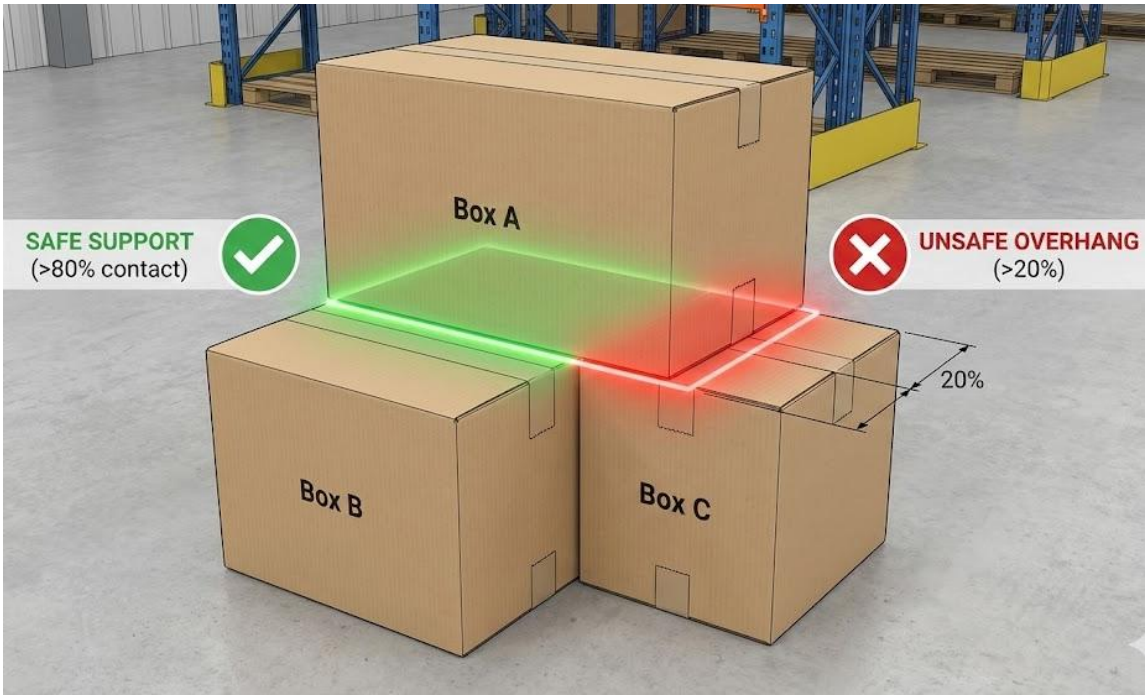

Рисунок 1.11 – Безпечна та небезпечна контактна площа при розміщенні пакунків один на одний

Для реалізації таких перевірок система повинна виконувати складні просторові запити типу «ray casting» (променеве сканування вниз), щоб точно визначити множину об'єктів, які слугують опорою для поточної коробки, і розподілити вагу між ними пропорційно площі перекриття або алгоритмічно обувити зберігання інформації про нижні пакунки [19]. Альтернативою цього є «запам’ятовування» кожною коробкою множини «сусідів» знизу. Проте, такий підхід сильно збільшує час виконання програми.

1.3.1 Визначення та моделювання заборонених зон (Forbidden Zones) та геометричних перешкод

Наукові логістичні джерела мають одну спільну поширену помилку. А саме те, що автори свідомо спрощують умови задачі, розглядаючи контейнер як простий прямокутний паралелепіпед. З математичної точки зору це зручно, але на практиці такий підхід майже одразу розбивається об реальність. Внутрішній простір вантажівки вкрай рідко буває "чистим". Зазвичай там вистачає нюансів, які заважають пакуванню: це і колісні арки, що виступають всередину, і кріплення, і агрегати холодильних установок. Ігнорувати їх - значить отримати план, який неможливо виконати фізично.

У нашому випадку, першим і найбільш масивним типом перешкод є кліматичне обладнання. У рефрижераторних напівпричепах (наприклад, серії Utility 3000R) холодильна установка, така як Carrier Transicold Vector (рис. 1.12), займає значний об'єм у верхній передній частині кузова. Окрім самого випарника, який виступає всередину на 300–400 мм, критично важливим є врахування повітряних каналів (air chutes) - гнучких рукавів, що проходять під стелею вздовж усього кузова. Алгоритм повинен резервувати простір висотою 100–150 мм під стелею як недоторканну зону для забезпечення циркуляції холодного повітря, інакше вантаж може зіпсуватися.

1.4 Постановка завдання на розробку

На основі проведеного аналізу існуючих підходів та виявлених недоліків класичних геометричних алгоритмів, сформовано комплексне завдання на розробку. Ключова ідея полягає у створенні спеціалізованого програмного модуля (бібліотеки Pack3D), який би заповнив нішу між дорогими корпоративними TMS-системами та примітивними "калькуляторами об'єму". Головним викликом є необхідність об'єднати в одному рішенні високу обчислювальну ефективність, характерну для евристичних методів, із точністю фізичного моделювання, яка зазвичай притаманна інженерним CAD-системам.

Функціональні пріоритети. Розробка має фокусуватися на вирішенні трьох критичних проблем, які ігноруються більшістю open-source рішень:

- дотримання логістики «останньої милі»: алгоритм зобов'язаний гарантувати суворе виконання принципу LIFO для мульти-дроп маршрутів, унеможливлюючи ситуації блокування вантажу;
- ваговий контроль у реальному часі: система повинна не просто рахувати загальну масу, а динамічно відстежувати навантаження на кожну вісь тягача та напівпричепа (відповідно до формули мостових ваг), відбраковуючи варіанти пакування, що ведуть до штрафів;
- адаптивність до геометрії: програмне ядро має підтримувати роботу з «брудним» простором кузова, коректно обробляючи зони виключення (колісні арки, агрегати) через механізми Constructive Solid Geometry.

Технічні вимоги до реалізації. З огляду на необхідність інтеграції майбутньої бібліотеки у високочестотні системи обробки замовлень, встановлено жорсткі вимоги до архітектури. Рішення повинно бути реалізоване на платформі .NET (Standard 2.1 або вище) із використанням структур даних, що мінімізують навантаження на Garbage Collector. Критичним показником є швидкодія: час розрахунку типового сценарію (до 1000 одиниць вантажу) не повинен перевищувати 100-150 мс, що дозволить використовувати алгоритм у режимі "на льоту". Окрім самого двигуна пакування, завдання включає розробку інструментарію для діагностики та візуалізації результатів, який би перетворював абстрактні координати у зрозумілі для людини схеми завантаження.

Рисунок 1.12 – Холодильна установка Carrier Transicold Vector серії Utility 3000R

Другий тип перешкод пов’язаний з особливостями ходової частини та конструкції кузова транспортного засобу. У спеціалізованих напівпричепах типу Drop Deck, а також у вантажівках, що використовуються для міської логістики (зокрема на базі шасі Ford F-650), колісні арки часто частково заходять у вантажний відсік. У результаті корисна ширина підлоги в зоні задніх осей зменшується, що необхідно враховувати під час формування плану завантаження.

До цієї ж групи належать елементи заднього порталу, зокрема внутрішні петлі розпашних дверей або напрямні ролетних воріт (Roll-up doors), які широко застосовуються у вантажному транспорті США, наприклад виробництва Whiting Door. Наявність ролетного механізму зменшує ефективну висоту завантаження біля заднього краю кузова в середньому на 2–3 дюйми, що на практиці часто призводить до колізій під час спроб розмістити високі палети максимально близько до порталу [20].

Висновки до розділу 1

У першому розділі розглядається проблема автоматизованого розміщення вантажів у транспортних засобах із позиції практичного застосування в сучасній логістиці. Аналіз показав, що задача тривимірного пакування контейнерів (3D-CLP) належить до класу NP-важких, що суттєво обмежує можливість використання

точних алгоритмів у задачах промислового масштабу. За великих обсягів вхідних даних такі підходи не забезпечують прийнятного часу обчислень і тому не можуть бути використані для побудови планів завантаження в режимі, близькому до реального часу.

Оскільки вирахувати все математично точно в цій задачі - надто комплексна задача через складність, на практиці потрібно спиратися на евристику. Порівняльний аналіз показав, що для «строкатого» (гетерогенного) вантажу найкраще підходять конструктивні методи - ті, що будують рішення цеглинка за цеглинкою.

Крім того, у першому розділі вдалося формалізувати ті обмеження, на які класичні геометричні моделі зазвичай закривають очі. Зокрема, запропоновано спосіб контролювати навантаження на осі через розрахунок моментів сил ще під час планування, а не коли машина вже виїжджає на ваги. А щоб дотримуватися черговості вивантаження (LIFO), застосували механізм динамічних «віртуальних стін» - це дозволяє уникнути зайвого перекладання товару. Також врахували специфіку самих вантажів: їхню крихкість та здатність витримувати вагу інших коробок зверху.

2 МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ ТА РОЗРОБКА БАГАТОКРИТЕРІАЛЬНОГО АЛГОРИТМУ

У даному розділі розглядається ключовий етап роботи, що полягає у формалізації логістичної задачі засобами математичного апарату. Метою розділу є перетворення узагальненої прикладної проблеми розміщення вантажу у формалізовану математичну модель, придатну для подальшого алгоритмічного опрацювання. Важливим аспектом побудови моделі є забезпечення її сумісності з реалізацією у вигляді програмного коду мовою C#.

2.1. Розробка математичної моделі системи та її цільової функції

2.1.1 Моделювання вантажу, транспортного контейнера та їхніх геометричних властивостей

Введемо систему координат, де початок *(0,0,0)* знаходиться у лівому нижньому куті передньої стінки трейлера, тобто біля кабіни водія. Таким чином геометричні орієнтири – це:

- вісь X - поздовжня вісь (довжина кузова);
- вісь Y - вертикальна вісь (висота кузова);
- вісь Z - поперечна вісь (ширина).

Для початку, нехай $I = \{1, 2, \ldots, n\}$ – це теоретична множина пакунків. Кожен елемент характеризується наступним вектором з властивостей:

$$b_I = \{l_I, w_I, h_I, m_I, R_I, S_I, F_I, Stop_I\}, \quad (2.1)$$

де:

- l_I, w_I, h_I - габаритні розміри (довжина, ширина, висота);
- m_I - маса;
- $R_I \in \{0 \ldots 5\}$ - допустимі орієнтації (обертання);
- S_I - ліміт навантаження зверху (кг);

- $F_I \in \{0 \dots 1\}$ - маркер крихкості (1 - крихке, а 0 - некрихке);
- $Stop_I$ - індекс черги вивантаження. Є спрощеним числовим маркером.

Контейнер описується розмірами $L * W * H$ та множиною заборонених підпросторів $F_Z = \{f_{z1} \dots f_{zk}\}$, де кожен f_{zj} визначається координатами $(x_{min}, y_{min}, z_{min})$ та $(x_{max}, y_{max}, z_{max})$. Ефективний простір для пакування V_{eff} [21] визначається як:

$$V_{eff} = \frac{|0, L| * [0, H]}{\cup f_{zj}}. \quad (2.2)$$

В сухому підсумку, для кожного вантажу *i* необхідно визначити:

- координати розміщення: (x_i, y_i, z_i);
- орієнтацію: r_i (вибір перестановки вимірів).

2.1.2 Формування багатокритеріальної скорингової функції для оцінки розміщення (щільності, балансу та стабільності)

Досліджувана задача пакування зводиться до пошуку допустимого розміщення, що максимізує цільову функцію. У розробленій системі використано функцію оцінки позиції [22], яка обчислюється для кожної одиниці на розміщення у кожному доступному вільному просторі:

$$s_{core\{box,space\}} = s_{Geom} + s_{Stab} + s_{Phys}. \quad (2.3)$$

1. Геометрична щільність (S_{Geom})**.** Цей компонент мінімізує втрату простору при виборі конкретного вільного місця:

$$s_{Geom} = -\alpha, \quad (2.4)$$

де α - ваговий коефіцієнт. Чим менша різниця між розміром коробки та розміром "дірки", тим вища оцінка. Тим самим це мінімізує штраф.

2. Стабільність (S_{Stab}). Цей показник вкрай важливий, бо він максимізує розміщення вантажів ближче та передньої стінки траку:

$$s_{Stab} = -\beta * \gamma_{pos} . \quad (2.5)$$

Це гарантує, що алгоритм заповнюватиме простір знизу вгору, створюючи надійну базу.

3. Фізичний баланс (S_{Phys}). Це динамічний компонент, що залежить від поточного стану завантаження трака:

$$s_{Phyc} = -\gamma(State) * |X_{center_{of_{box}}} - |X_{ideal_{balance}}| , \quad (2.6)$$

де $\gamma(State)$ - це не просто константа, а окрема функція від поточного навантаження на осі :

$$X_{ideal_{balance}} = \frac{Pos_{Kingrin}+Pos_{Tandem}}{2} . \quad (2.7)$$

Далі обчислення опрацьовує коефіцієнт, що зростає за експонентною характеристикою:

$$\gamma = 0.1 + 100 * (max(Ratio_{street}, Ratio_{drive}, Ratio_{tandem}))^5 , \quad (2.8)$$

такий п’ятий ступінь залежності означає, що доки осі недовантажені, алгоритм майже не звертає уваги на баланс і віддає перевагу щільності пакування. Але коли навантаження підходить до максимуму, значення балансу різко зростає - у сотні разів - і алгоритм починає поступово «жертвувати» щільністю, щоб розташувати вантаж у місцях, які допомагають вирівняти розподіл мас.

2.2. Розробка евристичного алгоритму розміщення з контролем геометрії

2.2.1 Впровадження методу Guillotine Split для точного та неперетинаючого розрізання вільного простору

Для кожного вантажного елементу система шукає найоптимальніше місце розташування. Головною інновацією реалізації став підхід до управління простором. Замість складних дерев розбиття, які зазвичай застосовують у складній бізнес-логіці, використовується концепція «зворотної гільйотини» для виділення вільного об’єму.

Коли коробка поміщається *B* у простір *S*, останній видаляється зі списку. На його місце додаються нові простори, утворені різницею *S* - *B*. Гільйотинний розріз виконується послідовно по осях за наступним шаблоном:

- коли виявиться, що коробка коротша за площу - утвориться залишок праворуч від неї;
- якщо коробка вужча за простір, утворюється залишок "перед коробкою". Важливо, що цей залишок формується лише в межах довжини коробки (щоб не перетинатися з R_{right}.);
- якщо коробка нижча за простір, утворюється залишок "над коробкою". Йому присвоюється *LayerIndex + 1*.

Ця стратегія гарантує, що всі вільні простори у списку є диз'юнктивними, тобто не перетинаються. Це критично спрощує перевірку, оскільки щоб знайти місце, достатньо знайти один простір, який може вмістити коробку. І при цьому немає потреби перевіряти інші.

2.2.2 Реалізація механізму "Віртуальних Стін" для забезпечення жорсткого маршрутного критерію (LIFO)

У стандартному BPP (Bin Packing Problem) положення по X не є обмежене. У нашій задачі воно суворо регламентоване індексом зупинки. Реалізація цього обмеження виконана через фільтрацію просторів (Space Filtering) [23]. Нехай $L_{trailer}$ - довжина трейлера, N_{stops} - кількість зупинок. Довжина сектора для однієї зупинки:

$L_{sec} = L_{trailer} / N_{stops}$. Для вантажу з індексом i (де 1 - перша зупинка, N - остання) визначається "заборонена зона" спереду:

$$x_{\text{minAllowed}} = \left(N_{stops-1} - i\right) * L_{sec} . \quad (2.9)$$

При переборі просторів алгоритм ігнорує будь-який простір, якщо початок його є $X < X_{minAllowed}$. Це створює, так званий, ефект "віртуальної стіни", яка фізично не існує, але яку алгоритм не може перетнути ні за яких умов.

Для вантажів останньої зупинки $(i=N)$, $X_{min} = 0$, тобто їм доступний весь трейлер (але вони завантажуються першими завдяки сортуванню, тому займають глибину). Для першої зупинки $(i=1)$, X_{min} буде близько до кінця трака, змушуючи вантажити їх біля дверей. Це забезпечує збереження принципу LIFO.

2.3. Впровадження фізичних та безпекових обмежень у скоринг

2.3.1 Алгоритм розрахунку навантаження на осі (Steer, Drive, Tandem) при розміщенні одиниці вантажу

Фізична модель реалізована у методі CalculateAxleWeights. Вона базується на розрахунку моментів сил [23] за наступними вхідними даними:

- geo - геометрія осей (відстані від бампера до Kingpin, до Drive, до Tandem);
- payload - маса вантажу;
- moment - сумарний момент вантажу відносно передньої стінки.

Розрахунок навантаження на осі відбувається в декілька взаємопов’язаних кроків:

- спочатку, визначається центр ваги вантажу (CoG):

$$X_{cog} = \frac{Moment}{Payload} . \quad (2.10)$$

- на наступному етапі застосовується закон розподілу навантажень у визначеній системі. Його можна пояснити як: визначення реакції опори на

задній візок напівпричепа базується на моментному рівнянні. Важливо розуміти, що тут ключовим параметром виступає плече сили. Розрахувати плече сили можна за формулою :

$$F_{tandem} = Payload * \frac{X_{cog} - D_{kingpin}}{D_{tandem} - D_{kingpin}}. \quad (2.11)$$

- кожна наступна і наступна ітерація передбачає виокремлення величини навантаження. Це навантаження припадає безпосередньо на вузол зчеплення:

$$F_{kingnip} = Payload - F_{tandem}. \quad (2.12)$$

- насамкінець, навантаження передається на сам трак. Це можна описати наступними формулами:

$$F_{streep\_load} = F_{kingnip} * \frac{D_{drive} - D_{kingpin}}{WheelBase_{tractor}}, \quad (2.13)$$

$$F_{drive\_load} = F_{kingnip} - F_{streep_{load}}. \quad (2.14)$$

Оскільки наведений каскад розрахунків виконується у внутрішньому циклі перебору для кожної потенційної позиції пакування, вимоги до швидкодії є критичними. Для нівелювання затримок у програмній реалізації відмовлено від високорівневих абстракцій на користь примітивних типів даних та прямого управління пам’яттю (зокрема, через механізм Span<T> у середовищі .NET). Такий архітектурний підхід забезпечує виконання тисяч ітерацій перерахунку фізики за секунду без суттєвого впливу на загальний час роботи планувальника.

2.3.2 Впровадження штрафу за недовантаження Tandem для забезпечення безпекового балансу

Як зазначалося в аналізі, недовантаження осей є небезпечним. Хоча у моделі не буде реалізовано явної перевірки *Minimum Weight*, механізм physicsFactor працює двосторонньо. Цільова функція *distToTarget* намагається мінімізувати відстань від центру ваги коробки до геометричного центру між осями X_{target}.

Це автоматично "стягує" вантаж до центру трейлера. Зокрема, таким чином унеможливлюються ситуації, коли весь вантаж лежить або на «носі», або на «хвості». Таким чином, функція й зберігає баланс.

2.3.3 Логіка перевірки крихкості та лімітів штабелювання (IsStackingValid)

Це одна з найскладніших частин алгоритму, яка відповідає за вертикальну взаємодію. Метод *IsStackingValid* реалізує перевірку "знизу-вгору". Коли коробка займає положення висоти $Y > 0$, алгоритм повинен знайти, на що вона спирається. Для цього виконуються всього дві дії:

1. Пошук опор, коли сканується список вже розміщених коробок (placements). Кандидатом на опору є коробка B_{bottom}, якщо:
- $|\ (B_{bottom}.Y + B_{bottom}.Height) - B_{top}.Y\ | < \mathcal{E}$ (верхня грань нижньої збігається з нижньою гранню верхньої);
- $Intersect2D(B_{bottom}, B_{top}) = true$ (є перетин у плані X-Z).
2. Перевірка обмежень:
- якщо $B_{bottom}.Fragile = true$, то розміщення заборонено;
- якщо $B_{top}.Weight > B_{bottom}.MaxStackOnTopKg$, то розміщення заборонено.

Цей механізм гарантує, що на схемі завантаження важкі палети ніколи не опиняться на крихких коробках з електронікою.

Висновки до розділу 2

У другому розділі зібрано математичну модель, яка дозволяє дивитися на задачу 3D-пакування не лише як на геометричну головоломку, а як на задачу з кількома конфліктними цілями. Для вантажу й контейнера введено набір параметрів, що описують розміри, масу, крихкість та можливість обертання. Цих характеристик виявилося достатньо, щоб відсіяти більшість нереалістичних варіантів розміщення ще на початковому етапі.

Оцінка рішень виконується через скорингову функцію. Вона поєднує щільність пакування, вертикальну стабільність і баланс центру мас. Важливо зауважити, що ця оцінка не є фіксованою, тобто під час роботи алгоритму пріоритети змінюються. Якщо варіант виглядає ненадійно з точки зору стійкості, він швидко втрачає бали й фактично випадає з розгляду.

Окремий акцент зроблено на підсистемі візуалізації та діагностики. Адже голі цифри мало про що говорять користувачеві. Моя задача полягала в тому, щоб створити інструмент, який інтерпретує складні математичні розрахунки і перетворює їх на «людський», інтуїтивно зрозумілий формат, з яким легко працювати.

Фізичні обмеження не винесені в окрему перевірку, а вбудовані прямо в ядро розрахунків. Зокрема, реалізовано оцінку осьових навантажень і перевірку допустимості штабелювання з урахуванням перекриття, ваги та крихких позицій. У результаті алгоритм формує плани завантаження, які виглядають не ідеальними на папері, зате нормально працюють у реальних умовах перевезення.

3 СИСТЕМНА ІНТЕГРАЦІЯ ТА ІНСТРУМЕНТИ ДІАГНОСТИКИ

На цьому етапі магістерської роботи, відбувається перехід від теоретичних математичних моделей до конкретних, здатних функціонувати в умовах реального виробничого середовища, інженерних рішень, Третій розділ магістерської роботи присвячено детальній технічній реалізації програми Pack3D, яка втілює розроблені у попередніх розділах алгоритми просторової оптимізації та фізичного моделювання навантажень.

Найбільшою проблемою при реалізації систем класу 3D-BPP є мінімізація NP-важкої природи задачі, що вимагає значних обчислювальних потужностей, та необхідність отримання результату в режимі реального часу або хоча б в soft real-time для інтеграції у складські бізнес-процеси.

3.1. Обґрунтування вибору апаратно-програмних засобів та архітектури

Фундаментом для побудови високопродуктивної системи є правильний вибір середовища виконання та інструментарію розробки. Враховуючи вимоги до швидкодії, кросплатформності та надійності, для реалізації бібліотеки Pack3D було обрано екосистему Microsoft.NET. Далі виконується детальна аргументація, стосовно цього вибору.

3.1.1 Обґрунтування вибору .NET Core (Standard 2.1) та мови C# для розробки Core Packing Library

Для реалізації проєкту я обрав саме .NET Standard 2.1 та мову C# відповідно. Цей вибір – є результатом холодного розрахунку та технічних вимог. Серед значущих переваг NET:

1. Ефективне управління пам'яттю та оптимізація роботи Garbage Collector (GC) `Головна причина криється в особливостях управління пам'яттю. Справа в тому, що евристичні алгоритми 3D-пакування - це справжній

стрес-тест для системи. Поки програма шукає оптимальне місце для однієї-єдиної коробки, вона створює і тут же відкидає тисячі тимчасових об'єктів (координати, вектори, варіанти розміщення). У пам'яті накопичується величезна кількість «короткоживучих» екземплярів. Якби середовище виконання (Runtime) не вміло з цим ефективно працювати, Garbage Collector (збирач сміття) постійно б зупиняв програму, щоб розчистити місце, роблячи її непридатною для реального використання. Вибір C# дозволив використати специфічні механізми оптимізації пам'яті [25]. Серед них:

- у .NET об’єкти обробляються за моделлю трьох поколінь (Gen 0, 1, 2). Більшість тимчасових об’єктів алгоритму пакування живе лише одну ітерацію циклу, тому потрапляє у Generation 0. Збірка сміття в цьому поколінні проходить дуже швидко і зазвичай не блокує основні потоки, що забезпечує високий рівень продуктивності системи..NET використовує модель трьох поколінь (Gen 0, 1, 2). Оскільки більшість об'єктів в алгоритмі пакування є "ефемерними" (живуть лише одну ітерацію циклу), вони потрапляють у Generation 0. Збірка сміття у цьому поколінні є надзвичайно швидкою. Зазвичай, очищення відбувається навіть без блокування основних потоків, що забезпечує високу пропускну здатність системи;
- хоча основна логіка програми використовує масиви, підтримка типів Span<T> у. NET Standard 2.1 відкриває можливості для подальшої оптимізації через "slicing" масивів без алокацій. Це є перспективним для обробки великих масивів даних про вантажі.

2. Незмінність даних (Immutability) та Record Types. Як правило, проектування складних алгоритмів тяжіє до використання незмінних структур даних. Це, у свою чергу, підвищує передбачуваність коду та спрощує відлагодження. Впровадження типів Records (записів) у C# дозволило реалізувати концепцію "immutability by default" [26]. Основні

сутності системи, такі як Box та Trailer, оголошені як record, що надає ряд переваг:

- потокобезпечність. Оскільки стан об'єкта неможливо змінити після створення, декілька потоків можуть безпечно читати дані одного й того ж об'єкта Box без необхідності використання блокування виконання одним потоком. Це позитивно впливає на продуктивність;
- порівняння за значенням. Автоматично згенеровані методи порівняння дозволяють легко перевіряти, чи є дві конфігурації пакування однаковими. Це сильно спрощує логіку кешування та тестування;
- недеструктивна мутація. Використання оператора with дозволяє створювати нові версії об'єктів. При цьому, основа об’єкта в пам’яті зберігається, змінюються лише окремі поля. Це активно використовується при моделюванні теоретичним сценарій.

3. Кросплатформність та уніфікація API через .NET Standard 2.1. Вибір платформи .NET є стратегічним рішенням при проєктуванні системи, оскільки вона забезпечує високий рівень кросплатформності та довготривалу підтримку програмного продукту. Використання стандарту .NET Standard 2.1 дозволяє уніфікувати програмний інтерфейс бібліотеки Pack3D.Core та гарантує наявність єдиного набору API, доступного в усіх сучасних реалізаціях платформи .NET [27]. Застосування цього стандарту забезпечує сумісність бібліотеки з такими середовищами виконання, як .NET Core 3.x, .NET 5/6/7/8, а також із кросплатформними фреймворками Xamarin та ігровим рушієм Unity3D (див. табл. 3.1). Це дає змогу використовувати розроблене ядро алгоритмів без модифікації коду як у серверних та настільних застосунках, так і в мобільних або інтерактивних візуалізаційних системах;

Таблиця 3.1 - переваги обраного підходу для різних сценаріїв розгортання

Сценарій використання	Платформа	Перевага .NET Standard 2.1
Хмарний сервіс (Backend)	Linux (Docker/Kubernetes)	Висока продуктивність на Linux-серверах, легка контейнеризація, відсутність прив'язки до Windows API.
Мобільний термінал	Android / iOS (Xamarin)	Можливість запуску алгоритму безпосередньо на планшеті комірника для перерахунку плану в офлайн-режимі.
Десктоп клієнт	Windows (WPF/WinForms)	Інтеграція з існуючими корпоративними системами на базі Windows.
3D Візуалізація	Unity 3D	Можливість використання того ж коду розрахунку фізики всередині ігрового рушія для реалістичної візуалізації.

4. Використання LINQ та функціональних патернів. Для обробки колекцій даних багаторазово було використано технологію LINQ (Language Integrated Query). Незважаючи на потенційні накладні витрати при неправильному використанні, LINQ забезпечує декларативний стиль коду, подібний до SQL, що є критичним для реалізації складної бізнес-логіки сортування та фільтрації. До прикладу, у Solver.cs є попередня обробка черги завантаження. Вона реалізована через ланцюжок викликів OrderBy / ThenBy, що дозволяє чітко визначити евристичні пріоритети (спочатку LIFO, потім пріоритет клієнта, потім щільність). Такий функціональний підхід також використано при

оновленні станів колекцій. Це дозволяє уникнути побічних ефектів (side effects), що характерно для імперативного стилю [28].

3.1.2 Детальний опис структур даних та логічних зв'язків між сутностями

Архітектура Pack3D тримається на чіткому розділенні: дані окремо, логіка окремо. Це стандарт для сучасних розподілених систем. Але головне, що в основі закладена незмінність (Immutability). Припустимо, що в основі проблеми - задача комбінаторної оптимізації, де простір варіантів вимірюється мільйонами. Якщо в процесі розрахунку якийсь об'єкт випадково "мутує" (змінить свої координати чи стан), весь алгоритм видасть помилку. Тому гарантія того, що створений об'єкт залишається сталим назавжди - це критична вимога, особливо якщо ми хочемо без страху запускати обчислення в кілька потоків.

Для реалізації цього в .NET Standard 2.1 я зробив ставку на типи record. Вони ідеально підходять, бо забезпечують порівняння за значенням (value-based equality) прямо з коробки. Це дозволяє працювати з великими структурами даних так само зручно, як з простими числами. На відміну від звичайних класів, де треба вручну писати купу коду для порівняння, record сам генерує коректні методи хешування. Це в рази спрощує задачу, коли, скажімо, треба реалізувати кешування або відсіювати накопичені діблукати.

Центральною сутністю доменної області є Box (див. табл. 3.2). У розробленій системі це не абстрактний геометричний примітив, а насичена інформаційна модель. Структура Box інкапсулює в собі всі фізичні та логістичні обмеження того чи іншого вантажу. Проєктування цього класу вимагає першочергового врахування гетерогенності реальних вантажопотоків, в яких кожен об'єкт може мати унікальні правила обробки.

Таблиця 3.2. Детальна специфікація та семантика полів класу Box

Поле	Тип даних	Семантика та вплив на алгоритм
Id	string	Унікалізований ідентифікатор (наприклад, GUID чи штрихкод). Використовується в якості первинного ключа для зв'язку між вхідними масивами та результатами розміщення (Placement). Критичний для інтеграції з WMS/ERP системами.
Sku	string	Артикул товару (Stock Keeping Unit). Використовується для агрегації статистики та групування у звітах. Дозволяє аналізувати ефективність пакування за категоріями товарів.
SizeMm	Dimensions	Композитний об'єкт, що містить виміри (L, W, H).
WeightKg	double	Є вхідним параметром для розрахунку тензора інерції та моментів сил при перевірці осьових навантажень.
AllowedRotations	RotationMask	Бітова маска . Вона визначає допустимі ступені свободи обертання.
Stackability	double	Коефіцієнт штабелювання (0.0–1.0). Визначає, яку частку площі поверхні можна використовувати як опору. Значення < 1.0 моделюють нерівні поверхні або пірамідальні вантажі.
MaxStackOnTopKg	double	Це поле рекурсивно перевіряє сумарну вагу стовпця над коробкою і відхиляє розміщення при найменшому перевищенні ліміту.
Fragile	bool	Якщо true, об'єкт отримує найвищий пріоритет на розміщення у верхньому шарі ("Top-most")
DeliveryStopIndex	int	Це ключовий параметр для забезпечення LIFO-сумісності. Менший індекс означає пізнішу вигрузку
Priority	int	Евристичний пріоритет. Використовується для вирішення конфліктів при дефіциті простору. Вантажі з вищим пріоритетом розглядаються алгоритмом раніше в межах однієї групи DeliveryStopIndex.

Сутність Box є незалежною сутністю. Вона не містить посилань на інші об'єкти, що дозволяє легко серіалізувати її та передавати між мікросервісами. Зв'язок з результатом встановлюється виключно через поле Id у об'єкті Placement.

Клас Trailer (див. табл. 3.3) інкапсулює геометричні та фізичні параметри простору пакування. На відміну від спрощених підходів, в Pack3D реалізовано детальну модель шасі та внутрішніх перешкод.

Таблиця 3.3. Специфікація та фізична модель класу Trailer

Поле	Тип даних	Опис фізичної моделі
InnerMm	Dimensions	Визначає глобальну систему координат, де (0,0,0) - лівий нижній кут біля передньої стінки.
Axles	AxleLimits	Об'єкт-контейнер для нормативних лімітів навантаження на групи. Ці параметри є динамічними, тобто вони можуть і мають змінюватися залежно від законодавства штату/країни транзиту. В магістерській роботі, цільовою країною є США.
AxleGeo	AxlePositions	Відстань від бампера до шворня, положення сідельно-зчіпного пристрою та заднього візка, тобто це складова для точної розрахунку розподілу реакцій опор.
FloorStrengthPsf	double	Допустиме навантаження для підлоги Може бути використане для перевірки локальних перевантажень від важких вантажів.
ForbiddenZones	List<Rect3>	Зони, в яких заборонено розміщувати будь-які об’єкти

Окрім геометричного розміщення коробок у вантажному просторі, врахування параметра AxlePositions додає фізичний зміст у процес оптимізації. Завдяки цьому Solver може коректно визначати положення центру мас вантажу та проєктувати навантаження на осі тягача й напівпричепа. У результаті класична

задача пакування виходить за межі абстрактної геометрії та розглядається як повноцінна інженерна задача статики твердого тіла. Це дозволяє оцінювати балансування, допустимі осьові навантаження та експлуатаційну безпеку перевезення.

Об'єкт Placement (див. табл. 3.4) фіксує конкретне рішення алгоритму щодо окремої коробки. Він є зв'язуючою ланкою між вхідними даними (Box, Trailer) та візуалізацією програми.

Таблиця 3.4 Структура результату Placement

Поле	Тип даних	Призначення
BoxId	string	Зовнішній ключ до таблиці/списку вхідних коробок.
PositionMm	Vector3	Точні координати кута коробки. Використання double забезпечує субміліметрову точність, необхідну для щільного пакування без помилок округлення.
Rotation	(int, int, int)	Кути Ейлера (дискретні: 0, 90, 180, 270). Визначають орієнтацію локальної системи координат коробки відносно кузова.
LayerIndex	int	Індекс шару пакування. В коді, він використовується для групування об'єктів по висоті.
AxleContribution	AxleWeights	Зберігання цього значення дозволяє реалізувати інкрементальний перерахунок ваги при ручному редагуванні плану (видалення коробки просто віднімає її внесок, без повного перерахунку).

Архітектура системи пакування не будувалась як “ідеальна UML-картинка”. Вона скоріше виросла навколо того, що реально потрібно зберегти після роботи алгоритму. У центрі всього - об’єкт SolveResult. Це просто контейнер стану. Тут зберігається вся інформація про роботу алгоритму: як розташовані коробки, що не помістилося, які метрики отримані на виході та які обсяги вільного простору залишилися в кузові.

Продуктивність Pack3D з’явилась не тому, що алгоритм “розумний”. Основна вигода - в дрібних, місцями нудних рішеннях по пам’яті. Дуже швидко стало очевидно, що класичний об’єктно-орієнтований стиль у геометрії - це шлях у GC-пекло. Купа тимчасових об’єктів, вектори, координати, прямокутники - все це створюється і вмирає тисячі разів. Тому базові геометричні типи на кшталт Vector3, Dimensions і Rect3 були зроблені як record struct. Без цього алгоритм просто захлинався на великій кількості перевірок.

Структури дали стекову алокацію, а разом із нею - мінус зайві проходи Garbage Collector’a. Для задачі, де за секунду можуть виконуватись мільйони перевірок перетинів, це вирішує більше, ніж будь-яка мікрооптимізація в логіці. Додатково в тих місцях, де доводиться ганяти масиви даних, використовується Span<T>. Це дозволяє працювати з пам’яттю напряму, без копіювання, і не плодити зайві буфери. Менше руху даних - менше затримок. Все просто.

Ще один момент, який з’явився не з теорії, а з практики - імутабельність. Об’єкти типу Box, Trailer і Placement після створення не змінюються. Взагалі. Завдяки цьому кілька потоків можуть одночасно дивитися на один і той самий стан пакування, рахувати альтернативи або метрики й не заважати один одному.

3.1.3 Структурні патерни проєктування програмного забезпечення

Центральним елементом архітектури Pack3D є патерн «Фасад». Він забезпечує взаємодію між клієнтом та складною підсистемою алгоритмів Згідно з класифікацією GoF, цей структурний патерн надає уніфікований інтерфейс до набору інтерфейсів у підсистемі. Це спрощує її використання.

У системі Pack3D роль фасаду виконує статичний клас Pack3D.Solver. Він є єдиною точкою входу для всіх операцій розрахунку.

Архітектурні обов'язки Фасаду Solver:

- оркестрація конвеєра. Метод Solve не реалізує алгоритм пакування монолітно. Замість цього, він координує виконання декількох послідовних етапів. До цих етапів можна віднести препроцесинг даних, виконання евристичного розміщення, розрахунок фізики та пост-аналіз;
- ізоляція підсистем. Solver інкапсулює від користувача всі внутрішні допоміжні класи. До прикладу, алгоритм оперує AxleCalculator для розрахунку навантаження на осі, SpaceManager для керування вільним простором чи StabilityChecker для перевірки стійкості, поки клієнт працює тільки з високорівневими поняттями;
- керування конфігурацією. Фасад приймає об'єкт SolveOptions, транслюючи налаштування користувача (наприклад, стратегію балансування або ліміт часу) у відповідні параметри внутрішніх алгоритмів.

Ця структура дозволяє змінювати внутрішню логіку (наприклад, оптимізувати метод InitializeSpaces для врахування колісних арок). При цьому, користувач не отримує порушення бінарної сумісності з клієнтськими додатками, оскільки сигнатура методу Solve залишається незмінною.

Без використання Фасаду клієнту довелося б самостійно інстанціювати класи алгоритмів, керувати списками вільних просторів та вручну викликати методи перевірки перетинів. В теорії, це призвело б до сильного зв'язування клієнтського коду з деталями реалізації бібліотеки, що ускладнило б подальший супровід та оновлення системи.

Для формального опису статичної структури системи розроблено діаграму класів UML (рис. 3.1). Вона демонструє взаємозв'язки між основними компонентами ядра, допоміжними сервісами та структурами даних.

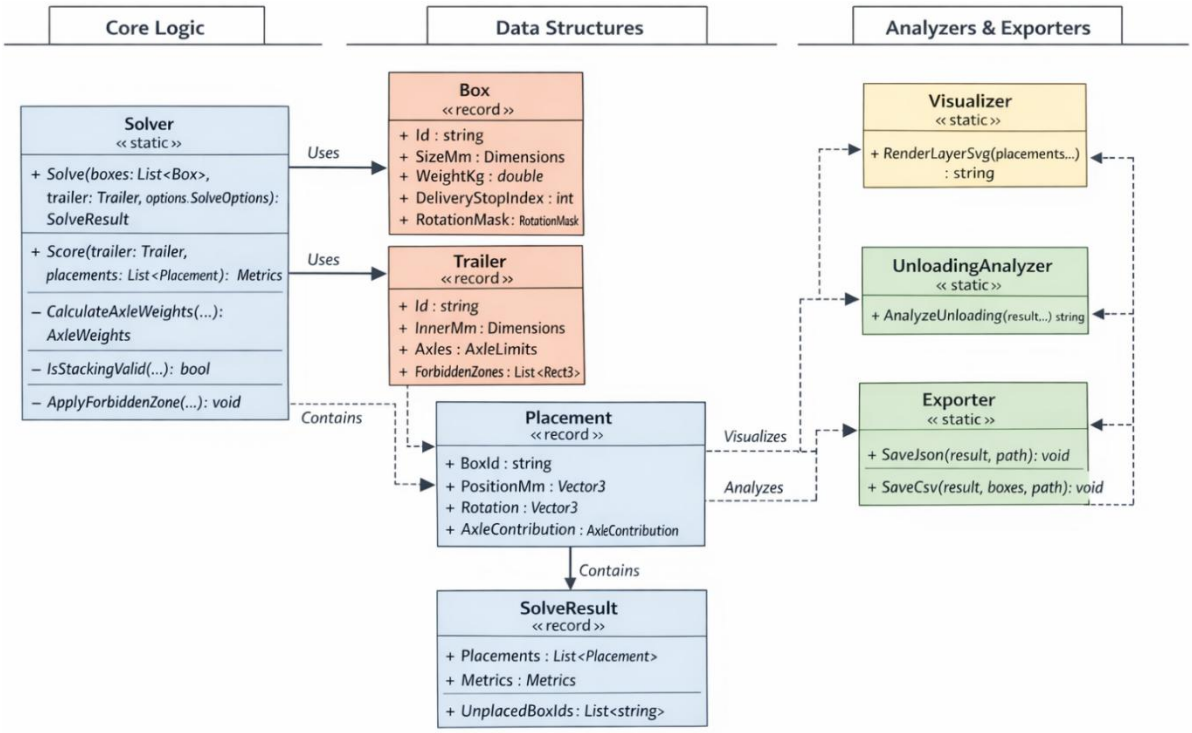

Рисунок 3.1 – UML діаграма розроблюваного програмного забезпечення

Статична структура класів системи набуває більш практичного сенсу під час виконання програми, коли між ними починають циркулювати дані. Для опису цього процесу в роботі використано архітектурний патерн «Конвеєр». У його межах вхідні дані послідовно проходять через набір етапів перетворення (рис. 3.2). Такий підхід дозволяє чітко організовувати алгоритм, ізолюючи при цьому відповідальності окремих фаз. Зокрема, це дозволяє підвищити керованість складного евристичного процесу пакування.

Рисунок 3.2 – Основний UseCase розроблюваного програмного забезпечення

Виконання запиту Solve не стартує одразу з магії алгоритмів. Спочатку йде підготовка. Клієнт викликає Solver.Solve(), а система банально перевіряє, чи вхідні дані взагалі мають сенс. Після цього список коробок сортується. Саме цей крок сильно впливає на фінальний результат, хоча на перший погляд виглядає другорядним. Порядок обробки коробок визначається не одним параметром, а

одразу кількома: індексом зупинки доставки (щоб не ламати LIFO), пріоритетом, щільністю та об’ємом.

На цьому ж етапі з простору “вирізаються” всі перешкоди - колісні арки, технічні ніші, конструктивні елементи кузова. Для цього використовується окремий механізм, який по суті зводиться до булевих операцій над об’ємами. Рішення не найпростіше, зате дозволяє коректно працювати з реальною, а не ідеалізованою геометрією контейнера.

Основна робота починається в циклі розміщення. Коробки обробляються по черзі, вже у відсортованому порядку. Для кожної з них система перебирає всі допустимі орієнтації, а потім намагається “приміряти” коробку до кожного доступного вільного простору. Кожен варіант оцінюється: дивляться на щільність заповнення, стійкість конструкції та вплив на баланс навантаження по осях. Паралельно виконуються перевірки сумісності зі вже розміщеним вантажем. Зокрема, через IsStackingValid відсіюються варіанти, де коробка перевантажує нижні рівні або тисне на крихкі позиції. Більшість комбінацій відпадає саме тут.

Коли підходящий варіант знайдено, система його фіксує. Створюється об’єкт Placement, після чого зайнятий об’єм розрізається на нові вільні зони за алгоритмом Guillotine Split. Простір змінюється, і це важливо - всі наступні коробки вже працюють з оновленою картиною.

Окремо від самого пакування існує перевірка коректності розвантаження. Вона винесена в компонент UnloadingAnalyzer і працює вже з готовим SolveResult. Розміщення групуються за індексами зупинок, після чого імітується процес вилучення вантажу. На цьому кроці шукаються так звані блокери - коробки, які фізично заважають доступу до потрібного вантажу, хоча за логікою маршруту їх ще не мали б чіпати. Перевірка зводиться до геометрії по осі X і виконується через метод IsPhysicallyBlocking. Якщо блокування знайдено, система не ламає план, а просто фіксує штраф. Результатом роботи аналізатора є текстовий звіт, який показує, наскільки зручним вийшов план з точки зору реального водія або складу.

Ще один сценарій, який не вкладається в класичну модель, - дозавантаження. У цьому випадку система не починає з нуля, а працює з уже частково завантаженим кузовом. Для цього використовується тип FrozenPlacement, який описує зафіксовані коробки з попередніх хвиль. Під час нового виклику Solve ці об’єми одразу вважаються зайнятими, але їхня вага продовжує враховуватись при розрахунку осьових навантажень. Таким чином, система поводиться як stateful, хоча кожен виклик формально залишається незалежним.

Завдяки цьому Pack3D досить легко інтегрується у REST-сервіси та хмарні системи, при цьому залишаючи здатність завжди зберігати інформацію про реальний стан завантаження.

3.2. Розробка системи візуалізації та діагностики

Сухі цифри координат та кутів повороту, які генерує наш алгоритм, є непридатними для безпосереднього сприйняття людиною. Кінцевий користувач системи - оператор складу або логіст - потребує наочної інструкції. Тому невід'ємною частиною Pack3D є окрема система візуалізації. Її основна задача трансформування абстрактної математичної моделі у набір зрозумілих графічних схем.

3.2.1 Проєктування інструменту візуалізації (SVG) для пошарового аналізу завантаження

Для формування графічного представлення результатів пакування було обрано формат SVG. Він найбільше відповідає вимогам інженерної документації та експлуатаційному аналізу логістичних систем. Ключовою перевагою SVG є його векторна природа, що забезпечує масштабованість без втрати якості. Схеми завантаження можуть використовуватися як у вигляді великоформатних друкованих матеріалів для рампи, так і на екранах мобільних терміналів збору даних.

Також перевагою SVG є структура, орієнтована на XML специфіку. Це означає, що SVG забезпечує тісну інтеграцію з DOM у веб-середовищах. Це дозволить в перспективі вбудовувати згенеровані схеми у клієнтські інтерфейси та доповнювати їх інтерактивною поведінкою. До прикладу, можливе підсвічування тих чи інших коробок, фільтрація за ідентифікаторами або контекстним аналізом розміщень. Семантика SVG-елементів, як-от <g>, <rect>, <text> також відкриває можливості для індексації та подальшого аудиту схем. Це реалізовано за допомогою внутрішніх аналітичних інструментів.

Підсистема візуалізації реалізована у вигляді статичного класу Visualizer. Він не містить бізнес-логіки та виконує виключно функцію транслятора результатів у графічне представлення (рис. 3.3). Ключовим методом є RenderLayerSvg, що перетворює тривимірну сцену завантаження на серію двовимірних пошарових зрізів. Такий підхід дозволяє аналізувати структуру пакування по висоті, що є критично важливим для оцінки стабільності, доступності вантажу та відповідності експлуатаційним вимогам.

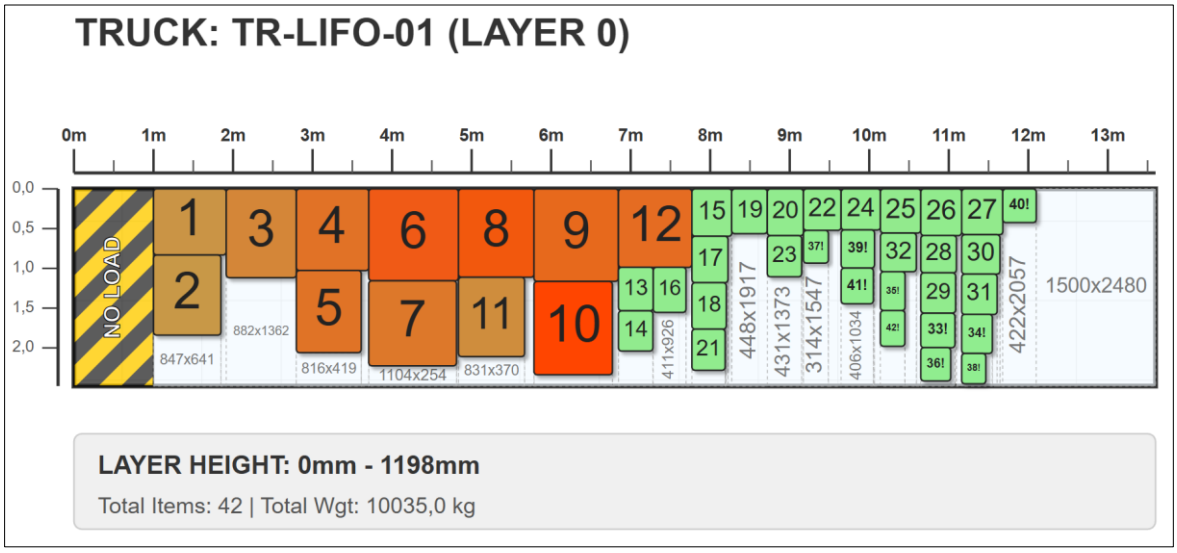

Рисунок 3.3 – Результат роботи SVG візуалізатора окремого шару

Коли система малює черговий шар, вона спочатку просто відсікає все зайве - беруться лише ті Placement, які реально належать до цього рівня. Далі починається переклад з “математики” в “картинку”: тривимірні координати переводяться у

двовимірний SVG. Принцип простий - звичайна лінійна пропорція між міліметрами й пікселями, плюс масштаб і відступи, щоб усе не масштабувалося до країв.

Окремим та вкрай важливим аспектом є заборонені зони. Їх не можна ігнорувати. Для кожного шару доводиться перевіряти: перетин по висоті Якщо перетин присутній - зона з’являється на схемі з характерною штриховкою. Такий підхід убезпечує від теоретичних помилок розміщення.

Крім коробок, на схемі видно і те, що залишилось порожнім. Ці шматки простору, зазвичай, з’являються автоматично після пакування і надають ряд корисної інформації. Якщо їх забагато або вони дивної форми - значить, алгоритм працює некоректно. Візуально вони відображуються напівпрозорими, щоб не перевантажувати UI.

З метою підвищення наочності оцінювання вагових характеристик вантажу застосовується колірна індикація за принципом «холодно - тепло». Легкі коробки відображаються у відтінках зеленого кольору, тоді як важкі - у відтінках червоного. Колір кожного об’єкта обчислюється динамічно на основі нормалізації значень ваги в межах мінімального та максимального показників у поточній партії вантажу. Такий підхід дозволяє з одного погляду ідентифікувати зони концентрації важкого вантажу та оцінити коректність його розміщення по висоті.

Окрему увагу приділено візуальному розрізненню типів вільного простору в контейнері. Простір, безпосередньо доступний на рівні підлоги, позначається окремо, оскільки він може бути використаний для розміщення додаткової палети або важкого вантажу. Натомість порожній простір, розташований у верхніх зонах, має обмежену практичну цінність і відображається іншим способом. Така диференціація дозволяє уникнути помилкової інтерпретації показників залишкового об’єму та забезпечує більш об’єктивну оцінку ефективності завантаження.

3.2.2 Проєктування клієнтського застосунку та підтримка динамічних сценаріїв відвантаження

Хоча обчислювальне ядро Pack3D оформлене як бібліотека класів, для його практичної перевірки та демонстрації роботи інтерфейсу було створено окремий консольний клієнт - Pack3D.Client. Фактично цей застосунок виконує роль спрощеного емулятора TMS-системи: він готує вхідні дані, задає параметри сценарію пакування та збирає результати у вигляді, зручному для подальшого аналізу.

Клієнт реалізовано у файлі Program.cs, а логіка його роботи побудована так, щоб максимально наближатися до реального процесу планування завантаження. На старті ініціалізується сценарій: створюється об’єкт Trailer з параметрами, які відповідають типовому напівпричепу. Для ускладнення задачі в передній частині трейлера задається конструктивна перешкода - холодильний агрегат, представлений у вигляді забороненої зони. Такий елемент дозволяє перевірити, чи коректно алгоритм працює з внутрішніми обмеженнями та чи не намагається «поставити коробку туди, де її фізично бути не може».

Далі формується тестового навантаження за допомогою компонента ScenarioGenerator виведено ряд стандартизованих сценаріїв. До прикладу, важкі палети для перевірки навантаження на підлогу, довгі об’єкти для тестування логіки орієнтації та поворотів, крихкі коробки для перевірки правил Fragile, а також вантажі з різними індексами доставки. Останнє з переліченого, особливо важливо для перевірки коректної роботи LIFO-логіки в сценаріях з кількома точками розвантаження.

Після підготовки даних запускається основний розрахунок шляхом виклику методу Solver.Solve з відповідними параметрами оптимізації. Результат роботи алгоритму не залишається лише у внутрішньому форматі. Повний об’єкт результату серіалізується у JSON для потенційної інтеграції з зовнішніми системами. Додатково формується табличний звіт у форматі CSV, який може бути безпосередньо відкритий у табличних редакторах, зокрема Microsoft Excel (рис.

3.4). Компонент Exporter при цьому враховує локалезалежні нюанси, такі як роздільники та формат чисел з плаваючою комою, що зменшує ризик помилок під час подальшої обробки даних. Окрім цього, клієнтський застосунок генерує набір SVG-файлів із пошаровими зрізами плану завантаження, які використовуються для наочного та швидкого аналізу отриманого рішення.

	Sequence	BoxId	Length	Width	Height	Weight	X	Y	Z	Rotation	Layer	SteerKg	DriveKg	TandemKg
2	1	Box_15	6000	500	346	208	0	0	0	0;0;0	0	18.5	156.8	32.7
3	2	Box_3	6000	284	712	243	6000	0	0	0;0;0	0	4.3	36.6	202.0
4	3	Box_18	6000	243	942	275	6000	0	284	0;0;0	0	4.9	41.5	228.7
5	4	Box_17	6000	248	668	199	6000	0	527	0;0;0	0	3.5	30.0	165.5
6	5	Box_20	6000	485	522	304	6000	0	775	0;0;0	0	5.4	45.8	252.8
7	6	Box_5	6000	304	715	261	6000	0	1260	0;0;0	0	4.6	39.4	217.0
8	7	Box_10	6000	309	946	351	6000	0	1564	0;0;0	0	6.2	52.9	291.8
9	8	Box_14	6000	566	374	254	6000	0	1873	0;0;0	0	4.5	38.3	211.2
10	9	Box_12	6000	333	816	326	0	0	500	0;0;0	0	28.9	245.8	51.3
11	10	Box_8	6000	295	989	350	0	0	833	0;0;0	0	31.0	263.9	55.1
12	11	Box_11	6000	590	527	373	0	0	1128	0;0;0	0	33.1	281.2	58.7
13	12	Box_4	6000	394	345	163	0	0	1718	0;0;0	0	14.5	122.9	25.6
14	13	Box_1	6000	381	567	259	6000	374	1873	0;0;0	1	4.6	39.1	215.3
15	14	Box_6	6000	521	530	331	0	527	1128	0;0;0	1	29.4	249.6	52.1
16	15	Box_9	6000	341	893	365	0	0	2112	0;0;0	0	32.4	275.2	57.4
17	16	Box_16	6000	235	721	203	0	346	0	0;0;0	1	18.0	153.1	31.9
18	17	Box_19	6000	459	331	182	6000	522	775	0;0;0	1	3.2	27.4	151.3
19	18	Box_2	6000	259	458	142	0	346	235	0;0;0	1	12.6	107.1	22.3

Рисунок 3.4 – Приклад експортованих даних у форматі CSV через компонент Exporter

Окремої уваги в системі приділено сценаріям часткового завантаження, які в логістиці зазвичай називають rolling shipments або дозавантаженням. У реальних умовах це типовий кейс: частина вантажу вже знаходиться у трейлері або закріплена раніше затвердженим планом, і нову партію необхідно додати так, щоб не чіпати те, що вже завантажено. Для підтримки такого режиму в методі Solve передбачено додатковий параметр frozen, який містить перелік зафіксованих розміщень у вигляді об’єктів FrozenPlacement.

На етапі попередньої обробки ці заморожені розміщення сприймаються системою як статичні елементи простору - по суті, як ще одна група внутрішніх перешкод поряд із конструктивними особливостями трейлера. Об’єми, зайняті вже наявним вантажем, вирізаються з доступного простору за допомогою алгоритму гільйотинного розрізу, після чого вони повністю зникають зі списку FreeSpaces. Це просте, але надійне рішення, яке унеможливлює випадкове розміщення нового вантажу у фізично зайнятих зонах.

Водночас система не «забуває» про фізику вже виконаного завантаження. Хоча заморожені об’єкти не беруть участі в оптимізації та не змінюють своє положення, їх маса і координати враховуються при обчисленні початкового центру

мас та осьових навантажень. Завдяки цьому додавання нових коробок відбувається з урахуванням реального стану трейлера і не призводить до порушення вагових або експлуатаційних обмежень.

Завершальним кроком роботи є перевірка коректності сформованого плану з точки зору розвантаження. Для цього використовується окремий компонент UnloadingAnalyzer, який аналізує результат роботи Solver за принципом LIFO (рис. 3.5). Після завершення пакування клієнт передає об’єкт SolveResult до аналізатора, який групує всі розміщення за значенням DeliveryStopIndex та послідовно імітує процес розвантаження на кожній зупинці, перевіряючи фізичну доступність кожного елемента.

```
--- SCENARIO: 1. Logistics LIFO Mix ---
Time: 30ms | Placed: 42/42
Util: 16,6% Vol | 41,8% Wgt
WARNING: Significant Side Imbalance!
=== UNLOADING SIMULATION & TIME ESTIMATION ===

[STOP 1] Unloading...
  -> WARN: Item S1_Fragile_2 blocked by 1 items (e.g. S2_Mid_17). Delay +3m
  -> WARN: Item S1_Fragile_7 blocked by 4 items (e.g. S2_Mid_1). Delay +12m
  -> WARN: Item S1_Fragile_10 blocked by 2 items (e.g. S2_Mid_9). Delay +6m
  -> WARN: Item S1_Fragile_9 blocked by 6 items (e.g. S2_Mid_10). Delay +18m
  -> WARN: Item S1_Fragile_5 blocked by 5 items (e.g. S2_Mid_1). Delay +15m
  -> WARN: Item S1_Fragile_4 blocked by 6 items (e.g. S2_Mid_10). Delay +18m
  Items: 10 | Blocked: 6 | Est. Time: 80,0 min

[STOP 2] Unloading...
  Items: 20 | Blocked: 0 | Est. Time: 16,0 min

[STOP 3] Unloading...
  Items: 12 | Blocked: 0 | Est. Time: 30,0 min
```

Рисунок 3.5 – Логування клієнтської частини при симуляції вивантаження вантажу

Під час імітації перевіряється наявність фізичних «блокерів» - коробок з пізнішими зупинками, що геометрично перекривають доступ до цільового вантажу вздовж осі X. Для цього виконується перевірка IsPhysicallyBlocking, і у випадку виявлення блокування до результату додається штрафний час. За підсумком UnloadingAnalyzer формує текстовий звіт про якість плану, реалізуючи чітке розділення відповідальності: Solver відповідає за побудову плану, а аналізатор - за незалежну перевірку його практичної придатності.

3.3. Експериментальні дослідження та оцінка ефективності роботи системи за просторовими, ваговими та маршрутними критеріями.

Щоб перевірити, як система поводиться в умовах, максимально близьких до реальних, було використано спеціалізований генератор сценаріїв. Його завданням було не просто створити тестові набори даних, а відтворити ті самі «незручні» ситуації, з якими логістика стикається щодня: конструктивні обмеження всередині кузова (колісні арки, рефрижераторні модулі), вимоги до крихкого вантажу та жорсткі маршрутні сценарії, де навіть незначне порушення принципу LIFO робить план завантаження непридатним до виконання.

Кількісну оцінку якості сформованих планів виконували за допомогою модуля UnloadingAnalyzer. Цей інструмент дозволив перевести візуальні схеми пакування у набір чітких числових показників, завдяки чому стало можливим не лише зафіксувати факт успішного завантаження, а й швидко виявити та локалізувати системні «вузькі місця» у логістичному ланцюгу.

Для комплексного тестування системи сформовано шість еталонних сценаріїв (Benchmark Scenarios), які охоплюють основні класи задач пакування (див. табл. 4.1). Вхідні дані для кожного сценарію генерувалися через клас Pack3D.DataGen.ScenarioGenerator із використанням псевдовипадкових розподілів габаритів та маси вантажів. Такий підхід забезпечує відтворюваність експериментів та дозволяє повторно перевіряти поведінку алгоритму при різних комбінаціях параметрів.

Таблиця 3.4 - Характеристика тестових сценаріїв для експериментального дослідження

№	Назва сценарію	Опис проблеми	Кількість одиниць (N)	Ключові обмеження
1	Logistics LIFO Mix	Багатохвильова доставка (3 зупинки)	42	LIFO, Forbidden Zone (холодильник), різнорідна щільність.
2	Wheel Wells Constraint	Вантажівка з конструктивними перешкодами (арки).	40	Геометричні "вирізи" в підлозі, широкі палети.
3	Fragile Glass	Перевезення крихкого вантажу (скло).	50	Fragile = true, заборона штабелювання зверху.
4	Long Pipes	Довгомірні вантажі.	20	Довжина 6м, складність повороту.
5	E-Commerce	Дрібні посилки ("розсип").	150	Велика кількість дрібних об'єктів, висока фрагментація.
6	Palletized Load	Стандартні європалети.	33	Однорідний вантаж, висока щільність.

Експеримент проводився на ПК з процесором AMD Ryzen AI 7 та 32 ГБ ОЗУ. Для кожного сценарію фіксувалися такі показники:

- час розрахунку (T_{calc}): тривалість роботи алгоритму Solver.Solve;
- коефіцієнт використання об'єму (U_{vol});
- коефіцієнт використання вантажопідйомності (U_{weight});
- показник LIFO-конфліктів: час затримки при розвантаженні, розрахований UnloadingAnalyzer;

- балансування осей: відхилення навантаження на Steer, Drive та Tandem від нормативів.

3.3.1 Аналіз ефективності алгоритму в умовах обмежень LIFO

Сценарій "Logistics LIFO Mix" є найбільш показовим. Він поєднує геометричні обмеження з маршрутними. Вхідні дані передбачають три точки вивантаження (Stop 1, Stop 2, Stop 3). Важливо розуміти, що Stop 1 - це остання точка завантаження (ближче до дверей), а Stop 3 - перша (глибина кузова).

Результати роботи модуля UnloadingAnalyzer показали, що система успішно застосувала механізм "віртуальних стін". Розподіл вантажу (рис. 3.6) вздовж поздовжньої осі трейлера (вісь X) відбувся наступним чином:

- вантажі Stop 3 (глибинні): розміщено в діапазоні $X \in [0; 6500]$ мм;
- вантажі Stop 2 (середні): розміщено в діапазоні $X \in [6500; 10200]$ мм;
- вантажі Stop 1 (біля дверей): розміщено в діапазоні $X \in [10200; 13600]$ мм.

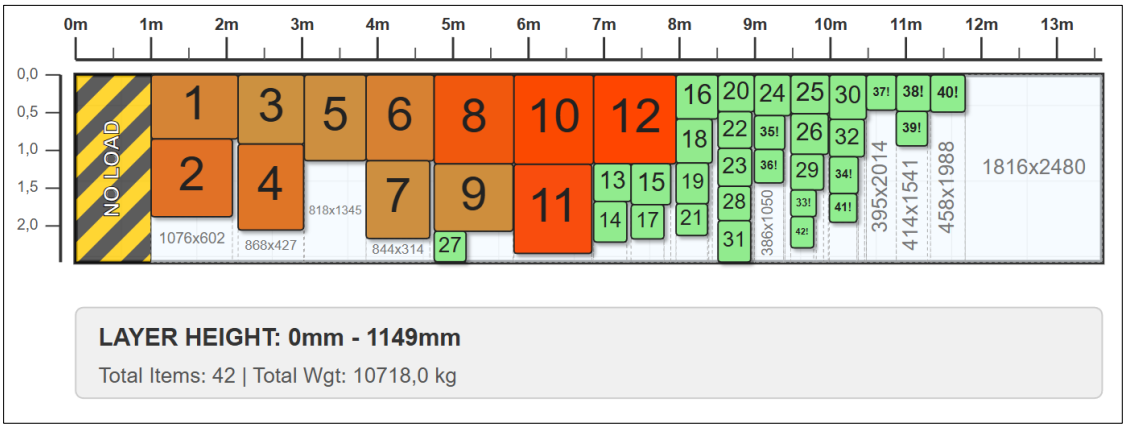

Рисунок 3.6 – Візуалізація розміщення вантажів в умовах обмежень LIFO

Симуляція розвантаження, проведена методом UnloadingAnalyzer. AnalyzeUnloading, зафіксувала незначні блокування вантажу (3 одиниці з 42 можливих) та незначне перевантаження однієї сторони кузова (рис. 3.7).

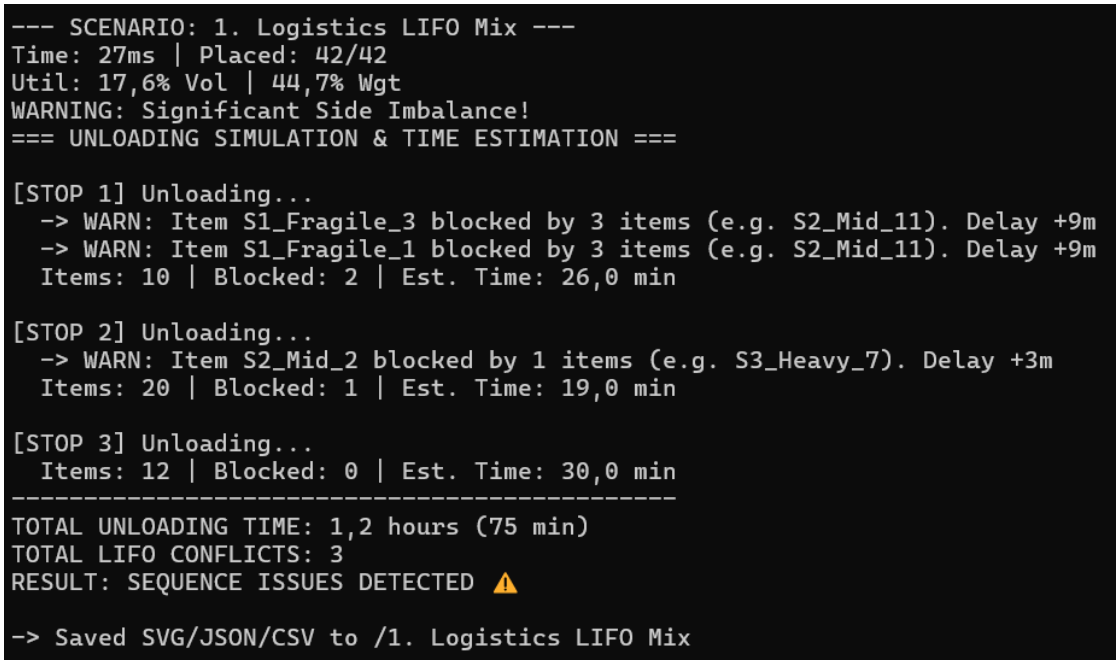

Рисунок 3.7 – Консольне логування для першого експерименту

3.3.2 Дослідження роботи з геометричними та фізичними обмеженнями

У сценарії з колісними арками тестувалася здатність алгоритму "обтікати" статичні перешкоди. У трейлері були визначені дві заборонені зони розміром 2000 × 450 × 400 мм над задніми осями. Ці заборонені зони символізують колісні арки, але принцип їхньої як математичної, так і програмної обробки нічим не відрізняється від холодильної установки. Тобто колісні арки також є статичною областю з забороною на розміщення об’єктів всередині себе.

Візуальний аналіз згенерованих SVG-зрізів показав, що алгоритм:

- не розмістив жодної коробки всередині габаритів арок нульового шару;
- ефективно використав простір між арками для розміщення вузьких коробок (шириною < 1200 мм);
- широкі палети (які конфліктують з арками) були автоматично зміщені вперед або назад відносно зони задніх осей, або підняті на другий ярус (рис. 3.8), де арки вже не заважають (оскільки їх висота обмежена 400 мм).

Рисунок 3.8 - Візуалізація розміщення вантажів для експерименту забороненими зонами розміщення вантажу

3.3.3 Забезпечення цілісності крихкого вантажу

У сценарії "Fragile Glass" 100% вантажу мали маркер Fragile = true. Аналіз результатів показав:

- параметр MaxStackOnTopKg для всіх нижніх коробок не був порушений;
- система надала перевагу розміщенню вантажів в один шар по всій площі підлоги (рис. 3.9), мінімізувавши вертикальне штабелювання;
- коефіцієнт використання об'єму (U_{vol}) впав до 25.5%, що є очікуваним та прийнятним результатом для даного типу вантажу заради забезпечення його збереження.

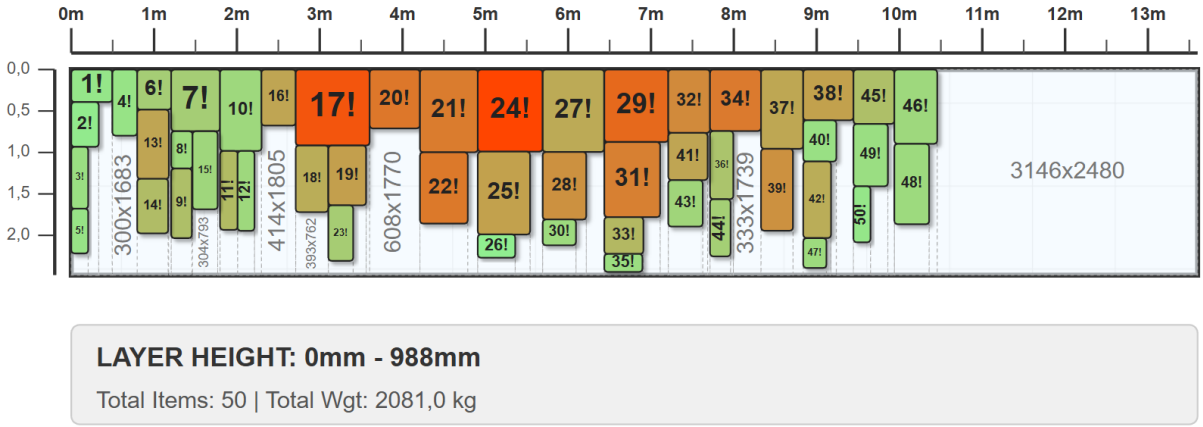

Рисунок 3.9 – Візуалізація розміщення вантажів при транспортуванні крихких об’єктів

3.3.4 Аналіз розподілу навантаження на осі (Axle Weight Distribution)

Однією з ключових переваг системи Pack3D є інтегрований фізичний рушій. Для перевірки його ефективності було проаналізовано ряд сценаріїв. Сценарій 4 та 6 симулює завантаження високогабаритного вантажу високої маси, а сценарій 5, навпаки, - низькогабаритного вантажу (рис 3.10).

```
--- SCENARIO: 4. Long Pipes ---
Time: 0ms | Placed: 17/20
Util: 25,2% Vol | 19,1% Wgt
=== UNLOADING SIMULATION & TIME ESTIMATION ===

[STOP 1] Unloading...
  Items: 17 | Blocked: 0 | Est. Time: 15,3 min
------------------------------------------------
TOTAL UNLOADING TIME: 0,3 hours (15 min)
TOTAL LIFO CONFLICTS: 0
RESULT: PERFECT LIFO SEQUENCE ✅

-> Saved SVG/JSON/CSV to /4. Long Pipes

--- SCENARIO: 5. E-Commerce ---
Time: 43ms | Placed: 150/150
Util: 2,7% Vol | 2,0% Wgt
WARNING: Significant Side Imbalance!
=== UNLOADING SIMULATION & TIME ESTIMATION ===

[STOP 1] Unloading...
  Items: 150 | Blocked: 0 | Est. Time: 120,0 min
------------------------------------------------
TOTAL UNLOADING TIME: 2,0 hours (120 min)
TOTAL LIFO CONFLICTS: 0
RESULT: PERFECT LIFO SEQUENCE ✅

-> Saved SVG/JSON/CSV to /5. E-Commerce

--- SCENARIO: 6. Pallets ---
Time: 0ms | Placed: 33/33
Util: 18,7% Vol | 14,2% Wgt
=== UNLOADING SIMULATION & TIME ESTIMATION ===

[STOP 1] Unloading...
  Items: 33 | Blocked: 0 | Est. Time: 26,4 min
------------------------------------------------
TOTAL UNLOADING TIME: 0,4 hours (26 min)
TOTAL LIFO CONFLICTS: 0
RESULT: PERFECT LIFO SEQUENCE ✅

-> Saved SVG/JSON/CSV to /6. Pallets
```

Рисунок 3.10 – Консольне логування для перевірки навантаження на осі

Було виявлено кореляцію між габаритами та масою. Таким чином, алгоритм добре обробляє високогабаритні вантажі (рис. 3.11), коректно балансуючи навантаження між осями, але чим дрібніший вантаж – тим більше балансує його до (0, 0, 0) координати, розміщуючи вантаж ближче до лівої стінки кузова.

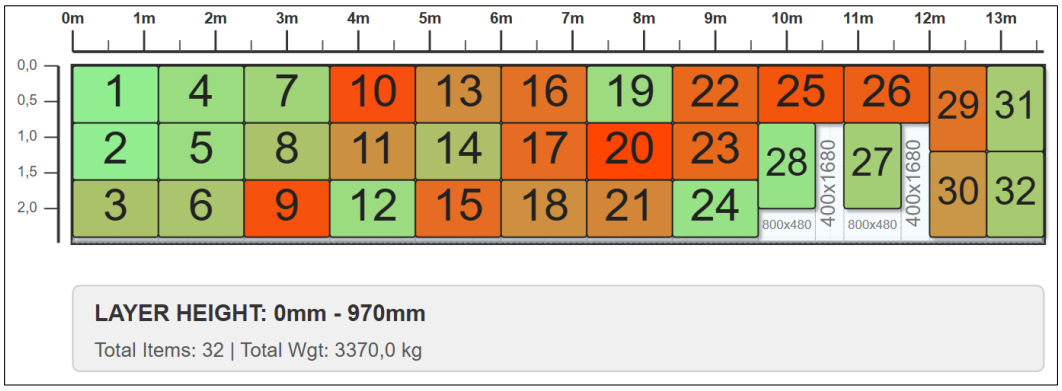

Рисунок 3.11 - Візуалізація розміщення вантажів для високогабаритного та масивного вантажу

3.4 Узагальнені показники продуктивності системи

Узагальнені результати прогону всіх тестових сценаріїв наведені на діаграмі продуктивності (рис. 3.12). Середній час розрахунку для сценаріїв до 150 об'єктів склав менше 100 мс, що підтверджує придатність системи для використання в режимі реального часу (Real-time).

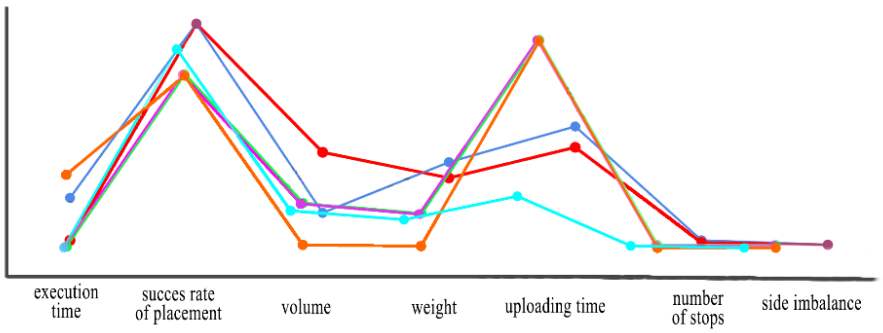

Рисунок 3.12 – Загальна діаграма продуктивності розроблюваного програмного забезпечення

Аналіз системи логування і вищенаведеної діаграми продуктивності показав, що Pack3D працює досить стабільно. Час виконання алгоритму, зазвичай, коливається в межах 0–48 мс, навіть якщо сценарії сильно різняться за кількістю об’єктів і складністю обмежень. У більшості випадків пакування вдається повністю, проте варто зауважити, що логіка алгоритму потребує сильного доопрацювання через конфліктність деяких математичних та програмних аспектів та рішень.

З аповнення об’єму та розподіл ваги змінюються від сценарію до сценарію - це природньо: легкі e-commerce відправлення зовсім не схожі на щільно складені набори вантажу з колісними нішами. Що стосується ефективності з точки зору логістики, система добре справляється з моделюванням розвантаження. У більшості тестів LIFO-послідовність зберігалась без конфліктів, а час розвантаження залежав від кількості коробок, а не від обчислювальних витрат.

Єдине «але» - сценарій Logistics LIFO Mix, де на першій зупинці зафіксовано один локальний конфлікт, трохи збільшивши час розвантаження, проте система його відразу ідентифікувала. Попередження про дисбаланс у деяких випадках теж не є

помилкою - це сигнали про реальні фізичні ризики, які Pack3D показує ще на етапі планування.

У підсумку результати підтверджують: алгоритм працює швидко, надійно і стійко до різнорідних даних.

Висновки до 3 розділу

У третьому розділі робота переходить від теорії до практики й показує, як Pack3D поводиться у реальних сценаріях використання. Тут розглянуто не абстрактні моделі, а конкретні інженерні рішення, на яких тримається система. Вибір платформи .NET Core та мови C# був зумовлений не зручністю, а вимогами до стабільної роботи під навантаженням і передбачуваної поведінки в довгих обчислювальних циклах. Використання struct і Span<T> стало практичним способом зменшити затримки доступу до пам’яті, що критично важливо при обробці потокових даних у режимі, близькому до реального часу.

Обчислювальне ядро організовано як ізольований конвеєр із єдиною точкою входу, що спрощує контроль виконання навіть у нестандартних ситуаціях - наприклад, під час дозавантаження (Rolling Shipments) або роботи з нерегулярною геометрією кузова. Для перевірки коректності результатів застосовано подвійний підхід: візуальний аналіз через SVG-зрізи та логічну перевірку за допомогою модуля UnloadingAnalyzer. Це дозволяє виявляти колізії та порушення LIFO ще на етапі розрахунків, до того як помилки перетворюються на проблеми під час реальних складських операцій.

ВИСНОВКИ

У межах магістерської роботи було розв’язано прикладну задачу, з якою регулярно стикається транспортна логістика, - підвищення ефективності вантажних перевезень. Підсумком цієї роботи стала автоматизована система Pack3D, розробка якої відбувалася поетапно, від аналізу існуючих рішень до практичної реалізації та перевірки результатів.

Початковий етап дослідження показав суттєве обмеження більшості підходів до задачі 3D-BPP. Значна частина наявних рішень зосереджується виключно на геометрії, фактично ігноруючи фізичні умови реального перевезення - зокрема розподіл навантаження на осі та вимоги до черговості розвантаження (LIFO). Саме це спостереження стало поштовхом до переходу від «чисто геометричної» логіки до гібридної моделі, у якій евристичні методи працюють разом із фізичним моделюванням.

В основі Pack3D лежить модифікований варіант стратегії Best-Fit Decreasing, доповнений механізмом гільйотинного розрізу (Guillotine Split) для керування вільним простором. Проте ключовим елементом системи стала динамічна скорингова функція. Вона дозволяє одночасно враховувати кілька суперечливих факторів: щільність укладання, стійкість вантажу та обмеження по осьових навантаженнях на тягач (Steer, Drive) і напівпричеп (Tandem). На практиці це означає, що алгоритм не «ганяється» за максимальною заповненістю будь-якою ціною, а приймає збалансовані рішення.

Алгоритмічна частина була реалізована у вигляді програмного комплексу Pack3D на платформі .NET Core. Оптимізація архітектури дозволила досягти часу обробки запитів менше 100 мс навіть у складних сценаріях, що робить систему придатною для використання у режимі, близькому до реального часу. Окрему увагу приділено практичному застосуванню: підтримка SVG-візуалізації та експорт результатів у форматах JSON і CSV дозволяють інтегрувати Pack3D у наявні TMS/WMS-системи без додаткової адаптації.

Перевірка системи на контрольних сценаріях підтвердила її здатність працювати з різнорідним, у тому числі крихким вантажем, без втрати ефективності. Симуляція процесів розвантаження показала стабільне дотримання принципу LIFO, що на практиці зменшує простої транспорту та потребу у додаткових складських операціях. У більш широкому сенсі Pack3D виконує не лише роль інструмента планування, а й дозволяє знизити вплив людського фактора, уникати перевантаження осей і підвищувати загальний рівень безпеки перевезень.

Перспективи подальших досліджень. Потенціал розвитку системи не обмежується поточною реалізацією. Одним із логічних напрямів подальшої роботи є інтеграція елементів машинного навчання. Якщо нині алгоритм працює за детермінованими правилами, то використання історичних даних могло б підвищити його адаптивність і дозволити краще працювати з нетиповими сценаріями та специфічними видами вантажу.

Не менш важливим кроком залишається перевірка системи в реальних умовах експлуатації. Використання Pack3D на реальному автопарку дасть змогу виявити нюанси, які неможливо повністю відтворити у симуляції. Зворотний зв’язок від логістів і водіїв може стати основою для остаточного налаштування параметрів алгоритму та адаптації системи до практичних вимог ринку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Romero-Olarte N., Amézquita-Ortiz S., Escobar J. W., Álvarez-Martínez D. Decision Support System to Solve Single-Container Loading Problem Considering Practical Constraints. *Mathematics*. 2025. Vol. 13, No. 10. Art. 1668. DOI: 10.3390/math13101668.
2. Martello S., Pisinger D., Vigo D. The Three-Dimensional Bin Packing Problem. *Operations Research*. 2000. Vol. 48, No. 2. P. 256–267.
3. Cordeau J. F., Iori M., Laporte G., Vigo D. Fifty Years of Vehicle Routing. *Transportation Science*. 2007. Vol. 41, No. 4. P. 565–598.
4. Johnson D. S. Near optimal bin packing algorithms. Tech. Report MAC TR-109. Cambridge : Mass. Inst. of Tech., Project MAC, 1973.
5. Coffman E. G., Garey M. R., Johnson D. S. Approximation algorithms for bin packing: a survey. *Approximation algorithms for NP-hard problems*. PWS Publishing Co., 1996. P. 46–93.
6. Chazelle B. The bottom-left bin-packing heuristic: An efficient implementation. *IEEE Transactions on Computers*. 1983. Vol. C-32, No. 8. P. 697–707.
7. George J. A., Robinson D. F. A heuristic for packing boxes into a container. *Computers & Operations Research*. 1980. Vol. 7, No. 3. P. 147–156.
8. Bortfeldt A., Wäscher G. Constraints in Container Loading – A State-of-the-art Review. *European Journal of Operational Research*. 2013. Vol. 229, No. 1. P. 1–20.
9. Garey M. R., Johnson D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco : W. H. Freeman & Co, 1979. 338 p.
10. Jylänki J. A Thousand Ways to Pack the Bin – A Practical Approach to Two-Dimensional Rectangle Bin Packing. *Finite Mass*. 2010. URL: http://clb.demon.fi/files/RectangleBinPack.pdf (дата звернення: 17.12.2025).

11. Hopper E., Turton B. C. H. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. *European Journal of Operational Research*. 2001. Vol. 128, Iss. 1. P. 34–57.
12. Bischoff E. E., Ratcliff M. S. W. Issues in the development of approaches to container loading. *Omega*. 1995. Vol. 23, No. 4. P. 377–390.
13. Code of Federal Regulations. Title 23 - Highways. Part 658 - Truck Size and Weight, Route Designations - Length, Width and Weight Limitations. Washington, D.C. : U.S. Government Publishing Office.
14. Krone Commercial Vehicle Group. Load securing and load distribution constraints. Technical Manual. Werlte : Fahrzeugwerk Bernard Krone GmbH & Co. KG, 2019. 45 p.
15. Fitch J. W. *Motor truck engineering handbook*. 4th ed. Warrendale, PA : Society of Automotive Engineers, 1994. 436 p.
16. Pollaris H., Braekers K., Caris A., Janssens G. K., Limbourg S. Vehicle routing problems with loading constraints: state-of-the-art and future directions. *OR Spectrum*. 2015. Vol. 37, No. 2. P. 297–330.
17. Iori M., Martello S. Routing problems with loading constraints. *TOP*. 2010. Vol. 18, No. 1. P. 4–27.
18. Alonso M. T., Alvarez-Valdes R., Parreno F., Tamarit J. M. Algorithms for the vehicle routing problem with three-dimensional loading constraints. *Annals of Operations Research*. 2016. Vol. 244. P. 269–289.
19. Eley M. Solving container loading problems with cargo stability constraints. *European Journal of Operational Research*. 2002. Vol. 141, Iss. 2. P. 393–409.
20. Utility Trailer Manufacturing Co. Driver’s Guide and Reference Manual for 3000R Refrigerated Vans. City of Industry, CA : Utility Trailer, 2018. 86 p.
21. Wäscher G., Haußner H., Schumann H. An improved typology of cutting and packing problems. *European Journal of Operational Research*. 2007. Vol. 183, Iss. 3. P. 1109–1130.

22. Trivella A., Pisinger D. The Load-Balanced Multi-Dimensional Bin-Packing Problem. *Computers & Operations Research*. 2016. Vol. 76. P. 34–53.
23. Benavent E., Landete M., Mota E., Tirado G. The multiple vehicle pickup and delivery problem with LIFO constraints. *European Journal of Operational Research*. 2015. Vol. 243, Iss. 3. P. 752–762.
24. Gillespie T. D. *Fundamentals of Vehicle Dynamics*. Warrendale, PA : Society of Automotive Engineers, 1992. 495 p.
25. Fundamentals of garbage collection [Електронний ресурс] // Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/standard/garbage-collection/fundamentals (дата звернення: 10.12.2025).
26. Why Records Exist in C#: A Modern Approach to Immutable Data [Електронний ресурс] // Medium. URL: https://medium.com/@serasiyasavan14/why-records-exist-in-c-a-modern-approach-to-immutable-data-331a194d25c8 (дата звернення: 17.12.2025).
27. Why .NET Standard Matters for Modern Development [Електронний ресурс] // Softacom. URL: https://www.softacom.com/wiki/why-net-standard-matters-for-modern-development/ (дата звернення: 10.12.2025).
28. LINQ Performance Optimization Tips and Tricks [Електронний ресурс] // TheCodeMan. URL: https://thecodeman.net/posts/linq-performance-otpimization-tips-and-tricks (дата звернення: 10.12.2025).

ДОДАТОК А – Код DataContracts.cs

```
using System;
using System.Collections.Generic;

namespace Pack3D
{
    [Flags] public enum RotationMask
{ None = 0, X = 1, Y = 2, Z = 4, All
= 7 }
    public record Dimensions(double
L, double W, double H);
    public record Vector3(double X,
double Y, double Z);

    public record AxlePositions(
        double
KingpinDistanceFromFrontMm, double
TractorWheelbaseMm,
        double
DriveAxleDistanceFromFrontMm, double
TandemAxleDistanceFromFrontMm
    );

    public record AxleLimits(
        double SteerMinKg, double
SteerMaxKg,
        double DriveMinKg, double
DriveMaxKg,
        double TandemMinKg, double
TandemMaxKg
    );

    public record Box(
        string Id, string Sku,
Dimensions SizeMm, double WeightKg,
        RotationMask
AllowedRotations, double
Stackability, double
MaxStackOnTopKg,
        bool Fragile, string?
PalletId, string Status, int
Priority = 0, int DeliveryStopIndex
= 1
    );

    public record Pallet(string Id,
Dimensions SizeMm, double
TareWeightKg, double
MaxStackOnTopKg,
IReadOnlyList<string> BoxIds);

    public record Rect3(double X,
double Y, double Z, double L, double
W, double H, int LayerIndex);

    public record Trailer(
        string Id, Dimensions
InnerMm, Dimensions DoorMm, double
MaxPayloadKg,
        AxleLimits Axles,
AxlePositions AxleGeo, double
FloorStrengthPsf,
        IReadOnlyList<Rect3>
ForbiddenZones, Vector3 Origin
    );

    public record Placement(
        string BoxId, Vector3
PositionMm, (int X, int Y, int Z)
Rotation,
        int SequenceIndex, int
LayerIndex,
        (double SteerKg, double
DriveKg, double TandemKg)
AxleContribution,
        double ContactAreaCm2,
IReadOnlyList<string> Notes
    );

    public record LoadPhysics(
        Vector3 CenterOfGravityMm,
        double MomentX,
        double MomentY,
        double MomentZ
    );

    public record Metrics(
        double VolumeUtilizationPct,
double WeightUtilizationPct,
        double TotalWeightKg,
(double SteerKg, double DriveKg,
double TandemKg) AxleTotals,
        LoadPhysics Physics,
        int PlacedCount, int
UnplacedCount
    );

    public record SolveResult(
        IReadOnlyList<Placement>
Placements,
        IReadOnlyList<string>
UnplacedBoxIds,
        IReadOnlyList<Rect3>
FreeSpaces,
        Metrics Metrics,
```

```
        IReadOnlyList<string>
Warnings,
        string? SolverLogPath
    );

    public record
FrozenPlacement(string BoxId,
Vector3 PositionMm, (int X, int Y,
int Z) Rotation, int SequenceIndex,
int LayerIndex);
    public enum Objective {
MaxVolumeUtilization,
MaxWeightUtilization,
MixedUtilizationBalance }
    public record SolveOptions(
        int Seed = 42, TimeSpan
QuickTimeout = default, TimeSpan
ImproveTimeout = default,
        bool EnableImprove = false,
Objective Objective =
Objective.MixedUtilizationBalance,
        bool RespectFrozenPlacements
= true, bool FilterReadyOnly = true
    );
}
```

ДОДАТОК Б – Код unloadingAnalyzer.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Pack3D
{
    public static class
UnloadingAnalyzer
    {
        public static string
AnalyzeUnloading(SolveResult result,
List<Box> allBoxes, Trailer trailer)
        {
            var sb = new
StringBuilder();
            sb.AppendLine("===
UNLOADING SIMULATION & TIME
ESTIMATION ===");

            var boxMap =
allBoxes.ToDictionary(b => b.Id);

            var stops =
result.Placements
                .Select(p =>
boxMap[p.BoxId].DeliveryStopIndex)
                .Distinct()
                .OrderBy(s => s)
                .ToList();

            double totalTimeMinutes
= 0;
            int totalBlockages = 0;

            foreach (var stop in
stops)
            {

sb.AppendLine($"\n[STOP {stop}]
Unloading...");

                var itemsForStop =
result.Placements
                    .Where(p =>
boxMap[p.BoxId].DeliveryStopIndex ==
stop)

.OrderByDescending(p =>
p.PositionMm.X)
                    .ToList();

                double stopTime = 0;
                int blockedCount =
0;

                foreach (var item in
itemsForStop)
                {
                    var box =
boxMap[item.BoxId];

                    double itemTime
= (box.WeightKg > 500) ? 2.5 : 0.8;


                    var blockers =
result.Placements
                        .Where(p =>
boxMap[p.BoxId].DeliveryStopIndex >
stop)
                        .Where(p =>
IsPhysicallyBlocking(p, item,
boxMap))
                        .ToList();

                    if
(blockers.Any())
                    {

blockedCount++;

totalBlockages++;
                        double
penalty = blockers.Count * 3.0;
                        itemTime +=
penalty;


sb.AppendLine($"  -> WARN: Item
{item.BoxId} blocked by
{blockers.Count} items (e.g.
{blockers[0].BoxId}). Delay
+{penalty}m");
                    }

                    stopTime +=
itemTime;
                }

                sb.AppendLine($"
Items: {itemsForStop.Count} |
Blocked: {blockedCount} | Est. Time:
{stopTime:F1} min");
                totalTimeMinutes +=
stopTime;
            }
```

```
            sb.AppendLine("---------
----------------------------------
-");
            sb.AppendLine($"TOTAL
UNLOADING TIME: {totalTimeMinutes /
60.0:F1} hours
({totalTimeMinutes:F0} min)");
            sb.AppendLine($"TOTAL
LIFO CONFLICTS: {totalBlockages}");

            if (totalBlockages == 0)
sb.AppendLine("RESULT: PERFECT LIFO
SEQUENCE ☑");
            else
sb.AppendLine("RESULT: SEQUENCE
ISSUES DETECTED ⚠");

            return sb.ToString();
        }

        private static bool
IsPhysicallyBlocking(Placement a,
Placement b, Dictionary<string, Box>
map)
        {
            var boxA = map[a.BoxId];
            var boxB = map[b.BoxId];

            if (a.PositionMm.X <
b.PositionMm.X) return false;

            double bY1 =
b.PositionMm.Y;
            double bY2 =
b.PositionMm.Y + GetH(b, boxB);
            double bZ1 =
b.PositionMm.Z;
            double bZ2 =
b.PositionMm.Z + GetW(b, boxB);

            double aY1 =
a.PositionMm.Y;
            double aY2 =
a.PositionMm.Y + GetH(a, boxA);
            double aZ1 =
a.PositionMm.Z;
            double aZ2 =
a.PositionMm.Z + GetW(a, boxA);

            bool overlapY = (aY1 <
bY2 && aY2 > bY1);
            bool overlapZ = (aZ1 <
bZ2 && aZ2 > bZ1);

            return overlapY &&
overlapZ;
        }

        private static double
GetH(Placement p, Box b) =>
(p.Rotation.X == 90) ? b.SizeMm.W :
b.SizeMm.H;
        private static double
GetW(Placement p, Box b) =>
((p.Rotation.Y % 180 != 0) ^
(p.Rotation.Z % 180 != 0)) ?
b.SizeMm.L : b.SizeMm.W;
    }
}
```

ДОДАТОК В – Код Exporter.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text.Json;
using System.Text;
using System.Globalization;

namespace Pack3D
{
    public static class Exporter
    {
        public static void
SaveJson(SolveResult result, string
path)
        {
            var options = new
JsonSerializerOptions {
WriteIndented = true };
            string json =
JsonSerializer.Serialize(result,
options);
            File.WriteAllText(path,
json);
        }

        public static void
SaveCsv(SolveResult result,
List<Box> boxes, string path)
        {
            var sb = new
StringBuilder();


sb.AppendLine("Sequence,BoxId,Length
,Width,Height,Weight,X,Y,Z,Rotation,
Layer,SteerKg,DriveKg,TandemKg");

            var boxMap =
boxes.ToDictionary(b => b.Id);
            IFormatProvider ci =
CultureInfo.InvariantCulture;

            foreach (var p in
result.Placements.OrderBy(x =>
x.SequenceIndex))
            {
                var b =
boxMap[p.BoxId];

                string rotString =
$"{p.Rotation.X};{p.Rotation.Y};{p.R
otation.Z}";

sb.AppendLine(string.Format(ci,

"{0},{1},{2},{3},{4},{5},{6},{7},{8}
,{9},{10},{11:F1},{12:F1},{13:F1}",
                    p.SequenceIndex,
                    p.BoxId,
                    b.SizeMm.L,
b.SizeMm.W, b.SizeMm.H, b.WeightKg,
                    p.PositionMm.X,
p.PositionMm.Y, p.PositionMm.Z,
                    rotString,
                    p.LayerIndex,

p.AxleContribution.SteerKg,

p.AxleContribution.DriveKg,

p.AxleContribution.TandemKg
                ));
            }
            File.WriteAllText(path,
sb.ToString());
        }
    }
}
```

ДОДАТОК Г – Код ScenarioGenerator.cs

```
using System;
using System.Collections.Generic;
using Pack3D;

namespace Pack3D.DataGen
{
    public record
SimulationScenario(string Name, string
Description, Trailer Trailer,
List<Box> Boxes);

    public static class
ScenarioGenerator
    {
        private static readonly Random
_rnd = new Random();

        public static
List<SimulationScenario>
GetScenarios()
        {
            var scenarios = new
List<SimulationScenario>();


scenarios.Add(CreateScenario_LifoMix()
);

scenarios.Add(CreateScenario_HeavyTop_
WithWheelWells());

scenarios.Add(CreateScenario_FragileGl
ass());

scenarios.Add(CreateScenario_LongPipes
());

scenarios.Add(CreateScenario_ECommerce
());

scenarios.Add(CreateScenario_Palletize
d());

            return scenarios;
        }



        private static
SimulationScenario
CreateScenario_LifoMix()
        {
            var forbidden = new
List<Rect3> { new Rect3(0, 0, 0, 1000,
2480, 2700, 0) };
            var trailer =
GetStandardTrailer("TR-LIFO-01",
forbidden);

            var boxes = new
List<Box>();

boxes.AddRange(GenerateRandomBoxes("S3
_Heavy", 12, 3, 800, 1200, density:
400, heavy: true));

boxes.AddRange(GenerateRandomBoxes("S2
_Mid", 20, 2, 400, 600, density:
150));

boxes.AddRange(GenerateRandomBoxes("S1
_Fragile", 10, 1, 300, 500, density:
50, fragileChance: 1.0));

            return new
SimulationScenario("1. Logistics LIFO
Mix", "Маршрут з холодильником
спереду.", trailer, boxes);
        }

        private static
SimulationScenario
CreateScenario_HeavyTop_WithWheelWells
()
        {
            var wheelWells = new
List<Rect3>();

            wheelWells.Add(new
Rect3(7500, 0, 0, 2000, 450, 400, 0));

            wheelWells.Add(new
Rect3(7500, 0, 2030, 2000, 450, 400,
0));

            var trailer =
GetStandardTrailer("TR-WHEELS-02",
wheelWells);
            trailer = trailer with {
FloorStrengthPsf = 5000 };

            var boxes = new
List<Box>();


boxes.AddRange(GenerateRandomBoxes("Wi
de_Pallet", 10, 1, 1000, 2400,
density: 200));

boxes.AddRange(GenerateRandomBoxes("St
```

```
d_Box", 30, 1, 400, 600, density:
100));

            return new
SimulationScenario(
                "2. Wheel Wells
Constraint",
                "Трейлер має колісні
арки, що звужують підлогу в зоні
задніх осей.",
                trailer, boxes);
        }

        private static
SimulationScenario
CreateScenario_FragileGlass() { return
CreateSimpleScenario("3. Fragile
Glass", "TR-GLASS", 50, true); }
        private static
SimulationScenario
CreateScenario_LongPipes() { return
CreateSimpleScenario("4. Long Pipes",
"TR-LONG", 20, false, fixedLen: 6000);
}
        private static
SimulationScenario
CreateScenario_ECommerce() { return
CreateSimpleScenario("5. E-Commerce",
"TR-ECOMM", 150, false, maxDim: 300);
}
        private static
SimulationScenario
CreateScenario_Palletized() { return
CreateSimpleScenario("6. Pallets",
"TR-PAL", 33, false, fixedLen: 1200,
fixedWid: 800); }


        private static
SimulationScenario
CreateSimpleScenario(string name,
string trId, int count, bool fragile,
int maxDim = 1000, int? fixedLen =
null, int? fixedWid = null)
        {
            var t =
GetStandardTrailer(trId);
            var b =
GenerateRandomBoxes("Box", count, 1,
200, maxDim, 200, false, fragile ? 1.0
: 0.0, fixedLen, null, fixedWid);
            return new
SimulationScenario(name, "Standard
test.", t, b);
        }

        private static Trailer
GetStandardTrailer(string id,
List<Rect3>? forbidden = null)
        {
            return new Trailer(id, new
Dimensions(13600, 2480, 2700), new
Dimensions(0, 2480, 2700), 24000,
                new AxleLimits(6000,
11500, 8000, 19000, 8000, 24000), new
AxlePositions(1600, 3800, 2000,
10500),
                2000, forbidden ?? new
List<Rect3>(), new Vector3(0, 0, 0));
        }

        private static List<Box>
GenerateRandomBoxes(string prefix, int
count, int stop, int minDim, int
maxDim, double density, bool heavy =
false, double fragileChance = 0.0,
int? fixedLength = null, int?
fixedHeight = null, int? fixedDepth =
null)
        {
            var list = new
List<Box>();
            for (int i = 0; i < count;
i++)
            {
                double L = fixedLength
?? _rnd.Next(minDim, maxDim);
                double W = fixedDepth
?? _rnd.Next(minDim,
fixedLength.HasValue ? 600 : maxDim);
                double H = fixedHeight
?? _rnd.Next(minDim, maxDim);
                double vol = (L * W *
H) / 1e9;
                double w = Math.Max(1,
Math.Round(vol * (heavy ? density * 2
: density)));
                bool frag =
_rnd.NextDouble() < fragileChance;

                list.Add(new
Box($"{prefix}_{i + 1}", prefix, new
Dimensions(L, W, H), w, frag ?
RotationMask.Z : RotationMask.All,
frag ? 0 : 1, frag ? 0 : (heavy ? 2000
: 500), frag, null, "Ready", heavy ?
10 : 1, stop));
            }
            return list;
        }
    }
}
```

ДОДАТОК Д – Код Vizualizer.cs

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Globalization;



namespace Pack3D

{

    public static class Visualizer

    {

        private static readonly (int R, int G, int B) BoxLightColor = (144, 238, 144);

        private static readonly (int R, int G, int B) BoxHeavyColor = (255, 69, 0);

        private static readonly (int R, int G, int B) VoidFloorColor = (240, 248, 255);

        private static readonly (int R, int G, int B) VoidShelfColor = (230, 230, 250);



        public static string RenderLayerSvg(IEnumerable<Placement> placements, IEnumerable<Rect3> freeSpaces, Trailer trailer, Metrics globalMetrics, IReadOnlyDictionary<string, Box> boxById, double pxPerMm, int layer)

        {

            var sb = new StringBuilder();

            IFormatProvider ci = CultureInfo.InvariantCulture;

            var layerPlacements = placements.Where(p => p.LayerIndex == layer).OrderBy(p => p.SequenceIndex).ToList();

            var layerSpaces = freeSpaces.Where(s => s.LayerIndex == layer).ToList();



            double minW = 0, maxW = 1, layerWeight = 0, layerMinY = double.MaxValue, layerMaxY = double.MinValue;

            if (layerPlacements.Any())

            {

                var w = layerPlacements.Select(p => boxById[p.BoxId].WeightKg).ToList();

                layerWeight = w.Sum(); minW = w.Min(); maxW = w.Max();

                foreach (var p in layerPlacements)

                {

                    if (p.PositionMm.Y < layerMinY) layerMinY = p.PositionMm.Y;

                    var b = boxById[p.BoxId];

                    if (p.PositionMm.Y + b.SizeMm.H > layerMaxY) layerMaxY = p.PositionMm.Y + b.SizeMm.H;

                }

            }

            else { layerMinY = 0; layerMaxY = 0; }

            double rangeW = (maxW - minW) == 0 ? 1 : (maxW - minW);
```

```
            int padL = 70;
            int padT = 70;
            int pad = 40;
            int headH = 60;

            int truckW =
(int)(trailer.InnerMm.L * pxPerMm);
            int truckH =
(int)(trailer.InnerMm.W * pxPerMm);
            int tableW = 380;
            int rowH = 22, tHeadH =
30;

            int footerH = 80;
            int leftColH = truckH +
footerH + 20;

            int rightColH = tHeadH +
(layerPlacements.Count * rowH);

            int contentH =
Math.Max(leftColH, rightColH);
            int totW = padL + truckW
+ pad + tableW + pad;
            int totH = padT + headH
+ contentH + pad;

            sb.AppendLine($"<svg
xmlns='http://www.w3.org/2000/svg'
width='{totW}' height='{totH}'
viewBox='0 0 {totW} {totH}'>");

sb.AppendLine("<defs><filter id='sh'
x='-20%' y='-20%' width='140%'
height='140%'><feDropShadow dx='2'
dy='2' stdDeviation='1' flood-
opacity='0.3'/></filter>");

            sb.AppendLine("<pattern
id='haz'
patternUnits='userSpaceOnUse'
width='20' height='20'
patternTransform='rotate(45)'><rect
width='10' height='20'
fill='#ffcc00'/><rect x='10'
width='10' height='20'
fill='#333'/></pattern>");
            sb.AppendLine("<pattern
id='gr' width='50' height='50'
patternUnits='userSpaceOnUse'><path
d='M 50 0 L 0 0 0 50' fill='none'
stroke='#e0e0e0' stroke-
width='1'/></pattern></defs>");
            sb.AppendLine("<rect
width='100%' height='100%'
fill='white'/>");

            sb.AppendLine($"<text
x='{padL}' y='40' font-
family='Arial' font-size='24' font-
weight='bold' fill='#333'>TRUCK:
{trailer.Id} (LAYER
{layer})</text>");

            sb.AppendLine($"<text
x='{totW - pad}' y='40' font-
family='Arial' font-size='16' font-
weight='bold' fill='#555' text-
anchor='end'>Corrugated Supplies
Company, LLC Pack3D</text>");

            double tx = padL, ty =
padT + headH;

            sb.AppendLine($"<line
x1='{tx}' y1='{ty - 10}' x2='{tx +
truckW}' y2='{ty - 10}'
stroke='#333' stroke-width='2'/>");
            for (double m = 0; m <=
trailer.InnerMm.L; m += 1000)
            {
```

```
                double x = tx + (m *
pxPerMm);

sb.AppendLine($"<line
x1='{x.ToString("F1", ci)}' y1='{ty
- 10}' x2='{x.ToString("F1", ci)}'
y2='{ty - 25}' stroke='#333' stroke-
width='1.5'/>");

sb.AppendLine($"<text
x='{x.ToString("F1", ci)}' y='{ty -
30}' font-family='Arial' font-
size='11' font-weight='bold' text-
anchor='middle' fill='#333'>{(m /
1000):F0}m</text>");
            }
            for (double m = 500; m
<= trailer.InnerMm.L; m += 1000)
            {
                double x = tx + (m *
pxPerMm); if (x > tx + truckW)
break;

sb.AppendLine($"<line
x1='{x.ToString("F1", ci)}' y1='{ty
- 10}' x2='{x.ToString("F1", ci)}'
y2='{ty - 18}' stroke='#555' stroke-
width='1'/>");
            }
            sb.AppendLine($"<line
x1='{tx - 10}' y1='{ty}' x2='{tx -
10}' y2='{ty + truckH}'
stroke='#333' stroke-width='2'/>");
            for (double m = 0; m <=
trailer.InnerMm.W; m += 500)
            {
                double y = ty + (m *
pxPerMm);

sb.AppendLine($"<line x1='{tx - 10}'
y1='{y.ToString("F1", ci)}' x2='{tx
- 20}' y2='{y.ToString("F1", ci)}'
stroke='#333' stroke-width='1'/>");

sb.AppendLine($"<text x='{tx - 25}'
y='{y.ToString("F1", ci)}' font-
family='Arial' font-size='10' text-
anchor='end' dominant-
baseline='middle' fill='#555'>{(m /
1000):F1}</text>");
            }

            sb.AppendLine($"<rect
x='{tx}' y='{ty}' width='{truckW}'
height='{truckH}' fill='url(#gr)'
stroke='#333' stroke-width='3'/>");

            if
(trailer.ForbiddenZones != null)
            {
                foreach (var z in
trailer.ForbiddenZones)
                {
                    double zx = tx +
(z.X * pxPerMm), zy = ty + (z.Z *
pxPerMm), zw = z.L * pxPerMm, zh =
z.W * pxPerMm;

sb.AppendLine($"<g><title>Forbidden:
{z.L}x{z.W}mm</title>");

sb.AppendLine($"<rect
x='{zx.ToString("F1", ci)}'
y='{zy.ToString("F1", ci)}'
width='{zw.ToString("F1", ci)}'
height='{zh.ToString("F1", ci)}'
fill='url(#haz)' stroke='black'
opacity='0.8'/>");

                    if (zw > 20 &&
zh > 20)
                    {
                        double cx =
zx + zw / 2, cy = zy + zh / 2;
```

```
                        string label
= "NO LOAD";
                        if (z.W <
800) label = "ZONE";
                        bool rotate
= zh > zw;
                        string tr =
rotate ? $"transform='rotate(-90
{cx.ToString("F1", ci)}
{cy.ToString("F1", ci)})'" : "";

sb.AppendLine($"<text
x='{cx.ToString("F1", ci)}'
y='{cy.ToString("F1", ci)}' {tr}
font-family='Arial' font-
weight='bold' font-size='14'
fill='white' stroke='black' stroke-
width='0.5' text-anchor='middle'
dominant-
baseline='middle'>{label}</text>");
                    }

sb.AppendLine("</g>");
                }
            }

            foreach (var s in
layerSpaces)
            {
                if (s.L < 10 || s.W
< 10) continue;
                var col = s.Y > 100
? VoidShelfColor : VoidFloorColor;
                string hex =
$"#{col.R:X2}{col.G:X2}{col.B:X2}";
                double sx = tx +
(s.X * pxPerMm), sy = ty + (s.Z *
pxPerMm), sw = s.L * pxPerMm, sh =
s.W * pxPerMm;

sb.AppendLine($"<g><title>Free:
{s.L:F0}x{s.W:F0}mm</title>");

sb.AppendLine($"<rect
x='{sx.ToString("F1", ci)}'
y='{sy.ToString("F1", ci)}'
width='{sw.ToString("F1", ci)}'
height='{sh.ToString("F1", ci)}'
fill='{hex}' fill-opacity='0.6'
stroke='#aaa' stroke-dasharray='2'
stroke-width='0.5'/>");
                if (s.L > 250 && s.W
> 250)
                {
                    string txt =
$"{s.L:F0}x{s.W:F0}";
                    var lay =
FitText(sw, sh, txt, 8, 14, false);
                    if (lay.Fits)
                    {
                        double cx =
sx + sw / 2, cy = sy + sh / 2;
                        string tr =
lay.Rotate ? $"transform='rotate(-90
{cx.ToString("F1", ci)}
{cy.ToString("F1", ci)})'" : "";

sb.AppendLine($"<text
x='{cx.ToString("F1", ci)}'
y='{cy.ToString("F1", ci)}' {tr}
font-family='Arial' font-
size='{lay.FontSize.ToString("F1",
ci)}' fill='#777' text-
anchor='middle' dominant-
baseline='middle' style='pointer-
events:none'>{txt}</text>");
                    }
                }

sb.AppendLine("</g>");
            }
```

```
            foreach (var p in
layerPlacements)
            {
                var b =
boxById[p.BoxId];
                bool swap =
(p.Rotation.Y % 180 != 0) ^
(p.Rotation.Z % 180 != 0);
                double dl = swap ?
b.SizeMm.W : b.SizeMm.L, dw = swap ?
b.SizeMm.L : b.SizeMm.W;
                double bx = tx +
(p.PositionMm.X * pxPerMm), by = ty
+ (p.PositionMm.Z * pxPerMm), bw =
dl * pxPerMm, bh = dw * pxPerMm;
                double t =
(b.WeightKg - minW) / rangeW;
                string f =
Interp(BoxLightColor, BoxHeavyColor,
t);



sb.AppendLine($"<g><title>#{p.Sequen
ceIndex}: {b.Id}</title>");


sb.AppendLine($"<rect
x='{bx.ToString("F1", ci)}'
y='{by.ToString("F1", ci)}'
width='{bw.ToString("F1", ci)}'
height='{bh.ToString("F1", ci)}'
fill='{f}' stroke='#222' rx='2'
ry='2' filter='url(#sh)'/>");



                string lbl =
$"{p.SequenceIndex}" + (b.Fragile ?
"!" : "");



                var lay =
FitText(bw, bh, lbl, 6, 32, true);
                if (lay.Fits)
                {
                    double cx = bx +
bw / 2, cy = by + bh / 2;
                    string tr =
lay.Rotate ? $"transform='rotate(-90
{cx.ToString("F1", ci)}
{cy.ToString("F1", ci)})'" : "";
                    string fw =
b.Fragile ? "bold" : "normal";


sb.AppendLine($"<text
x='{cx.ToString("F1", ci)}'
y='{cy.ToString("F1", ci)}' {tr}
font-family='Arial' font-
weight='{fw}' font-
size='{lay.FontSize.ToString("F1",
ci)}' fill='#222' text-
anchor='middle' dominant-
baseline='middle' style='pointer-
events:none'>{lbl}</text>");
                }


sb.AppendLine("</g>");
            }


            double footY = ty +
truckH + 30;



            sb.AppendLine($"<rect
x='{padL}' y='{footY}'
width='{truckW}' height='60'
fill='#f0f0f0' rx='5'
stroke='#ccc'/>");
            sb.AppendLine($"<text
x='{padL + 15}' y='{footY + 25}'
font-family='Arial' font-size='16'
font-weight='bold' fill='#333'>LAYER
HEIGHT: {layerMinY:F0}mm -
{layerMaxY:F0}mm</text>");
            sb.AppendLine($"<text
x='{padL + 15}' y='{footY + 50}'
```

```
font-family='Arial' font-size='14'
fill='#555'>Total Items:
{layerPlacements.Count} | Total Wgt:
{layerWeight:F1} kg</text>");


            double tblX = tx +
truckW + pad, tblY = ty;


            sb.AppendLine($"<rect
x='{tblX}' y='{tblY}'
width='{tableW}' height='{tHeadH}'
fill='#333' stroke='#333'/>");


            double col1 = tblX + 35,
col2 = tblX + 150, col3 = tblX +
260;

            sb.AppendLine($"<text
x='{tblX + 5}' y='{tblY + 20}' font-
family='Arial' font-size='12' font-
weight='bold'
fill='white'>#</text>");

            sb.AppendLine($"<text
x='{col1 + 5}' y='{tblY + 20}' font-
family='Arial' font-size='12' font-
weight='bold'
fill='white'>ID</text>");

            sb.AppendLine($"<text
x='{col2 + 5}' y='{tblY + 20}' font-
family='Arial' font-size='12' font-
weight='bold' fill='white'>Size
(mm)</text>");

            sb.AppendLine($"<text
x='{col3 + 5}' y='{tblY + 20}' font-
family='Arial' font-size='12' font-
weight='bold'
fill='white'>Kg</text>");


            double curY = tblY +
tHeadH, botY = tblY + tHeadH +
(layerPlacements.Count * rowH);


            sb.AppendLine($"<line
x1='{col1}' y1='{tblY}' x2='{col1}'
y2='{botY}' stroke='#777' stroke-
width='1'/>");

            sb.AppendLine($"<line
x1='{col2}' y1='{tblY}' x2='{col2}'
y2='{botY}' stroke='#777' stroke-
width='1'/>");

            sb.AppendLine($"<line
x1='{col3}' y1='{tblY}' x2='{col3}'
y2='{botY}' stroke='#777' stroke-
width='1'/>");


            foreach (var p in
layerPlacements)

            {

                var b =
boxById[p.BoxId];

                double t =
(b.WeightKg - minW) / rangeW; string
c = Interp(BoxLightColor,
BoxHeavyColor, t);


sb.AppendLine($"<rect x='{tblX}'
y='{curY}' width='{tableW}'
height='{rowH}' fill='{c}' fill-
opacity='0.2' stroke='#aaa' stroke-
width='1'/>");


                string idS =
b.Id.Length > 12 ? b.Id.Substring(0,
11) + "." : b.Id;

                string szS =
$"{b.SizeMm.L}x{b.SizeMm.W}x{b.SizeM
m.H}";


sb.AppendLine($"<text x='{tblX + 5}'
y='{curY + 15}' font-
family='Consolas' font-size='11'
fill='black'>{p.SequenceIndex,3}</te
xt>");


sb.AppendLine($"<text x='{col1 + 5}'
```

```
y='{curY + 15}' font-
family='Consolas' font-size='11'
fill='black'>{idS}</text>");


sb.AppendLine($"<text x='{col2 + 5}'
y='{curY + 15}' font-
family='Consolas' font-size='11'
fill='black'>{szS}</text>");


sb.AppendLine($"<text x='{col3 + 5}'
y='{curY + 15}' font-
family='Consolas' font-size='11'
fill='black'>{b.WeightKg}</text>");

                curY += rowH;

            }


            sb.AppendLine("</svg>");

            return sb.ToString();

        }



        private static string
Interp((int R, int G, int B) c1,
(int R, int G, int B) c2, double t)

        {

            t = Math.Clamp(t, 0, 1);
int r = (int)(c1.R + t * (c2.R -
c1.R)), g = (int)(c1.G + t * (c2.G -
c1.G)), b = (int)(c1.B + t * (c2.B -
c1.B)); return
$"#{r:X2}{g:X2}{b:X2}";

        }

        private struct TLR { public
bool Fits; public double FontSize;
public bool Rotate; }

        private static TLR
FitText(double w, double h, string
t, double min, double max, bool b)

        {

            double r = b ? 0.7 :
0.6; int l = t.Length;

            for (double s = max; s
>= min; s -= 1) if (l * s * r <= w *
0.95 && s <= h * 0.95) return new
TLR { Fits = true, FontSize = s,
Rotate = false };

            for (double s = max; s
>= min; s -= 1) if (l * s * r <= h *
0.95 && s <= w * 0.95) return new
TLR { Fits = true, FontSize = s,
Rotate = true };

            return new TLR { Fits =
false };

        }

    }

}
```

ДОДАТОК Е – Solver.cs

```
using System;
using System.Collections.Generic;
using System.Linq;


namespace Pack3D
{
    public static class Solver
    {
        private const double EPS = 1.0;


        public static SolveResult Solve(
            IReadOnlyList<Box> boxes, IReadOnlyList<Pallet> pallets, Trailer trailer,
            SolveOptions options, IReadOnlyList<FrozenPlacement>? frozen = null)
        {
            var sortedBoxes = boxes
                .Where(b => !options.FilterReadyOnly || b.Status == "Ready")
                .OrderByDescending(b => b.DeliveryStopIndex)
                .ThenByDescending(b => b.Priority)
                .ThenByDescending(b => b.WeightKg / (b.SizeMm.L * b.SizeMm.W * b.SizeMm.H))
                .ThenByDescending(b => b.SizeMm.L * b.SizeMm.W)
                .ToList();
            var placements = new List<Placement>();
            var unplacedBoxes = new List<string>();
            var allBoxesMap = boxes.ToDictionary(b => b.Id);


            var emptySpaces = new List<Rect3> {
                new Rect3(0, 0, 0, trailer.InnerMm.L, trailer.InnerMm.W, trailer.InnerMm.H, 0)
            };


            if (trailer.ForbiddenZones != null)
            {
                foreach (var zone in trailer.ForbiddenZones) ApplyForbiddenZone(emptySpaces, zone);
            }


            double currentPayloadKg = 0;
            double currentMomentX = 0;
            double currentMomentZ = 0;
            double targetCenterX = (trailer.AxleGeo.KingpinDistanceFromFrontMm + trailer.AxleGeo.TandemAxleDistanceFromFrontMm) / 2.0;
            double targetCenterZ = trailer.InnerMm.W / 2.0;
```

```
            int maxStopIndex =
sortedBoxes.Any() ?
sortedBoxes.Max(b =>
b.DeliveryStopIndex) : 1;

            double sectorLength =
maxStopIndex > 0 ? trailer.InnerMm.L
/ maxStopIndex : trailer.InnerMm.L;



            foreach (var box in
sortedBoxes)

            {

                var (curSteer,
curDrive, curTandem) =
CalculateAxleWeights(trailer,
currentPayloadKg, currentMomentX);

                double tandemPenalty
= (curTandem < 4000) ? (4000 -
curTandem) * 10.0 : 0;



                double currentCoGZ =
currentPayloadKg > 0 ?
currentMomentZ / currentPayloadKg :
targetCenterZ;

                double sideImbalance
= currentCoGZ - targetCenterZ;



                Rect3? bestSpace =
null;

                double bestScore =
double.MinValue;

                Dimensions bestDims
= box.SizeMm;

                (int X, int Y, int
Z) bestRot = (0, 0, 0);



                double minAllowedX =
(maxStopIndex > 1) ? (maxStopIndex -
box.DeliveryStopIndex) *
sectorLength : 0;



                var
possibleRotations =
GenerateRotations(box);



                for (int i = 0; i <
emptySpaces.Count; i++)

                {

                    Rect3 space =
emptySpaces[i];

                    if (space.X <
minAllowedX - 100) continue;



                    foreach (var rot
in possibleRotations)

                    {

                        if
(rot.Dims.L <= space.L + EPS &&
rot.Dims.W <= space.W + EPS &&
rot.Dims.H <= space.H + EPS)

                        {

                            if
(!IsStackingValid(box, space,
rot.Dims, placements, allBoxesMap))
continue;



                            double
score = 0;

                            score -=
((space.L - rot.Dims.L) + (space.W -
rot.Dims.W));

                            score -=
space.Y * 50.0;



                            double
boxCenterX = space.X + (rot.Dims.L /
2.0);

                            score -=
Math.Abs(boxCenterX - targetCenterX)
* 0.5;
```

```
                            double
boxCenterZ = space.Z + (rot.Dims.W /
2.0);
                            double
distFromCenterZ = boxCenterZ -
targetCenterZ;


                            if
(Math.Sign(sideImbalance) ==
Math.Sign(distFromCenterZ) &&
Math.Abs(sideImbalance) > 100)

score -= Math.Abs(distFromCenterZ) *
5.0;
                            else

score += Math.Abs(distFromCenterZ) *
1.0;


                            if
(tandemPenalty > 0) score -=
tandemPenalty * (1.5 - (boxCenterX /
trailer.InnerMm.L));


                            if
(score > bestScore)
                            {

bestScore = score;

bestSpace = space;

bestDims = rot.Dims;

bestRot = rot.RotFlags;
                            }
                        }
                    }
                }

                if (bestSpace ==
null)
                {

unplacedBoxes.Add(box.Id);
                    continue;
                }


                var pos = new
Vector3(bestSpace.X, bestSpace.Y,
bestSpace.Z);


                double actualLen =
((bestRot.Y % 180 != 0) ^ (bestRot.Z
% 180 != 0)) ? box.SizeMm.W :
box.SizeMm.L;
                double
boxCenterXFinal = pos.X + actualLen
/ 2.0;
                double boxMomentX =
box.WeightKg * boxCenterXFinal;


                var boxAxleContrib =
CalculateAxleWeights(trailer,
box.WeightKg, boxMomentX);


                var pl = new
Placement(
                    box.Id,
                    pos,
                    bestRot,
                    placements.Count
+ 1,

bestSpace.LayerIndex,
```

```
                    boxAxleContrib,
                    bestDims.L *
bestDims.W,
                    new
List<string>()
                );


                placements.Add(pl);

UpdateSpaces(emptySpaces, bestSpace,
bestDims);


                currentPayloadKg +=
box.WeightKg;
                double actualWid =
((bestRot.Y % 180 != 0) ^ (bestRot.Z
% 180 != 0)) ? box.SizeMm.L :
box.SizeMm.W;
                currentMomentX +=
boxMomentX;
                currentMomentZ +=
box.WeightKg * (pos.Z + actualWid /
2.0);
            }


            return
GenerateResult(trailer, placements,
unplacedBoxes, boxes,
currentPayloadKg, currentMomentX,
currentMomentZ, emptySpaces);
        }


        private static void
ApplyForbiddenZone(List<Rect3>
spaces, Rect3 zone)
        {
            var newSpaces = new
List<Rect3>();
            foreach (var space in
spaces)
            {
                if (Intersect(space,
zone))
newSpaces.AddRange(CutSpace(space,
zone));
                else
newSpaces.Add(space);
            }
            spaces.Clear();

spaces.AddRange(newSpaces);
        }


        private static bool
Intersect(Rect3 s, Rect3 z)
        {
            return (s.X < z.X + z.L
- EPS && s.X + s.L > z.X + EPS) &&
                   (s.Y < z.Y + z.H
- EPS && s.Y + s.H > z.Y + EPS) &&
                   (s.Z < z.Z + z.W
- EPS && s.Z + s.W > z.Z + EPS);
        }


        private static List<Rect3>
CutSpace(Rect3 s, Rect3 z)
        {
            var res = new
List<Rect3>();
            double ix1 =
Math.Max(s.X, z.X); double ix2 =
Math.Min(s.X + s.L, z.X + z.L);
            double iy1 =
Math.Max(s.Y, z.Y); double iy2 =
Math.Min(s.Y + s.H, z.Y + z.H);
```

```
            double iz1 =
Math.Max(s.Z, z.Z); double iz2 =
Math.Min(s.Z + s.W, z.Z + z.W);



            if (ix1 > s.X + EPS)
res.Add(new Rect3(s.X, s.Y, s.Z, ix1
- s.X, s.W, s.H, s.LayerIndex));

            if (ix2 < s.X + s.L -
EPS) res.Add(new Rect3(ix2, s.Y,
s.Z, (s.X + s.L) - ix2, s.W, s.H,
s.LayerIndex));

            if (iy1 > s.Y + EPS)
res.Add(new Rect3(ix1, s.Y, s.Z, ix2
- ix1, s.W, iy1 - s.Y,
s.LayerIndex));

            if (iy2 < s.Y + s.H -
EPS) res.Add(new Rect3(ix1, iy2,
s.Z, ix2 - ix1, s.W, (s.Y + s.H) -
iy2, s.LayerIndex));

            if (iz1 > s.Z + EPS)
res.Add(new Rect3(ix1, iy1, s.Z, ix2
- ix1, iz1 - s.Z, iy2 - iy1,
s.LayerIndex));

            if (iz2 < s.Z + s.W -
EPS) res.Add(new Rect3(ix1, iy1,
iz2, ix2 - ix1, (s.Z + s.W) - iz2,
iy2 - iy1, s.LayerIndex));

            return res;

        }



        private static void
UpdateSpaces(List<Rect3> spaces,
Rect3 used, Dimensions dim)

        {

            spaces.Remove(used);

            if (used.L - dim.L >
EPS) spaces.Add(new Rect3(used.X +
dim.L, used.Y, used.Z, used.L -
dim.L, used.W, used.H,
used.LayerIndex));

            if (used.W - dim.W >
EPS) spaces.Add(new Rect3(used.X,
used.Y, used.Z + dim.W, dim.L,
used.W - dim.W, used.H,
used.LayerIndex));

            if (used.H - dim.H >
EPS) spaces.Add(new Rect3(used.X,
used.Y + dim.H, used.Z, dim.L,
dim.W, used.H - dim.H,
used.LayerIndex + 1));

        }



        private static bool
IsStackingValid(Box cur, Rect3
space, Dimensions dims,
List<Placement> placements,
Dictionary<string, Box> all)

        {

            if (space.Y < 1.0)
return true;

            double myX2 = space.X +
dims.L; double myZ2 = space.Z +
dims.W;

            foreach (var p in
placements)

            {

                var bBelow =
all[p.BoxId];

                double hBelow =
(p.Rotation.X == 90) ?
((p.Rotation.Z == 90) ?
bBelow.SizeMm.L : bBelow.SizeMm.W) :
bBelow.SizeMm.H;

                if
(Math.Abs((p.PositionMm.Y + hBelow)
- space.Y) > 5.0) continue;

                double lBelow =
((p.Rotation.Y % 180 != 0) ^
(p.Rotation.Z % 180 != 0)) ?
bBelow.SizeMm.W : bBelow.SizeMm.L;

                double wBelow =
((p.Rotation.Y % 180 != 0) ^
(p.Rotation.Z % 180 != 0)) ?
bBelow.SizeMm.L : bBelow.SizeMm.W;
```

```
                if (space.X <
p.PositionMm.X + lBelow - EPS &&
myX2 > p.PositionMm.X + EPS &&

                    space.Z <
p.PositionMm.Z + wBelow - EPS &&
myZ2 > p.PositionMm.Z + EPS)

                {

                    if
(bBelow.Fragile) return false;

                    if (cur.WeightKg
> bBelow.MaxStackOnTopKg) return
false;

                }

            }

            return true;

        }


        private static
List<(Dimensions Dims, (int, int,
int) RotFlags)>
GenerateRotations(Box b)

        {

            var l = new
List<(Dimensions, (int, int,
int))>();

            l.Add((b.SizeMm, (0, 0,
0)));

            if
(b.AllowedRotations.HasFlag(Rotation
Mask.Z) ||
b.AllowedRotations.HasFlag(RotationM
ask.All))

                l.Add((new
Dimensions(b.SizeMm.W, b.SizeMm.L,
b.SizeMm.H), (0, 0, 90)));

            return l;

        }
```

```
        private static SolveResult
GenerateResult(Trailer t,
List<Placement> p, List<string> u,
IReadOnlyList<Box> all, double w,
double mx, double mz, List<Rect3> f)

        {

            var al =
CalculateAxleWeights(t, w, mx);

            var phys = new
LoadPhysics(new Vector3(w > 0 ? mx /
w : 0, 0, w > 0 ? mz / w : 0), mx,
0, mz);

            double currentVolume =
p.Sum(pl => { var b = all.First(x =>
x.Id == pl.BoxId); return b.SizeMm.L
* b.SizeMm.W * b.SizeMm.H; });

            double trailerVolume =
t.InnerMm.L * t.InnerMm.W *
t.InnerMm.H;

            return new
SolveResult(p, u, f, new
Metrics(Math.Round((currentVolume /
trailerVolume) * 100, 2),
Math.Round((w / t.MaxPayloadKg) *
100, 2), w, al, phys, p.Count,
u.Count), new List<string>(), null);

        }


        private static (double
Steer, double Drive, double Tandem)
CalculateAxleWeights(Trailer t,
double w, double m)

        {

            if (w <= 0) return (0,
0, 0);

            double cog = m / w;

            double kp =
t.AxleGeo.KingpinDistanceFromFrontMm
;

            double tand =
t.AxleGeo.TandemAxleDistanceFromFron
tMm;
```

```
            double tl = w * ((cog -
kp) / (tand - kp));


            double kl = w - tl;


            double driveKp =
Math.Abs(t.AxleGeo.DriveAxleDistance
FromFrontMm - kp);
            double sl = kl *
(driveKp /
t.AxleGeo.TractorWheelbaseMm);


            return (Math.Round(sl,
1), Math.Round(kl - sl, 1),
Math.Round(tl, 1));
        }
    }
}
```

ДОДАТОК Є – Код Program.cs

```
using Pack3D;
using Pack3D.DataGen;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;

namespace Pack3D.Client
{
    class Program
    {
        static string _reportsFolder =
Path.Combine(Directory.GetCurrentDirec
tory(), "_Pack3D_Reports");

        static void Main(string[]
args)
        {
            Console.OutputEncoding =
System.Text.Encoding.UTF8;
            if
(!Directory.Exists(_reportsFolder))
Directory.CreateDirectory(_reportsFold
er);

            Console.WriteLine("===
Pack3D: Advanced Logistics Simulation
(vFinal) ===");
            Console.WriteLine($"[INFO]
Output: {_reportsFolder}");

            var scenarios =
ScenarioGenerator.GetScenarios();
            foreach (var scenario in
scenarios) Run(scenario);


Console.WriteLine($"\nDONE. Check
{_reportsFolder}");
        }

        static void
Run(SimulationScenario s)
        {
            Console.WriteLine($"\n---
SCENARIO: {s.Name} ---");
            Stopwatch sw =
Stopwatch.StartNew();
            var res =
Solver.Solve(s.Boxes, new
List<Pallet>(), s.Trailer, new
SolveOptions());
            sw.Stop();
            Console.WriteLine($"Time:
{sw.ElapsedMilliseconds}ms | Placed:
{res.Metrics.PlacedCount}/{s.Boxes.Cou
nt}");
            Console.WriteLine($"Util:
{res.Metrics.VolumeUtilizationPct:F1}%
Vol |
{res.Metrics.WeightUtilizationPct:F1}%
Wgt");

            var phys =
res.Metrics.Physics;
            if
(Math.Abs(phys.CenterOfGravityMm.Z -
s.Trailer.InnerMm.W / 2) > 200)

Console.WriteLine("WARNING:
Significant Side Imbalance!");


Console.WriteLine(UnloadingAnalyzer.An
alyzeUnloading(res, s.Boxes,
s.Trailer));

            string folder =
Path.Combine(_reportsFolder,
string.Join("_",
s.Name.Split(Path.GetInvalidFileNameCh
ars())).Trim());

Directory.CreateDirectory(folder);

            Exporter.SaveJson(res,
Path.Combine(folder, "plan.json"));
            Exporter.SaveCsv(res,
s.Boxes, Path.Combine(folder,
"manifest.csv"));

            var boxMap =
s.Boxes.ToDictionary(b => b.Id);
            var layers =
res.Placements.Select(p =>
p.LayerIndex).Distinct().OrderBy(l =>
l);

            foreach (var l in layers)
            {
                string svg =
Visualizer.RenderLayerSvg(res.Placemen
ts, res.FreeSpaces, s.Trailer,
res.Metrics, boxMap, 0.05, l);

File.WriteAllText(Path.Combine(folder,
$"Layer_{l}.svg"), svg);
            }
            Console.WriteLine($"->
Saved SVG/JSON/CSV to /{new
DirectoryInfo(folder).Name}");
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема: «Проєктування автоматизованої системи оптимального розміщення вантажу при транспортуванні з оцінкою ефективності за просторовими, ваговими та маршрутними критеріями»

Обсяг пояснювальної записки: 69 аркушів.

Кількість рисунків: 28 шт.

Кількість таблиць: 6 шт.

Графічний матеріал (креслення, схеми):

КРМ. АКСм - 11.00.00.001 – Дерево проекту. Схема функціональна

КРМ. АКСм - 11.00.00.002 – UML діаграма класів. Схема функціональна.

КРМ. АКСм - 11.00.00.003 – Конфігурація страртової точки входу в ПЗ. Вигляд загальний.

КРМ. АКСм - 11.00.00.004 – Схема UseCase Pack3D. Схема функціональна

„____”_____2025р.

Підпис студента Каралаша М. Р