

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
GRADUATION THESIS**

**Визуализация алгоритмов оптимизации размещения виртуальных машин в
вычислительных центрах**

Обучающийся / Student Митрофанов Егор Юрьевич

Факультет/институт/кластер/ Faculty/Institute/Cluster факультет программной инженерии и компьютерной техники

Группа/Group P34101

Направление подготовки/ Subject area 09.03.04 Программная инженерия

Образовательная программа / Educational program Системное и прикладное программное обеспечение 2019

Язык реализации ОП / Language of the educational program Русский

Статус ОП / Status of educational program

Квалификация/ Degree level Бакалавр

Руководитель ВКР/ Thesis supervisor Перл Иван Андреевич, кандидат технических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")

Консультант/ Consultant Перл Ольга Вячеславовна, ФПИ и КТ, ассистент (квалификационная категория "ассистент"), неосн по совм.

Обучающийся/Student

Документ подписан	
Митрофанов Егор Юрьевич	
15.05.2023	

(эл. подпись/ signature)

Митрофанов
Егор Юрьевич

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Перл Иван Андреевич	
15.05.2023	

(эл. подпись/ signature)

Перл Иван
Андреевич

(Фамилия И.О./ name
and surname)

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ /
OBJECTIVES FOR A GRADUATION THESIS

Обучающийся / Student Митрофанов Егор Юрьевич
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет программной инженерии и компьютерной техники
Группа/Group P34101
Направление подготовки/ Subject area 09.03.04 Программная инженерия
Образовательная программа / Educational program Системное и прикладное программное обеспечение 2019
Язык реализации ОП / Language of the educational program Русский
Статус ОП / Status of educational program
Квалификация/ Degree level Бакалавр
Тема ВКР/ Thesis topic Визуализация алгоритмов оптимизации размещения виртуальных машин в вычислительных центрах
Руководитель ВКР/ Thesis supervisor Перл Иван Андреевич, кандидат технических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")
Консультант/ Consultant Перл Ольга Вячеславовна, ФПИ и КТ, ассистент (квалификационная категория "ассистент"), неосн по совм.

Основные вопросы, подлежащие разработке / Key issues to be analyzed

Цель работы:

Исследование возможных подходов к визуализации принципов работы нескольких алгоритмов размещения виртуальных машин и их визуализация с использованием полученных данных.

Задачи работы:

Рассмотреть основные виды алгоритмов размещения виртуальных машин, в том числе эвристические алгоритмы и алгоритмы математической оптимизации. Также рассмотреть различные способы визуализации данных алгоритмов, представить примеры визуализации алгоритмов и проанализировать преимущества и недостатки каждого метода. Дать рекомендации по выбору наиболее подходящего метода визуализации в зависимости от задачи и целей исследования и на их основе визуализировать принципы работы рассмотренных алгоритмов.

Исходные данные к работе:

Описания принципов работы и применения наиболее распространенных алгоритмов размещения виртуальных машин.

Содержание выпускной квалификационной работы:

Анализ проблемы и решаемой задачи.

Анализ инструментов визуализации для решения поставленной задачи.

Реализовать программный комплекс для решения поставленной задачи на основе выбранных методов, подходов и инструментов.

Произвести тестирование полученного решения.

Провести анализ полученного решения.

Форма представления материалов ВКР / Format(s) of thesis materials:

Форма представления результатов ВКР: Текст работы, презентация

Форма представления приложений: Программный код

Дата выдачи задания / Assignment issued on: 30.09.2022

Срок представления готовой ВКР / Deadline for final edition of the thesis 22.05.2023

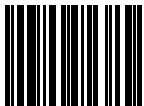
Характеристика темы ВКР / Description of thesis subject (topic)

Тема в области фундаментальных исследований / Subject of fundamental research: да / yes

Тема в области прикладных исследований / Subject of applied research: нет / not

СОГЛАСОВАНО / AGREED:

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Перл Иван Андреевич	
01.05.2023	

(эл. подпись)

Перл Иван
Андреевич


Задание принял к
исполнению/ Objectives
assumed BY

Документ подписан	
Митрофанов Егор Юрьевич	
10.05.2023	

(эл. подпись)

Митрофанов
Егор Юрьевич

Руководитель ОП/ Head
of educational program

Документ подписан	
Дергачев Андрей Михайлович	
26.05.2023	

(эл. подпись)

Дергачев
Андрей
Михайлович

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**АННОТАЦИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
SUMMARY OF A GRADUATION THESIS**

Обучающийся / Student Митрофанов Егор Юрьевич
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет программной инженерии и компьютерной техники
Группа/Group P34101
Направление подготовки/ Subject area 09.03.04 Программная инженерия
Образовательная программа / Educational program Системное и прикладное программное обеспечение 2019
Язык реализации ОП / Language of the educational program Русский
Статус ОП / Status of educational program
Квалификация/ Degree level Бакалавр
Тема ВКР/ Thesis topic Визуализация алгоритмов оптимизации размещения виртуальных машин в вычислительных центрах
Руководитель ВКР/ Thesis supervisor Перл Иван Андреевич, кандидат технических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")
Консультант/ Consultant Перл Ольга Вячеславовна, ФПИ и КТ, ассистент (квалификационная категория "ассистент"), неосн по совм.

**ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
DESCRIPTION OF THE GRADUATION THESIS**

Цель исследования / Research goal

Оптимальное размещение виртуальных машин в центрах обработки данных - одна из ключевых проблем для поставщиков и производителей оборудования. Проблема размещения виртуальных машин является NP – сложной задачей и не имеет единственного набора наиболее эффективных решений. Цель данной работы - исследование возможных подходов к визуализации принципов работы нескольких подобных алгоритмов и их визуализация с использованием полученных в исследовании данных. Качественная и корректная визуализация алгоритмов даст наглядное представление об их работе, что поможет глубже понять происходящие процессы при выборе и сравнительном анализе этих решений, что в свою очередь способствует снижению порога входа для исследователей и инженеров в предметной области.

Задачи, решаемые в ВКР / Research tasks

Рассмотреть основные виды алгоритмов размещения виртуальных машин, в том числе эвристические алгоритмы и алгоритмы математической оптимизации. Также рассмотреть различные способы визуализации данных алгоритмов, представить примеры визуализации алгоритмов и проанализировать преимущества и недостатки каждого метода. Дать

рекомендации по выбору наиболее подходящего метода визуализации в зависимости от задачи и целей исследования и на их основе визуализировать принципы работы рассмотренных алгоритмов. Также необходимо рассмотреть множество инструментов визуализации – различные библиотеки и open-source приложения, JS – фреймворки для двумерной визуализации, а также полноценные игровые движки для трехмерной визуализации. Провести анализ популярных VMP алгоритмов с целью выбора наилучшего подхода к визуализации. На основании полученных данных разработать прототип и полноценное приложение для 3D визуализации принципов работы VMP алгоритмов, после чего провести тестирование и анализ полученного решения, выявить преимущества и недостатки и предложить пути развития.

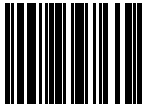
Краткая характеристика полученных результатов / Short summary of results/findings

Исследование методов визуализации алгоритмов размещения показало, что визуализация может существенно улучшить понимание и анализ процесса размещения виртуальных машин в вычислительных центрах. Рассмотрены различные инструменты для решения поставленной задачи, а также сами методы визуализации, включая графические и инфографические, и представлены примеры их применения для конкретных алгоритмов размещения виртуальных машин. Кроме того, проанализированы несколько наиболее популярных оптимизационных алгоритмов, для каждого из которых предложен оптимальный подход к визуализации его работы. Проведено сравнение алгоритмов с точки зрения выбора инструментов визуализации, предложены наиболее продуктивные подходы к выбору метода визуализации. Сделан вывод о необходимости качественной динамической визуализации, так как такой подход существенно влияет на понимание принципов работы ранее описанных процессов. На основании полученных ранее результатов исследования разработано полноценное трехмерное приложение для визуализации принципов работы VMP алгоритмов. Проведено тестирование и оценка полученного решения по выявленным критериям, в том числе интерактивности, качеству визуализации и также качеству реализации оптимизационных алгоритмов для решения задачи VMP. В заключительном анализе разработанного приложения были описаны недостатки и преимущества текущей реализации, а также предложены пути развития и улучшения.

Дополнительные сведения / Additional information

На Конгрессе Молодых Ученых весной 2023 по теме выпускной работы сделано выступление с победой в номинации "Лучший доклад молодого ученого". Также в процессе написания находится статья для Scopus

Обучающийся/Student

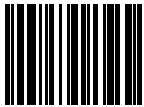
Документ подписан	
Митрофанов Егор Юрьевич	
15.05.2023	

(эл. подпись/ signature)

Митрофанов
Егор Юрьевич

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Перл Иван Андреевич	
15.05.2023	

(эл. подпись/ signature)

Перл Иван
Андреевич

(Фамилия И.О./ name)

Содержание

СПИСОК СОКРАЩЕНИЙ И СЛОВАРЬ ТЕРМИНОВ	8
ВВЕДЕНИЕ.....	10
1.1. Проблема размещения виртуальных машин	10
1.2. Важность визуализации данных	12
1.3. Цель работы	12
1.4. Заданные условия	13
1.5. Инструменты визуализации	13
1. Анализ VMP алгоритмов с точки зрения визуализации.....	17
1.1. Важность визуализации	17
1.2. Разнообразие VMP алгоритмов.....	18
1.3. Простейшие полиномиальные алгоритмы.....	20
1.4. MBFD – алгоритм.....	22
1.5. Алгоритм имитации отжига	23
1.6. Генетические алгоритмы	26
1.7. Алгоритмы на основе прогнозирования	31
1.8. Обобщение подходов к визуализации алгоритмов.....	32
2. Разработка визуализации алгоритмов VMP.....	35
2.2. Выбор технологий для реализации.....	35
2.3. Описание UI и UX	36
2.4. Реализация алгоритмов на языке программирования	37
2.5. Создание 3D – моделей и окружения.....	38
2.6. Графическое программирование и пост-процессинг	41
3. Тестирование разработанного приложения	44
3.1. Подход к тестированию визуализации.....	44
3.2. Тестирование алгоритмов.....	45
3.2.1. Анализ FF и BF	45
3.2.2. Анализ SA алгоритма.....	46
3.2.3. Анализ генетического алгоритма	47
3.3. Удобство использования	48
3.3.1. Пользовательское тестирование	48

3.3.2. Финальная оценка	49
4. Анализ разработанной визуализации	51
4.1. Анализ результатов тестирования	51
4.2. Преимущества трехмерной визуализации	51
4.3. Недостатки трехмерной визуализации.....	52
4.4. Способы развития и улучшения	53
ЗАКЛЮЧЕНИЕ	55
1.1. Выполнение поставленных задач	55
1.2. Достижение цели работы.....	55
1.3. Практическая значимость.....	56
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ	58
ПРИЛОЖЕНИЕ А	60

СПИСОК СОКРАЩЕНИЙ И СЛОВАРЬ ТЕРМИНОВ

VMP – Virtual Machine Placement, размещение виртуальных машин

VM – Виртуальная машина

QoS – Quality of Service, качество обслуживания

FF, BF – First fit, Best fit, базовые алгоритмы решения Bin package problem

SA – Simulated annealing, алгоритм имитации отжига

GA – Генетический оптимизационный алгоритм

ЦОД - Центр Обработки Данных

IAAS - Infrastructure as a Service (инфраструктура как сервис)

VM Placement - Размещение виртуальных машин

VM Provisioning - Создание виртуальных машин

VM Migration - Миграция виртуальных машин

Hypervisor - Гипервизор, программное обеспечение, которое управляет виртуальными машинами и ресурсами физического сервера

Host - Физический сервер, который используется для запуска виртуальных машин

Cluster - Группа связанных физических серверов, которые используются для хранения и обработки данных в ЦОД

SLA - Service Level Agreement (уровень обслуживания), договоренности между провайдером облачных услуг и клиентом относительно гарантий доступности, производительности и надежности услуг

VM Consolidation - Консолидация виртуальных машин, процесс объединения нескольких VM на одном хосте для оптимизации использования ресурсов

VM Sprawl - Распыление виртуальных машин, процесс создания большого количества VM без учета ресурсов, что может привести к ненужным затратам и сложностям в управлении

Resource Utilization - Использование ресурсов, процентное соотношение используемых ресурсов к доступным ресурсам

Over-Provisioning - Превышение выделенных ресурсов, процесс выделения больше ресурсов, чем необходимо, чтобы избежать проблем с производительностью

API - Application Programming Interface (интерфейс программирования приложений)

Machine Learning - Машинное обучение, подход к анализу данных, который позволяет компьютерным системам автоматически учиться и улучшаться без явного программирования

Data Visualization - Визуализация данных, процесс отображения информации в графической форме, который помогает лучше понимать и анализировать данные

SMAA - Техника сглаживания краев в компьютерной графике, которая позволяет уменьшить эффект ступенчатости (Aliasing) на краях объектов в сцене.

Bloom - Эффект визуального освещения, который используется для создания ярких объектов в сцене.

Ambient Occlusion - Техника, которая используется для создания более реалистичных теней и освещения в сценах.

Визуализация алгоритмов оптимизации размещения виртуальных машин в вычислительных центрах

ВВЕДЕНИЕ

1.1. Проблема размещения виртуальных машин

Вычислительные центры в основном используют для своей работы технологии виртуализации. Такой подход дает массу преимуществ, таких как масштабируемость, экономия затрат, мобильность данных и облегченный и комфортный менеджмент. Оптимальное размещение виртуальных машин в центрах обработки данных - одна из ключевых проблем для поставщиков и производителей подобного оборудования, ведь с помощью относительно несложных расчетов можно сэкономить до 30% (А. Beloglazov, 2012) серверных ресурсов.

Среди особенностей, которые нужно учесть, при распределении ВМ по физическим машинам в центрах обработки данных, можно выделить следующие (P. Svärd, 2015):

1. Энергозатраты на поддержание работы физических машин

Миграция нескольких ВМ на один физический кластер и снижение общего количества физических серверных стенов значительно сказывается на энергопотреблении и денежных расходах. (Hariharasudhan Viswanathan, 2012) Кроме того, используемое оборудование может различаться, что означает также и разное энергопотребление и ресурсы.

2. Quality of Service – качество обслуживания и управление приоритетами для различных классов трафика

Неоптимальное размещение ВМ влияет на многие параметры такие, как стоимость обслуживания, производительность, скорость соединения, что в совокупности оказывает значительное влияние на пользователей самих сервисов.

3. Вычислительная эффективность ВМ

Если несколько вычислительных систем, выполняющих требовательные операции будут находиться на одной физической машине, может происходить конфликт доступа к ресурсам, что приведет к снижению производительности (Tordsson., 2014).

4. Стоимость оборудования

Уменьшение количества используемых физических машин напрямую влияет на общую стоимость центра обработки данных и его обслуживание. Но непосредственное уменьшение этого числа не может являться единственной целью, так как падение производительности, отказоустойчивости или энергопотребления может в теории привести к большим капиталовложениям.

5. Отказоустойчивость оборудования и сложность его замены

Необходимо учитывать непосредственно физическое оборудование. Некоторые машины могут быть менее отказоустойчивыми, чем другие, что означает необходимость брать это во внимание при распределении ВМ, приоритизируя ВМ и сами физические машины.

6. Миграция ВМ и ее сложность в конкретных ситуациях

Миграция виртуальной машины может означать ее частичную недоступность в определенный промежуток времени (Strunk., 2012). Это, в свою очередь, влечет за собой дополнительные расходы из-за неработоспособности определенного сервиса. Поэтому, минимизация количества миграций – так же немаловажный пункт.

Стоит отметить, что можно выделить два типа задач размещения виртуальных машин:

1. Статическая

Есть заранее подготовленный набор ВМ, у которых определены все требуемые параметры. Задача сводится к поиску оптимального расположения всех машин.

2. Динамическая

На очередь размещения постоянно поступают новые ВМ, а старые могут выключаться. Кроме того, некоторые параметры, определяющие виртуальную машину, могут не являться константами и меняются со временем, что влечет за собой дополнительную потребность в постоянном пересмотре расстановок ВМ. В таком случае размещение ВМ происходит по потоку заявок, и необходимо по новым параметрам за короткое время определить, как будет происходить установка новой машины или миграция существующей.

Проблема размещения виртуальных машин - пример известной задачи «Bin paskage problem», которая является NP-полной задачей. Единственного набора наиболее эффективных решений нет и на практике используют суб-оптимальные алгоритмы (Mikhailyuk, 2011). Как продемонстрировано ранее,

количество особенностей, которые нужно учесть, достаточно велико, поэтому в различных ситуациях эффективными показывают себя разные алгоритмы, число которых на данный момент оценивается более сотни. (А. Beloglazov, 2012) Исходя из этих особенностей составляется целевая функция оптимизации, которую также называют функцией стоимости. В нашем случае функция стоимости выражает степень соответствия решения заданному критерию оптимальности – приближению к максимуму (минимуму) функции стоимости. Цель оптимизации состоит в том, чтобы минимизировать (или максимизировать) значение функции стоимости путем нахождения оптимального решения.

Также стоит отметить, что для предложенных эвристик нет доказанной временной сложности, а некоторые алгоритмы могут зависеть от случайных значений и иметь сильный разброс во времени выполнения (Mallawaarachchi, n.d.), что значительно усложняет понимание принципов работы.

1.2. Важность визуализации данных

Визуализация данных и алгоритмов имеет критическое значение для анализа данных и принятия решений на их основе. Качественная визуализация позволяет легко интерпретировать даже сложные для понимания концепции и процессы, принимать взвешенные решения на ее основе.

Визуализация не просто содержит некоторую информацию, но также качественно повышает эффективность ее восприятия за счет наглядности и привлечения внимания к наиболее важным моментам.

Качественная и корректная визуализация алгоритмов даст наглядное представление об их работе, что поможет упростить их понимание при выборе и сравнительном анализе этих решений, что в свою очередь способствует снижению порога входа для исследователей и инженеров в предметной области.

1.3. Цель работы

Цель выпускной квалификационной работы – наглядно визуализировать принцип работы нескольких наиболее распространенных алгоритмов.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Выбрать несколько наиболее распространённых алгоритмов размещения виртуальных машин, относящихся к разным типам. (Dabiah Ahmed Alboaneen, 2016)

2. Сравнить методы визуализации алгоритмов размещения виртуальных машин.
3. Сделать вывод о применимости метода визуализации для других алгоритмов.
4. Использовать данные из исследований для разработки визуализации, предварительно создав прототип.
5. Провести тестирование и анализ разработанного приложения, выявить преимущества и недостатки, дать рекомендации по улучшению.

1.4. Заданные условия

В рамках данной работы будет создана система визуализации алгоритмов для платформы VMP Cloud. Платформа VMP Cloud предоставляет исследователям возможность моделирования алгоритмов размещения виртуальных машин при помощи широко используемого инструмента CloudSim.

1.5. Инструменты визуализации

На сегодняшний день существует широкий набор средств визуализации информации. Для 2D визуализации и построения диаграмм и графиков наиболее популярны следующие:

7. Plotly – Приложение может выполнять анализ статистических данных при помощи Python или JavaScript и интегрируемо с Jupiter и Excel.
8. Raw – open-source приложение, которое можно настроить индивидуально для любых подходящих форматов и наборов данных.

Кроме того, существуют и JS плагины для двумерной визуализации:

- ZingChart – Многофункциональное API, позволяющее создавать интерактивные диаграммы с помощью HTML5 или Flash.
- Dygraphs – Аналогичный функционал с некоторыми ограничениями.

На Рисунке 1 представлена визуализация решения задачи о рюкзаке.

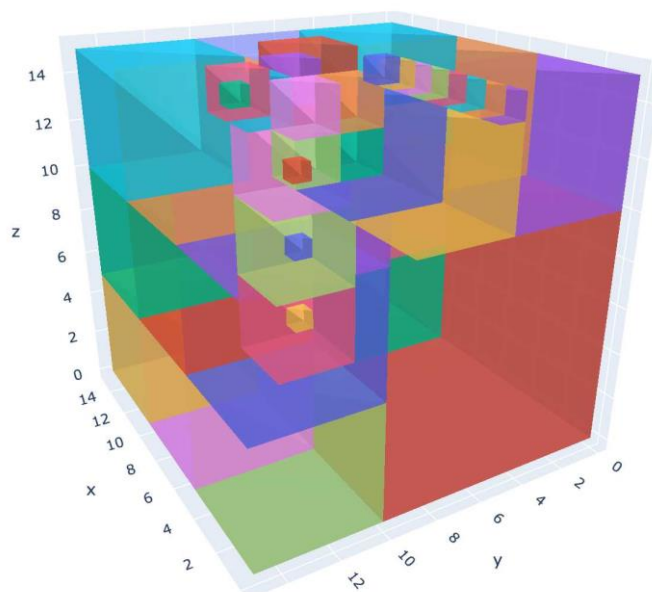


Рисунок 1, Кадр из анимации решения bin package problem

Для решения задачи визуализации алгоритмов VMP статичная двумерная визуализация не полностью удовлетворяет все потребности. Диаграммы и графики необходимы для представления статистики, собранных данных и сравнительного анализа вышеперечисленного. Тем не менее основная задача – визуализировать алгоритмы и принцип их работы.

Задача размещения виртуальных машин чаще всего сводится к трёхмерной упаковке, так как при расположении используются 3 вида параметров виртуальной машины (SLA: память, процессор и оперативная память). Кроме того, VMP алгоритмы бывают принципиально разными (Fikru Feleke Moges, 2019), что будет рассмотрено в следующей главе. Они могут включать различный набор входных данных, и поэтому не все из них могут быть наглядно показаны в 2D представлении, как, к примеру FF и BF (см. Рисунок 1). Исходя из этого стоит рассмотреть средства 3D визуализации. Для трехмерного представления также существуют плагины и библиотеки JS:

- Three.js – самая распространенная библиотека с большим комьюнити разработчиков. Есть глубокая настройка графики, эффектов и постобработки.
- Cesium – open-source библиотека, преимущественно используемая для визуализации карт.
- Deck.gl – WebGL2 фреймворк, аналог three.js со сравнимым функционалом на спецификации OpenGL.

На Рисунке 3 представлен скриншот динамической визуализации алгоритма сборки кубика-рубика, созданной с помощью Фреймворка Three.js

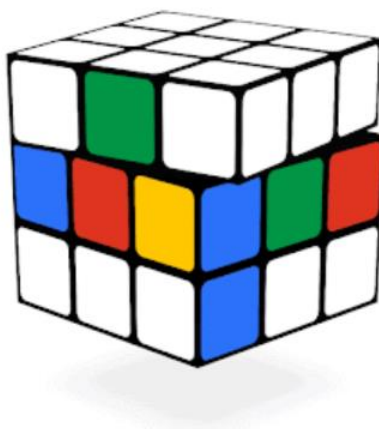


Рисунок 2, Визуализация алгоритма сборки кубика-рубика на Three.js (Автор - Forest Agostinelli)

Кроме библиотек и JS фреймворков необходимо рассмотреть полноценные графические движки. Проанализируем два самых популярных, так как для них существуют проработанные документации, поддержка сообщества и бесплатная образовательная версия.

- Unity3D – среда разработки игр и графики, позволяющая создавать мультиплатформенные приложения.
- Unreal Engine – игровой движок, направленный на создание масштабных игр и компьютерной графики.



Рисунок 3, Пример компьютерной визуализации в UE 4

Игровые движки в первую очередь позволят создавать приложение для любых платформ, включая Web – версию. Кроме того, они дают возможность использовать любой стек технологий и интегрироваться с разными сторонними сервисами. Также стоит отметить, что для двух вышеперечисленных технологий существует значительная база знаний от разработчиков и сообщества пользователей.

На Рисунке 4 представлен процесс визуализации рендера автомобиля.

Для визуализации алгоритмов VMP лучше всего подойдет Unity Engine. Он полностью решает все задачи, необходимые для данной работы, такие как качественная анимация, работа с 3D моделями и физикой, интеграция со сторонними сервисами (Rutenbar) и получение данных из внешнего источника. UE в свою очередь при разработке требует больше трудозатрат, но и предоставляет более широкий функционал, чем необходим для решения поставленной задачи. К примеру, более проработанная физика и рендер, трассировка лучей, постобработка и более качественные системы частиц.

1. Анализ VMP алгоритмов с точки зрения визуализации

В первой главе представлен анализ основных алгоритмов размещения виртуальных машин. В качестве изначального набора рассматриваются алгоритмы решения задачи «Упаковки рюкзака», а также решения, представленные на платформах CloudSim и OpenStack - платформах для моделирования и имитации инфраструктур и сервисов облачных вычислений. Конкретный выбор алгоритмов для визуализации обоснован их распространенностью и использованием в реальных условиях.

В рамках этой главы будем рассматривать исходные данные ВМ, а именно размер, ресурс и т. д. в качестве «Предмета» с некоторым объемом. Физические машины, на которых запускаются ВМ, рассматриваем как контейнеры для упаковки предметов, имеющие некоторый запас свободного пространства / объема.

Начнем с того, почему визуализация важна в целом и для решаемой задачи VMP в частности.

1.1. Важность визуализации

Визуализация имеет важное значение в различных областях, включая информационные технологии, науку, образование, инженерию и многие другие. Приведем несколько причин, почему это так: (Waskom)

1. Легкость восприятия информации

Визуализация позволяет представить данные в более понятном и доступном виде. С помощью графиков, диаграмм, карт, схем и других типов инфографики сложная информация легче интерпретируется и запоминается.

2. Улучшение коммуникации

Визуализация эффективна в процессе между различными группами людей. Особенно когда они имеют разные уровни знаний и опыта. Она может стать общим языком для команды разработчиков, управляющих, клиентов и других заинтересованных сторон.

3. Помощь в принятии решений

Визуализация может помочь принимать решения на основе данных и обоснованных фактов. Графическое представление данных может помочь идентифицировать проблемы, выявить паттерны и тренды, и найти новые возможности для развития.

4. Улучшение процессов и продуктов

Визуализация может быть использована для анализа процессов и продуктов, а также для определения их проблемных областей. Это может помочь оптимизировать процессы, усовершенствовать продукты и повысить эффективность бизнеса.

Алгоритмы размещения виртуальных машин в вычислительных центрах — это сложные задачи, требующие множества операций и вычислительной мощности при принятии решений. Визуализация может помочь в этом процессе, представляя результаты вычислений и расположения виртуальных машин в более понятном и доступном виде. Она может также идентифицировать узкие места в сети и максимизировать использование вычислительных ресурсов, что может повысить производительность и эффективность работы вычислительного центра.

1.2. Разнообразие VMP алгоритмов

Существует множество различных алгоритмов размещения виртуальных машин в вычислительных центрах, каждый из которых решает свои задачи и имеет свои преимущества и недостатки. Ниже приведены некоторые из наиболее распространенных видов алгоритмов размещения виртуальных машин (Fikru Feleke Moges, 2019):

1. Алгоритмы на основе жадных стратегий

Эти алгоритмы решают проблему размещения виртуальных машин на основе локальных решений, выбирая лучшую доступную машину на основе текущей конфигурации. Однако эти алгоритмы не всегда дают глобально оптимальные результаты.

- First Fit: виртуальные машины размещаются на первом доступном физическом сервере, который может их вместить.
- Best Fit: виртуальные машины размещаются на физическом сервере, на котором останется наименьший объем свободной памяти.

2. Оптимизации на основе генетических алгоритмов

Используют эволюционный подход, в котором популяция виртуальных машин изменяется и оценивается на каждой итерации, чтобы найти наилучшее решение. Однако, эти алгоритмы требуют значительных вычислительных ресурсов и могут быть медленными. (P Devarasetty)

- Genetic Algorithm for Virtual Machine Placement (GAVMP): генетический алгоритм, разработанный для размещения виртуальных машин в облачных вычислительных средах.
- A Genetic Algorithm for Resource Allocation in Clouds (GA4RA): генетический алгоритм, разработанный для оптимизации ресурсного распределения в облачных вычислительных средах.

3. Алгоритмы на основе машинного обучения

Эти алгоритмы используют методы машинного обучения, чтобы предсказать, какая физическая машина наилучшим образом подходит для размещения каждой виртуальной машины. Однако, эти алгоритмы требуют больших объемов обучающих данных и не всегда дают оптимальные результаты.

6. Алгоритмы на основе эвристик

Используют эмпирические правила, чтобы принимать решения о размещении виртуальных машин. Они могут быть достаточно быстрыми и легкими в реализации, но при этом не давать оптимальных результатов. Ниже приведены некоторые из них:

- Tabu Search: алгоритм поиска оптимального решения, использующий запреты для избежания повторения ходов, которые привели к невыгодному решению.
- Simulated Annealing: алгоритм, основанный на эмуляции процесса отжига металла, который используется для нахождения оптимальных решений в сложных проблемах оптимизации.
- Ant Colony Optimization: алгоритм, основанный на поведении муравьев, который используется для нахождения оптимального решения в сложных задачах комбинаторной оптимизации.

7. Алгоритмы на основе прогнозирования

Эти алгоритмы используют статистические методы для прогнозирования будущей нагрузки на вычислительный центр, чтобы определить наилучший вариант размещения виртуальных машин. Однако, эти алгоритмы требуют больших объемов данных и могут быть не очень точными.

Конкретный выбор алгоритма размещения виртуальных машин зависит от многих факторов, таких как тип приложения, объем нагрузки, требования к производительности и доступность ресурсов. Часто в реальных системах используются комбинации различных алгоритмов, чтобы достичь лучшего результата. (А. Beloglazov, 2012) Некоторые из распространенных алгоритмов включают использование комбинации жадных стратегий для решения

проблем малого масштаба и генетических алгоритмов для решения проблем большого масштаба, а также эвристик для быстрой оптимизации.

В целом выбор наиболее подходящего алгоритма зависит от конкретной задачи и требует тщательного анализа и оценки различных факторов. Однако в любом случае, визуализация является важным инструментом для понимания процесса размещения виртуальных машин и оценки эффективности различных алгоритмов.

1.3. Простейшие полиномиальные алгоритмы

К категории простейших алгоритмов относятся FFD (First Fit Decreasing – Первый подходящий по убыванию) и BFD (Best Fit Decreasing – Наилучший подходящий по убыванию). Все исходные «предметы» упорядочиваются по не возрастанию и пакуются в контейнеры. В случае FFD выбирается первый попавшийся контейнер, в котором достаточно места, в случае BFD выбирается контейнер, разность свободного пространства и объема предмета которого минимальна.

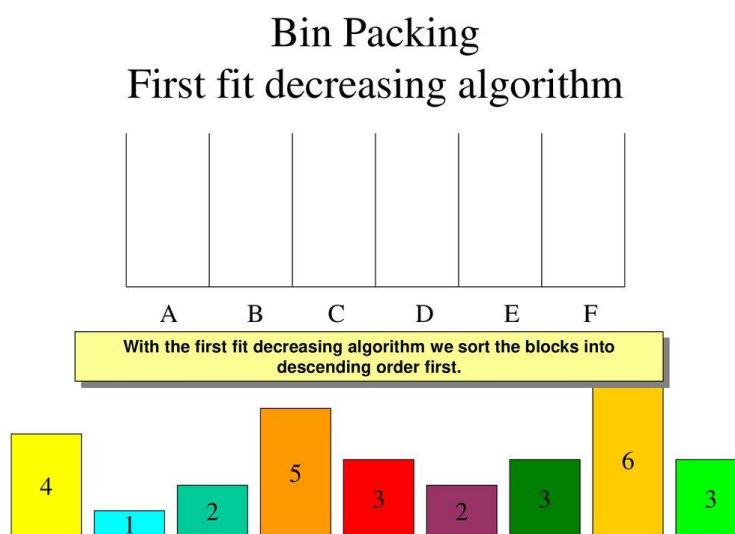


Рисунок 4, Изначальная конфигурация перед запуском алгоритма First Fit

Таким образом для FFD сложность алгоритма в худшем случае будет линейная, а для BFD в худшем случае квадратичная (Дазарев). При нехватке свободного места мы добавляем новый контейнер, куда и будет помещен «предмет». Очевидно, что разница в алгоритмической сложности очень большая, но существуют различные способы оптимизации такого подхода. К примеру, введение двух списков – для предметов и для контейнеров. В таком случае второй список можно содержать в отсортированном состоянии, что значительно скажется на «среднем» показателе сложности.

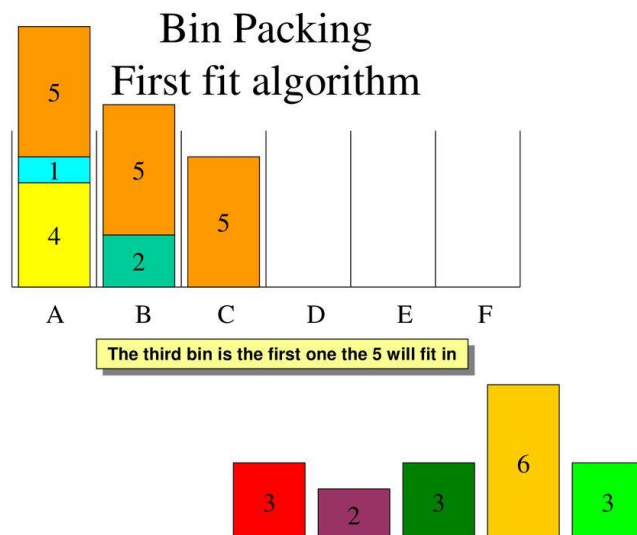


Рисунок 5, Четвертая итерация алгоритма First Fit

Выше (см. Рисунок 4) проиллюстрирована изначальная конфигурация перед запуском алгоритма FF. Каждую следующую итерацию выбирается новый элемент, и тот контейнер, в который он будет помещен. Элемент с объемом «4» помещается в первый контейнер, элемент с объемом «1» - также в первый, элемент «2» - во второй, так как в первом недостаточно места и т. д. На Рисунок 5 представлена четвертая итерация, где видно, как элемент объема «5» помещен в третий контейнер. При переборе всех возможных комбинаций можно заметить, что полученное на данном этапе расположение уже не является оптимальным, так как в первый контейнер можно поместить элементы «4» и «2», а во второй элементы «5» и «1», таким образом оставив третий контейнер не занятым. Этот пример демонстрирует, что «жадные» алгоритмы (алгоритмы, принимающие решения об оптимизации в настоящий момент без возможности планирования) имеют серьезные проблемы, и не могут быть применимы повсеместно (Curtis).

Стоит отметить также существование Worst Fit алгоритма. Его отличие от BF заключается в том, что на этапе анализа всех доступных контейнеров выбирается контейнер с наибольшим количеством свободного пространства. Таким образом мы будем разделять большое количество пространства, делая его более полезным. (Kendall, б.д.) Но такой подход не всегда эффективен, так как зачастую размер «предметов» сильно различается и удаление большого объема пространства может привести к неоптимальному решению.

Задача визуализации перечисленных выше алгоритмов сводится к представлению «предмета» как объекта с константным значением величины, а контейнера – как количество оставшегося свободного пространства внутри. Имеет смысл демонстрировать работу таких алгоритмов вместе, так как концептуально они очень схожи, а различия легко можно увидеть в результате работы, продемонстрировав контейнеры, заполненные «предметами».

В рамках практической части работы реализован прототип, показывающий принципы работы алгоритмов FF и BF на одном наборе данных в виде трехмерного представления (Рисунок 6).

[vmp-visualisation-prototype](#)

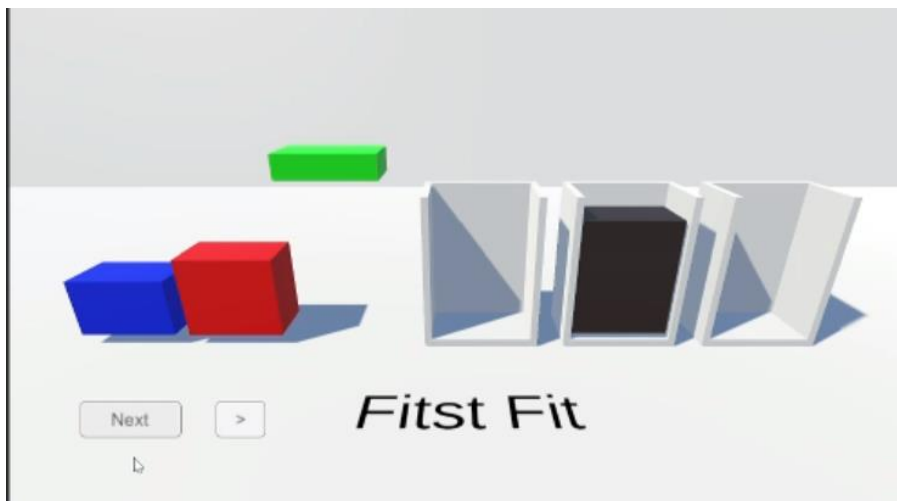


Рисунок 6, Кадр из прототипа визуализации

1.4. MBFD – алгоритм

Modified Best Decreasing алгоритм является модификацией BF алгоритма конкретно для решения задачи размещения ВМ и является стандартным поведением CloudSim. В данном случае мы рассматриваем не только абстрактный «объем» контейнера, но и конкретные вычислительные параметры – CPU и объем RAM. MBFD выбирает контейнеры с минимально доступным ресурсом CPU, который соответствует текущей виртуальной машине. В случае равенства выбирается контейнер с наименьшей доступной оперативной памятью.

Так как алгоритм основан на BF, его неоспоримым преимуществом является минимизация использованных контейнеров, что приводит к снижению потребления энергии. Но алгоритм имеет ряд недостатков – размещение ВМ на самом часто используемом хосте вероятно приведет к перегрузке и нарушению работы и нарушению SLA. (Fikru Feleke Moges, 2019)

Визуализация работы MBFD сводится к усложнению прототипа путем расширения набора параметров «предмета» и контейнера. Нет смысла вводить абстрактный объем, так как происходит оперирование реальными параметрами – CPU и объем RAM. Именно эти параметры и участвуют в сравнении.

1.5. Алгоритм имитации отжига

Simulated Algorithm, далее SA, основывается на имитации физического процесса, происходящего при кристаллизации вещества, например при отжиге металлов.

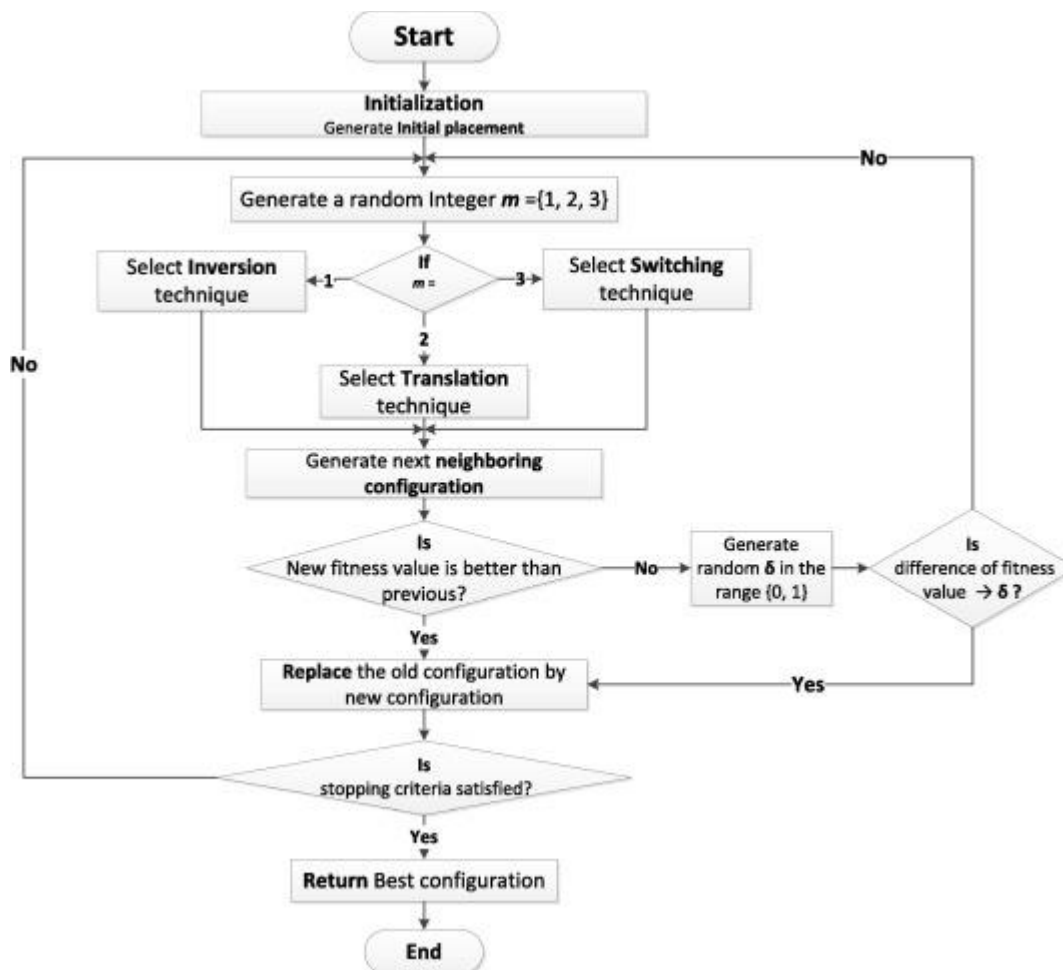


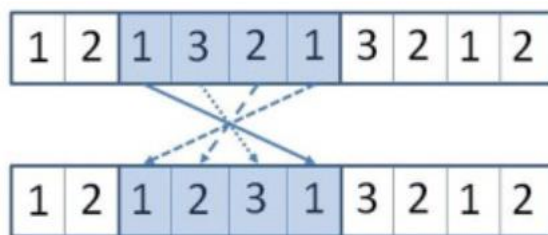
Рисунок 7, Схема работы SA алгоритма

Рассмотрим блок-схему алгоритма (Рисунок 7). Первым шагом является создание начальной конфигурации – набора VM. Эта конфигурация может быть абсолютно случайной, и нашей дальнейшей целью будет привести ее в оптимальное положение. Кроме того, задается изначальная «температура».

Следующий шаг заключается в создании новой конфигурации с помощью одной из трех техник:

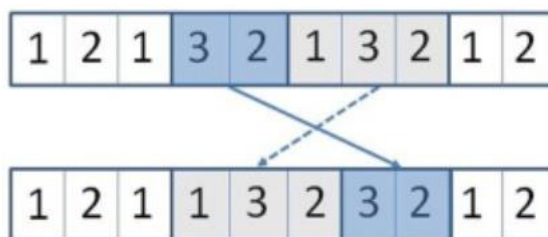
1. Инверсия

При этом процессе мы случайно меняем местами несколько позиций в существующей конфигурации. Ниже представлен пример, в котором выбираются 4 последовательных позиции, после чего они меняются местами в обратном порядке:



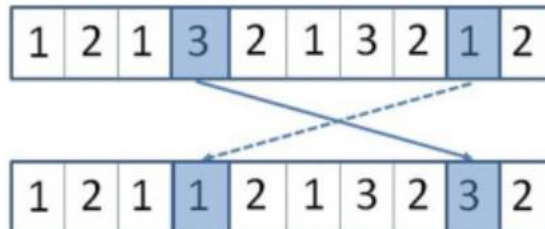
2. Сдвиг

В конфигурации удаляются две или более последовательных позиции, после чего они перемещаются в иное случайное место между двумя другими узлами. Ниже представлен пример сдвига:



3. Переключение

В этом процессе случайным образом выбираются две позиции, после чего они меняются местами для получения новой конфигурации. Ниже представлен пример переключения:



Выбирается только один из процессов, а изменение конфигурации может привести к более оптимальному решению (Sourav KantiAddya).

Для полученной конфигурации вычисляется значение «стоимости». Функция стоимости — это функция, которая оценивает качество решения задачи или модели. Она может быть определена для широкого диапазона задач, включая оптимизацию, машинное обучение, искусственный интеллект, анализ данных и другие.

Функция стоимости может быть простой или сложной в зависимости от задачи. В любом случае она должна быть обоснованной и измеряемой. В контексте нашей проблемы, например, она демонстрирует зависимость нагрузки серверов и текущего расположения виртуальных машин.

После вычисления вышеописанной функции ее значение сравнивается с изначальной «стоимостью» конфигурации. Если новое решение имеет меньшее значение функции стоимости, то оно принимается в качестве

текущего решения «х». Если же значение функции стоимости больше, то новое решение может быть принято с некоторой вероятностью, зависящей в том числе от «температуры». (Rutenbar) В рамках данной работы не будем останавливаться на используемых для вычислений формулах.

На более поздних итерациях вероятность принятия новых конфигураций снижается, так как уменьшаются и показатели температуры с каждой итерацией.

Для визуализации алгоритма имитации отжига можно предложить множество вариантов (Jonas Mockus), рассмотрим некоторые из них:

1. График зависимости функции стоимости от количества итераций

На графике (Рисунок 8) можно отобразить изменения функции стоимости в зависимости от количества итераций алгоритма. Таким образом, можно увидеть, как алгоритм приближается к оптимальному решению. В последствии такой график есть возможность сравнить с аналогичным у генетического алгоритма (ниже).

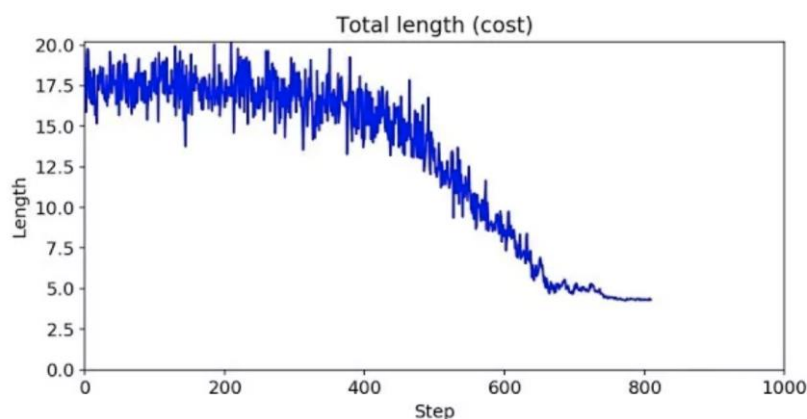


Рисунок 8, График зависимости стоимости от количества итераций

2. Анимация изменения решения на каждой итерации

На каждой итерации можно визуализировать текущее и новое решение, чтобы показать, как алгоритм исследует пространство решений и выбирает наилучшие варианты. В том числе это можно сделать, построив трёхмерную модель стендов, серверов и виртуальных машин.

3. Тепловая карта

Тепловая карта позволяет визуализировать плотность распределения решений в пространстве решений. Цветовая шкала показывает вероятность принятия решения с определенными параметрами. Таким образом, можно увидеть, где находятся наиболее вероятные решения и как они изменяются в зависимости от температуры.

4. График изменения температуры на каждой итерации

График изменения температуры позволяет визуализировать снижение температуры каждой итерации и как результат этого процесса влияет на вероятность принятия новых решений (Рисунок 9).

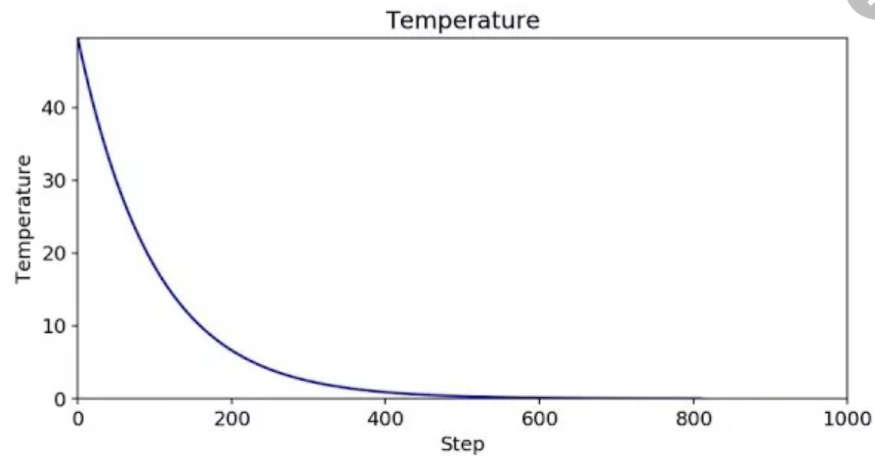


Рисунок 9, График изменения температуры на каждой итерации

5. График распределения вероятностей

График распределения вероятностей позволяет визуализировать вероятность принятия нового решения в зависимости от разницы между функциями стоимости текущего и нового решений и текущей температуры (Рисунок 10). Таким образом, можно увидеть, как вероятность принятия решения меняется в зависимости от параметров алгоритма.

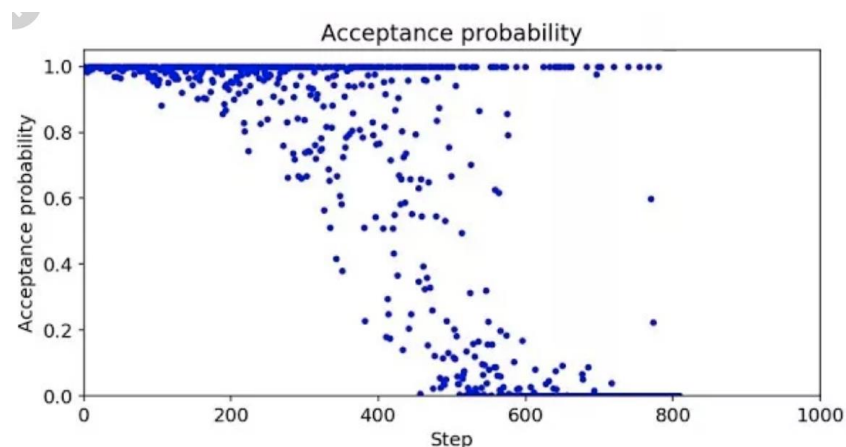


Рисунок 10, Распределение вероятностей принятия решений

1.6. Генетические алгоритмы

Генетические алгоритмы (GA) являются эволюционными алгоритмами оптимизации, которые используют идеи из генетики и эволюционной

биологии для нахождения решения оптимизационной задачи – в нашем случае, задаче размещения виртуальных машин. Генетические алгоритмы состоят из нескольких основных компонентов: генотипа, фенотипа, операторов скрещивания, мутации и селекции (Rania Hassan).

Генотип — это набор генов, которые определяют свойства решения. В контексте задачи размещения виртуальных машин генотип может представляться в виде двоичной строки, где каждый бит кодирует наличие или отсутствие виртуальной машины на конкретном стенде.

Фенотип — это реализация генотипа, то есть конкретное решение задачи. В нашем случае фенотипом будет являться конкретное распределение виртуальных машин на серверах.

Операторы скрещивания — это процедуры, которые комбинируют генотипы двух родителей, чтобы получить новое потомство. При размещении виртуальных машин операторы скрещивания могут использоваться для комбинирования генотипов двух решений и получения нового решения.

Мутация — это процесс изменения генотипа случайным образом. Мутация может быть использована для случайного изменения расположения виртуальных машин на серверах.

Селекция — это процесс выбора лучших решений из популяции для создания следующего поколения. При описании задачи VMP, селекция может использоваться для выбора наилучших распределений виртуальных машин на серверах для создания новых решений (M Wetter).

Применение генетического алгоритма для задачи размещения виртуальных машин в вычислительных центрах может быть следующим:

1. Определить функцию приспособленности

Определить функцию приспособленности, которая будет оценивать качество каждого решения, основываясь на таких факторах, как использование ресурсов, эффективность обмена данными, скорость доступа к данным и другие параметры.

2. Создание начальной популяции

Определение начальной популяции решений, где каждое решение будет представлено генотипом. Каждый генотип соответствует распределению виртуальных машин на серверах. Начальная популяция может быть сгенерирована случайным образом или на основе каких-то эвристических правил.

3. Вычисление функции приспособленности

Вычисление функции приспособленности для каждого решения в популяции. Это позволит определить, насколько хорошо каждое решение подходит для задачи размещения виртуальных машин.

4. Операторы скрещивания и мутации

Применение операторов скрещивания и мутации к генотипам в популяции, чтобы создать новые решения. Например, можно использовать один из многих операторов скрещивания, таких как одноточечное или двухточечное скрещивание, чтобы создать новые генотипы. Затем можно применить оператор мутации, чтобы случайно изменить генотип.

5. Вычисление функции приспособленности

Вычисление функции приспособленности для каждого нового решения, которое было создано в результате операторов скрещивания и мутации.

6. Селекция

Выбор лучшего решения из начальной популяции и новых решений, чтобы создать следующее поколение решений. Можно использовать различные методы селекции, такие как турнирная селекция или рулеточная селекция, чтобы выбрать лучшие результаты.

7. Повторение предыдущих шагов

Повторение шагов 4–6 до тех пор, пока не будет выполнено условие остановки. Это может быть достижение заданного уровня приспособленности, превышение максимального количества поколений или достижение максимального количества оценок функции приспособленности.

8. Выбор лучшего решения

После завершения генетического алгоритма происходит выбор лучшего решения из популяции в качестве окончательного результата. Это решение будет представлять оптимальное распределение виртуальных машин на серверах, которое удовлетворяет всем требованиям и ограничениям.

ГА может быть очень эффективен для задачи размещения виртуальных машин в вычислительных центрах, из-за способности работать с большим количеством переменных и ограничений.

Применение генетического алгоритма решения для поставленной задачи может быть достаточно простым (S.Garg, 2022). В качестве генотипов используются бинарные строки, где каждый бит будет соответствовать наличию или отсутствию виртуальной машины на сервере. Например, если у нас есть 10 серверов и 50 виртуальных машин, то генотип представлен строкой из 500 битов ($10 \text{ серверов} \times 50 \text{ виртуальных машин} = 500 \text{ битов}$).

Функция приспособленности может быть определена на основе различных критериев, таких как количество использованных серверов, распределение нагрузки, доступность и надежность. Критерии можно совместить в одну функцию приспособленности, которая будет оценивать решение как целое.

Операторы скрещивания и мутации могут быть разработаны таким образом, чтобы учитывать ограничения задачи, такие как количество виртуальных машин, которые могут быть размещены на каждом сервере, и доступность и надежность серверов.

GA может быть сконфигурирован путем изменения различных параметров, таких как размер популяции, вероятность скрещивания и мутации, количество поколений и выбор метода селекции. Их настройка может повлиять на эффективность и скорость сходимости алгоритма.

Однако генетический алгоритм также имеет свои недостатки и ограничения. Один из основных недостатков — это высокая вычислительная сложность (Rania Hassan). Использование генетического алгоритма может быть затруднительным для задач с большим количеством виртуальных машин и серверов, так как возможных решений может быть очень много.

Еще одним недостатком может являться вероятность «застревания» в локальных минимумах. В зависимости от выбора операторов скрещивания и мутации алгоритм может застрять в невыгодном положении и не найти оптимального решения.

Также важно отметить, что алгоритм может не учитывать специфические особенности и ограничения виртуальных машин и серверов, такие как требования к производительности, обработке данных и защите информации. Для их учета могут потребоваться дополнительные модификации и настройки алгоритма (S Jamali). В целом эффективность GA зависит от конкретной задачи, настроек параметров и выбора операторов скрещивания и мутации. Для достижения оптимального решения может потребоваться комбинация различных алгоритмов и подходов.

Как и в случае с оптимизационным алгоритмом имитации отжига, генетический алгоритм можно визуализировать с помощью диаграмм и графиков, которые отображают ключевые этапы алгоритма и его результаты.

1. Диаграмма потока работы GA

Эта диаграмма представляет последовательность действий, выполняемых в ходе работы алгоритма. Диаграмма может содержать блоки, представляющие операторы, такие как скрещивание, мутация, отбор и оценка, а также связи между блоками, которые указывают на порядок выполнения операторов. Для каждого блока может быть указано количество особей, проходящих через него.

2. График сходимости

Это график, показывающий как качество решения изменяется во время выполнения алгоритма. На нем отображается значение целевой функции (например, средняя загрузка серверов) в зависимости от числа поколений. Этот график позволяет оценить, как быстро алгоритм находит оптимальное решение и сколько поколений требуется для этого.

3. Диаграмма популяции

Диаграмма, отображающая распределение особей в популяции. Она может содержать гистограмму, показывающую количество особей в каждом из возможных состояний, или диаграмму рассеяния, которая отображает распределение особей в пространстве решений. Эта диаграмма может помочь понять, какие области пространства решений наиболее популярны и как эти области меняются во время выполнения алгоритма.

4. Визуализация лучшего решения

Это графическое представление лучшего решения, найденного в ходе выполнения алгоритма. Например, для задачи VMP, лучшее решение может быть представлено в виде графа, который показывает, какие виртуальные машины размещены на каких серверах. Это позволяет лучше понять, как алгоритм находит оптимальное решение и какие факторы на это влияют.

5. Генетическое дерево

Графическое представление структуры популяции и ее развития во времени. Дерево состоит из узлов, которые представляют популяции в различные моменты времени, и ребер, которые показывают, как популяции связаны друг с другом (например, как они производят потомков). Дерево может помочь визуализировать, как происходят операции скрещивания и мутации, и как они влияют на структуру популяции.

6. Анимация

Показать в динамике принцип работы алгоритма – например, создать анимацию, которая показывает, как популяция изменяется во времени и как происходят операции скрещивания и мутации.

7. График показателей

График демонстрирует, как различные показатели изменяются во время выполнения алгоритма. Например, можно построить график, который

отображает среднее значение целевой функции, максимальное и минимальное значение, а также стандартное отклонение.

8. 3D-визуализация

Для задачи размещения виртуальных машин можно создать 3D-модель серверной инфраструктуры, на которой отобразится расположение виртуальных машин и их перемещение при смене популяции.

1.7. Алгоритмы на основе прогнозирования

Алгоритмы на основе прогнозирования используются для анализа данных и определения тенденций, которые могут быть использованы для предсказания будущих значений. В контексте размещения виртуальных машин в вычислительных центрах эти алгоритмы могут помочь предположить будущую нагрузку на вычислительный центр и определить, какие ресурсы будут необходимы для ее обработки. Вот некоторые примеры алгоритмов на основе прогнозирования (Снитюк) :

1. Авторегрессионная модель (AR)

AR-модель используется для прогнозирования будущих значений на основе предыдущих данных с использованием временных рядов. В рамках VMP задачи AR-модель может использоваться для предсказания будущей нагрузки на вычислительный центр на основе предыдущих значений нагрузки.

2. Метод экспоненциального сглаживания (ES)

ES-модели используются для прогнозирования будущих значений на основе сглаженных предыдущих значений. Для этой цели они применяют экспоненциально взвешенные средние. ES-модель может использоваться для прогнозирования будущей нагрузки на вычислительный центр на основе сглаженных предыдущих значений нагрузки.

3. Нейронные сети

Нейронные сети используются для обработки больших объемов данных и для прогнозирования значений на основе взаимодействия между переменными. В нашем случае нейронные сети могут применяться для прогнозирования будущей нагрузки на вычислительный центр на основе множества переменных, таких как количество пользователей, тип приложений и другие.

Прогнозирование является важным инструментом для определения будущих требований к ресурсам и выбора наиболее подходящего алгоритма

размещения виртуальных машин. Однако, важно помнить, что любые прогнозы могут содержать ошибки и что принятие решений должно основываться на анализе нескольких факторов (Снитюк).

Один из примеров визуализации прогнозов — это график временных рядов. На этом графике исходные данные представлены как точки на графике, а прогнозы отображаются в виде линии, которая соединяет прогнозируемые значения во времени. Этот тип графика позволяет увидеть, как модель прогнозирования выполняет прогнозы на основе исходных данных. На (Рисунок 11) представлен пример подобного графика.

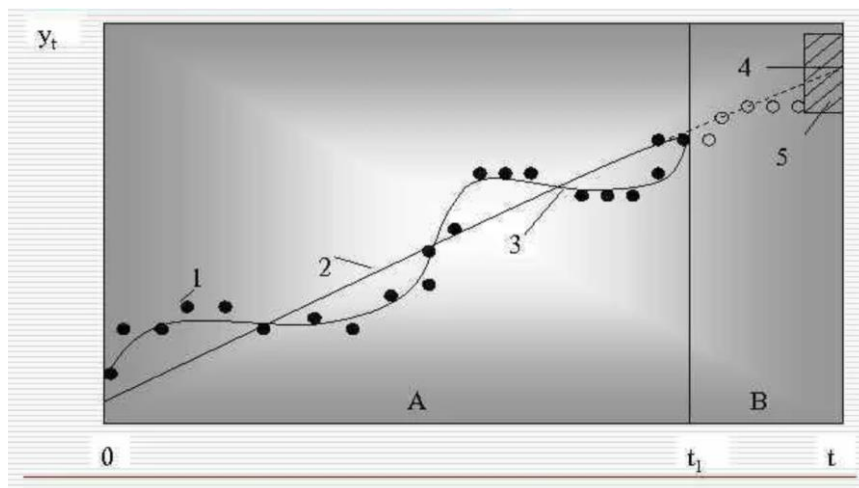


Рисунок 11, Прогнозирование на основе временных рядов

«1» - экспериментальные данные на этапе наблюдений, «2» - тренд, «3» - тренд и сезонная волна, «4» - значение точечного прогноза, «5» - интервальный прогноз.

Другим способом визуализации может быть использование круговых или столбчатых диаграмм. Эти диаграммы могут применяться для сравнения прогнозируемых значений с реальными значениями и для отображения предполагаемых значений в различных категориях.

Также можно использовать интерактивные графики, которые позволяют пользователю взаимодействовать с данными и изменять настройки отображения. Например, график может дать возможность пользователю выбрать интервал времени, для которого нужно построить прогноз, или настроить параметры модели прогнозирования.

Для визуализации прогнозов могут быть использованы различные инструменты, такие как Python-библиотеки Matplotlib, Seaborn или Plotly, а также инструменты для создания интерактивных визуализаций, такие как D3.js или Tableau.

1.8. Обобщение подходов к визуализации алгоритмов

Исходя из анализа вышеописанных алгоритмов и методов их визуализации, можно сделать вывод, что некоторые подходы применимы

сразу ко множеству алгоритмов. Рассмотрим их подробнее и сравним (см. **Ошибка! Источник ссылки не найден.**).

Таблица 1, перечисление методов визуализации

Метод визуализации	Описание	Пример
Графическое представление	График времени выполнения алгоритма, диаграмма загрузки вычислительных ресурсов, график изменения числа виртуальных машин и т. д.	Рисунок 8, График зависимости стоимости от количества итераций
Интерактивная визуализация	Изменение параметров алгоритма в режиме реального времени	
Анимация	Процесс решения задачи по шагам	Рисунок 1, Кадр из анимации решения bin package problem
3D - визуализация	Использование трехмерных моделей для отображения физического расположения вычислительных ресурсов и виртуальных машин	Рисунок 6, Кадр из прототипа визуализации

1. Графическое представление

Этот метод включает использование диаграмм и графиков для отображения параметров и метрик алгоритма. Это может быть график времени выполнения алгоритма, диаграмма загрузки вычислительных ресурсов, график изменения числа виртуальных машин и т. д. Графическое представление позволяет быстро оценить производительность алгоритма и сравнить его с другими.

2. Интерактивная визуализация

Позволяет пользователю взаимодействовать с визуализацией и изменять параметры алгоритма в режиме реального времени. Например, пользователь может изменять количество виртуальных машин, выбирать различные метрики и параметры алгоритма и смотреть, как они влияют на результат.

3. Анимация

Позволяет отображать процесс размещения виртуальных машин в виде анимации. Например, можно показать, как виртуальные машины перемещаются и изменяют свое положение на вычислительных ресурсах в процессе размещения. Это может помочь пользователю лучше понять, как работает алгоритм и какие изменения можно внести, чтобы улучшить его производительность.

4. Визуализация в 3D

Включает использование трехмерных моделей для отображения физического расположения вычислительных ресурсов и виртуальных машин. Это может быть особенно полезным для анализа алгоритмов размещения, которые учитывают физические характеристики вычислительных ресурсов, такие как расположение, температура и т. д.

Статичное графическое представление, как правило, имеет ряд минусов по сравнению с 3D визуализацией. Статичные графические представления могут быть более ограниченными в своей способности передавать информацию. В 3D визуализации мы можем увидеть объекты в разных углах и понять их расположение относительно других объектов, что невозможно сделать в статичном графическом представлении. Кроме того, благодаря трехмерному занимаемому пространству 3D визуализация может содержать больше информации и деталей.

Статичное графическое представление обычно не дает возможности взаимодействия с данными. 3D визуализация может быть интерактивной, позволяя пользователю вращать объекты, изменять их масштаб, выбирать элементы для просмотра подробной информации и т. д.

Одним из главных преимуществ 3D визуализации является возможность реалистичного отображения объектов. К примеру, можно увидеть, как выглядит каждый сервер и его местоположение в вычислительном центре, а также какие виртуальные машины размещены на серверах. 3D визуализация позволяет обнаруживать взаимодействия между объектами, которые могут быть невидимы на плоскости 2D. Кроме того, 3D визуализация может представляться в различных визуальных стилях, что позволяет настраивать отображение в соответствии с конкретными потребностями. Например, можно использовать различные цветовые схемы для обозначения типов серверов и виртуальных машин, что упрощает восприятие информации.

Таким образом, именно 3D визуализация является наиболее полезным и эффективным инструментом для демонстрации принципов работы алгоритмов размещения виртуальных машин в вычислительных центрах, так как позволяет наглядно представить пространственное размещение объектов, их взаимодействие и загрузку серверов.

2. Разработка визуализации алгоритмов VMP

В данной главе представлен поэтапный процесс разработки приложения 3D визуализации алгоритмов размещения виртуальных машин: начиная с реализации алгоритмов и заканчивая созданием 3D моделей. Первостепенная задача - выбор инструментов для разработки

2.2. Выбор технологий для реализации

Разработка приложения для 3D визуализации алгоритмов размещения виртуальных машин может включать в себя использование различных инструментов, таких как игровые движки, программы для моделирования, языки программирования и т. д. Во введении уже упоминались некоторые из них (Стр. 13). Рассмотрим те, которые могут быть использованы для поставленной цели 3D визуализации. Ниже представлены наиболее популярные инструменты, используемые в разработке подобных приложений:

- Игровые движки: Unity, Unreal Engine, CryEngine, Amazon Lumberyard, Godot.
- Программы для моделирования и создания 3D объектов: Blender, Autodesk 3ds Max, Maya, ZBrush, Cinema 4D.
- Языки программирования: C++, C#, Java, Python, JavaScript.
- Библиотеки для 3D визуализации: OpenGL, DirectX, WebGL, Three.js, Babylon.js.
- Фреймворки для создания веб-приложений с 3D визуализацией: A-Frame, React 3D, Three.js, Babylon.js.
- Инструменты для создания анимации: Adobe After Effects, Autodesk Maya, Blender.
- Инструменты для создания аудиоэффектов: Adobe Audition, Ableton Live, FL Studio.

Выбор конкретных инструментов зависит от требований к проекту, опыта разработчика, доступности ресурсов и технических возможностей. Для данного проекта выбраны следующие технологии:

1. Unity Engine

Игровой движок подходит для любых задач, включая трехмерные и двумерные игры и визуализации. Поддерживает множество возможностей для настройки анимации, спецэффектов, освещения, моделей, взаимодействия скриптов и так далее. Кроме того – возможность собрать проект под разные платформы, включая сборку в веб приложение для дальнейшего размещения на любом сайте. По

сравнению с основным аналогом – Unreal Engine, имеет меньший порог входа.

2. Autodesk 3ds Max и Autodesk Inventor

Комбинация двух инструментов 3D – моделирования, выполняющих разные задачи – анимация сложных объектов и система автоматизированного моделирования (САПР) соответственно. Являются стандартом индустрии в своих областях и прекрасно подходят для решаемой задачи.

3. Язык программирования C#

Основной поддерживаемый язык платформы Unity Editor, поддерживает ООП, имеет прямую интеграцию с Unity Engine

Выбор обусловлен опытом разработчиков в использовании вышеперечисленных инструментов, а также их распространенностью при реализации подобных проектов, что в дальнейшем может помочь в поддержке и развитии разработанного приложения.

2.3. Описание UI и UX

Функциональность приложения сводится к нескольким пунктам:

1. При запуске пользователь выбирает количество серверов и стендов, которые необходимо распределить по стендам от 1 до 10.
2. Для каждого сервера задается его размерность. Размерность определена числом от 1 до 10.
3. После подтверждения начальных настроек пользователь видит сцену с выставленными стендами и серверами.
4. В пользовательском интерфейсе выбирается конкретный алгоритм, который будет использоваться при размещении.
5. Кнопка «*Next*» размещает следующий по очереди сервер на стенд в соответствии с выбранным алгоритмом.
6. Кнопка «*Solve*» размещает автоматически все объекты по стендам по порядку, без необходимости нажатия «*Next*».
7. Если задействованы генетический алгоритм или алгоритм плавления, пользователь при нажатии кнопки видит не перемещение одного сервера, а целую итерацию этого алгоритма – расположение всех серверов по стендам.
8. Кнопка «*Reload*» возвращает объекты в исходное положение и позволяет выбрать другой алгоритм.

9. Кнопка «*Restart*» сбрасывает установленную конфигурацию и позволяет заново выбрать и настроить ее.

Разделение на «Next» и «Solve» позволит нагляднее рассмотреть работу алгоритмов итерационно. Это качественно повысит уровень понимания и доступности.

Кроме того, по результатам анализа, проведенного в первой главе, можно сделать вывод о том, что необходимы дополнительные параметры настройки. К примеру, генетический алгоритм должен конфигурироваться путем изменения количества итераций и коэффициента мутации. Алгоритм имитации отжига – начальной температурой и коэффициентом охлаждения.

Соответствующие настройки должны быть доступны в интерфейсе.

2.4. Реализация алгоритмов на языке программирования

Рассмотрим реализацию наиболее простых «жадных» алгоритмов. Алгоритмы First Fit и Best Fit являются алгоритмами решения задачи о рюкзаке, которая заключается в том, чтобы выбрать оптимальный набор предметов, которые можно уложить в рюкзак, учитывая его вместимость и стоимость предметов. Подробнее про алгоритм работы - в первой главе.

Заведем коллекции для серверов и стендов для каждого из алгоритмов, и начнем реализовывать их. Подробный принцип работы алгоритмов описан в разделе 1.3., поэтому не будем подробно останавливаться на каждом. В описании UI и UX объяснена необходимость итерационного вызова алгоритмов. Поэтому реализация полного цикла работы осуществлена с помощью многократного вызова итерационных методов.

Реализуем алгоритм First Fit (Блок кода 2, реализация First Fit на C#) и Best Fit (Блок кода 1, реализация Best Fit на C#). Алгоритм берет следующий сервер в очереди и находит подходящий стенд, вставляет сервер и одновременно передвигает все сервера дальше.

Реализация SA (см. раздел 1.5.) разделена на две части – поиск следующей расстановки и физическое перемещение серверов. Необходимо иметь функцию просчета энергии (Блок кода 4, расчёт текущей энергии), функцию генерации соседнего решения (Блок кода 5, генерация нового состояния SA), и полный цикл итерации с определением расстановки и снижением текущей температуры (Блок кода 3, реализация SA на C#).

Функция расчёта энергии использует следующий подход: сумма всех серверов на стенде умноженная на (номер стенда с конца + 5). Эта функция уменьшается, если алгоритм собирает больше серверов в одном стенде и, если сервера расположены ближе к 1 стенду.

Функция генерации соседнего решения работает следующим образом: алгоритм проходится по серверам и случайным образом выбирает передвигать сервер или нет, если нужно передвигать, выбирается случайный стенд, на котором достаточно места. Подробнее про передвижение описано в разделе 1.5.

Генетический алгоритм (см. раздел 1.6.) реализован следующим образом:

1. Генерируется случайное поколение, популяция (количество решений) которого равна N (Блок кода 7, генерация мутации).
2. Случайным образом берется либо каждое второе, либо каждое третье решение. После чего оно сравнивается с соседним решением.
3. Для решений подсчитывается функция приспособленности.
4. На место решения с меньшим значением функции создается потомство решения с наибольшим значением приспособленности (Блок кода 8, расчет приспособленности и продолжение потомства).
5. После этого из всего поколения выбирается решение с наибольшим счетом, и оно применяется для следующего расположения серверов (Блок кода 6, реализация генетического алгоритма на C#).

2.5. Создание 3D – моделей и окружения

Необходимо разработать два основных объекта на сцене – сервера различных размеров, которые будут распределяться, и стенды, вмещающие сервера. В качестве образца взяты существующие распространенные модели, использующиеся в вычислительных центрах (Рисунок 12).



Рисунок 12, Центр обработки данных

Ниже приведены рендеры моделей – полностью заполненный стенд (Рисунок), пустой стенд без серверов (**Ошибка! Источник ссылки не найден.**), и процесс вставки сервера в динамике (Рисунок 15).

Одиночный сервер (Ошибка! Источник ссылки не найден.) с оответствует размерности «1», а увеличенные размерности представляют собой стопки одиночных серверов.



Рисунок 13, Заполненный стенд с серверами

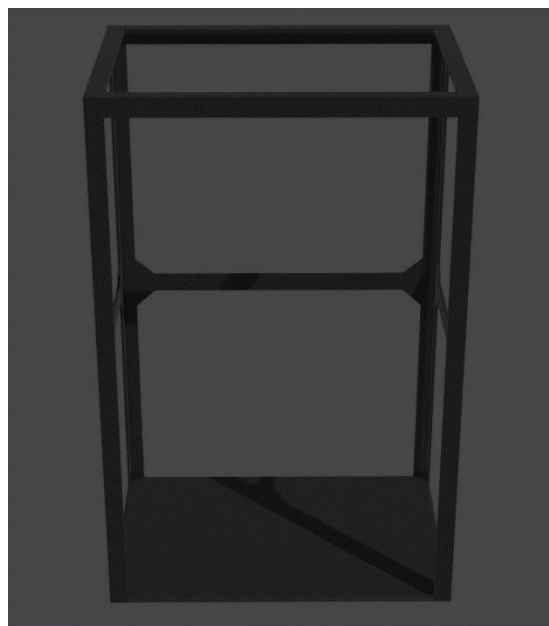


Рисунок 14-1, Пустой стенд



Рисунок 14-2, Отдельный сервер



Рисунок 15, Вставка сервера на стенд в динамике

Помимо основных динамических объектов, для визуализации нужно создать окружение и задний план. Это сильно повысит качество и общее восприятие разработанной визуализации. Ниже приведены некоторые из разработанных объектов (см. Рисунки 16, 17-1, 17-2, 17-3).

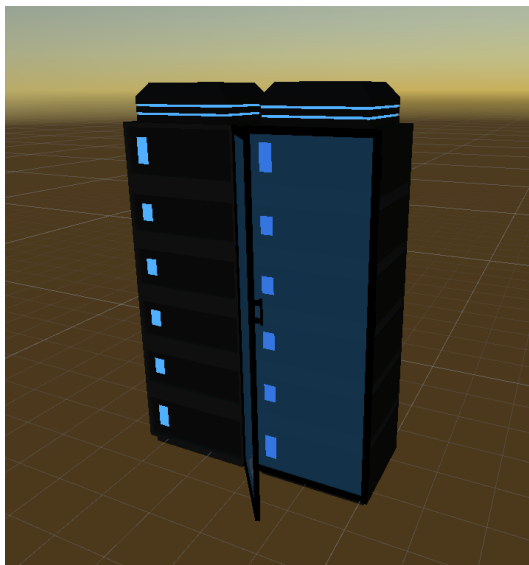


Рисунок 16, Двойной серверный стенд

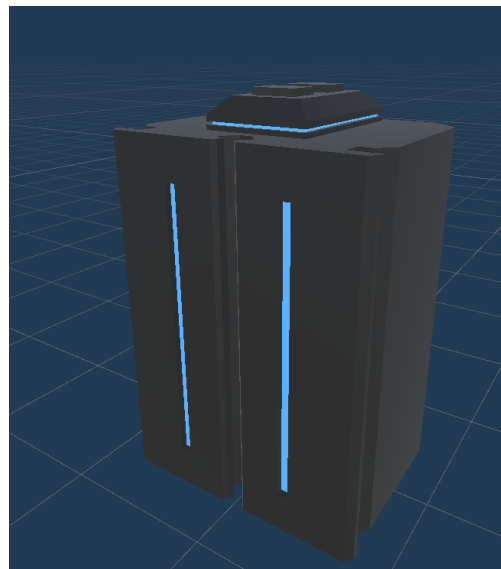


Рисунок 17-1, Закрытый серверный стенд

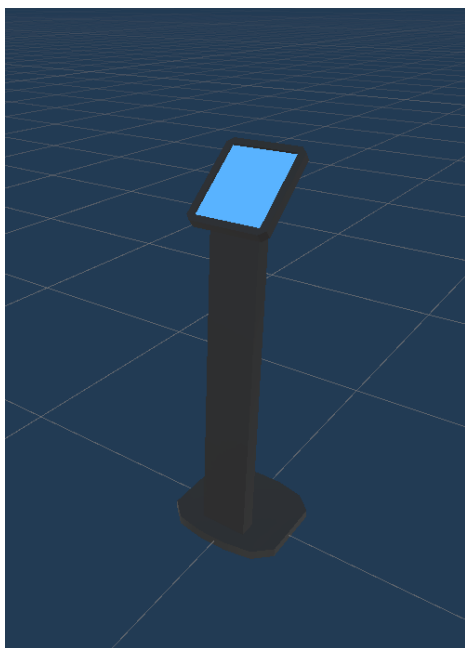


Рисунок 17-2, Панель управления

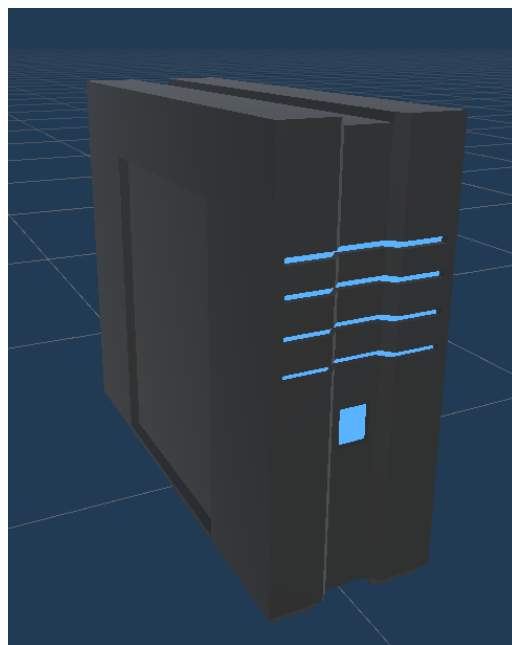


Рисунок 17-3, Компьютерный блок

Общий вид окружения показан на Рисунке 18-1 и Рисунке 18-2 ниже. Стоит заметить, что на них уже применены эффекты пост-обработки. О них подробнее будет сказано в разделе 2.6.



Рисунок 18-1, Окружение (1)

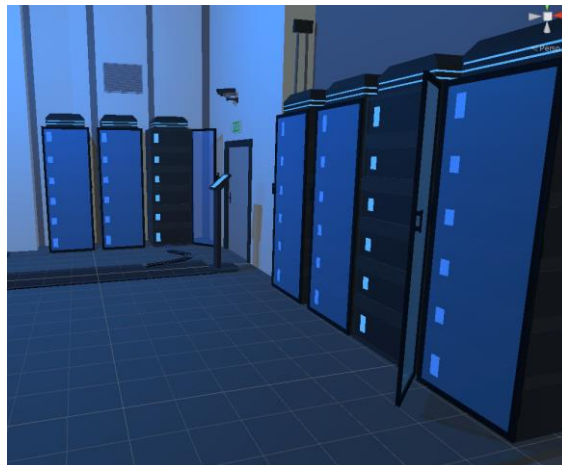


Рисунок 18-2, Окружение (2)

2.6. Графическое программирование и пост-процессинг

Пользовательский интерфейс приложения реализован с помощью утилиты UI Toolkit, позволяющей создавать настраиваемую верстку с помощью языка разметки и CSS.



Рисунок 18, Стартовый экран

Всего реализовано более 5 разных оверлеев для разных конфигураций и этапов визуализации. Для каждого типа алгоритмов в интерфейс вынесены настройки параметров и описания этапов.



Рисунок 19, Внутриигровой интерфейс

На Рисунке 19 показан интерфейс настройки генетического алгоритма, а именно прогресс расстановки, информация о популяции и изменяемые параметры и коэффициенты. UI оверлеи обновляются в реальном времени, читая данные из скриптов.

Передвижение объектов на сцене осуществлено с помощью функции «Lerp» (линейная интерполяция), в параметры которой передаются текущее положение сервера, нужное положение и фракция для обновления положения. Это все создает плавное передвижение объекта с одной трёхмерной координаты в другую. В генетическом и SA сервера сразу ставятся в нужное положение для экономии времени.

В разработанной визуализации также используются собственные эффекты пост-процессинга. Независимо от того, что Unity Engine уже использует некоторые эффекты по умолчанию, их бывает недостаточно для качественной картинки. Необходимо дополнительно подчеркнуть и выделить тени, эффекты сглаживания, цветокоррекцию и другие эффекты пост обработки.

В данной работе были использованы следующие механизмы:

1. Subpixel Morphological Anti-Aliasing

Это техника сглаживания краев в компьютерной графике, которая позволяет уменьшить эффект ступенчатости (Aliasing) на краях объектов в сцене. Эта техника используется для улучшения качества изображения, особенно на экранах с низким разрешением. Она необходима для сцен с использованием низко полигональных моделей, как в нашем случае.

2. Bloom

Это эффект визуального освещения, который используется для создания ярких, светящихся и блестящих объектов в сцене. Этот эффект имитирует рассеивание света в глазах зрителя, когда он смотрит на очень яркий источник света. В данном случае он применен для имитации свечения экранов и элементов серверов.

3. Ambient Occlusion

Техника, которая используется для создания более реалистичных теней и освещения в сценах. Эта техника основывается на предположении, что в затененных областях объектов, которые находятся близко друг к другу, должно быть меньше света, чем в открытых областях.

Ambient Occlusion создает тени, основанные на расстоянии между объектами, а не на направлении света. Это позволяет создавать более мягкие и естественные тени, особенно в тех областях, где объекты находятся близко друг к другу.

4. Color Grading

Позволяет применить к изображению игры дополнительную цветокоррекцию. Например, можно добавить художественности сцене, добавив какой-то цветовой оттенок.

5. Reflection Probe

Компонент, который используется для создания отражений и преломлений объектов в реальном времени. Он позволяет создавать реалистичные отражения на поверхностях объектов, отображая окружающую среду, освещение и другие объекты. Reflection Probe работает путем измерения освещения и других параметров в заданном пространстве, и сохранения этих данных в текстуре, которая затем используется для рендеринга отражений на поверхностях объектов.

3. Тестирование разработанного приложения

В данной главе представлен процесс и результаты тестирования разработанной 3D визуализации. Приложение оценено по нескольким критериям, которые описаны далее.

3.1. Подход к тестированию визуализации

Важно отметить, что трёхмерная визуализация демонстрирует лишь работы на ограниченном наборе данных, в то время как реальные системы оперируют десятками тысяч виртуальных машин (Dabiah Ahmed Alboaneen, 2016). Это ограничение необходимо для упрощения понимания происходящего и снижения нагрузки на вычислительные и графические ресурсы. Проверять работу приложения в данном случае необходимо с помощью тестирования на искусственно созданных данных. А именно создание нескольких сценариев с различным количеством виртуальных машин и стендов, которые можно использовать для тестирования различных алгоритмов.

Кроме того, необходимо заранее определить критерии анализа полученного решения. Перечислим наиболее важные:

1. Эффективность

Тестирование на время выполнения алгоритмов, использование ресурсов и производительности.

2. Точность

Сравнение результатов алгоритмов с ожидаемыми результатами.

3. Надежность

Тестирование на устойчивость алгоритмов к неправильным данным, ошибкам и другим неожиданным условиям.

4. Понятность

Оценка того, насколько легко понять и использовать визуализацию.

5. Гибкость

Возможность настройки параметров алгоритмов и визуализации, чтобы адаптировать их к различным условиям.

6. Наглядность

Визуализация должна ясно и наглядно отображать процесс размещения виртуальных машин на хостах и общее состояние системы в целом.

Начнем анализ с качества реализации алгоритмов и эффективности их работы, после чего оценим общее качество визуализации.

3.2. Тестирование алгоритмов

Перед началом тестирования стоит упомянуть, что в рамках этого процесса мы рассматриваем именно конкретные реализации оптимизационных алгоритмов, созданные специально для разрабатываемого приложения. Процесс реализации был описан ранее в разделе 2.4. (стр. 37).

3.2.1. Анализ FF и BF

Для проведения тестирования и сравнительного анализа его результатов необходимо задать статичную конфигурацию не менять ее при переключении режимов работы. Рассмотрим на примере сравнение First Fit и Best Fit.

Для First Fit (Рисунок 20) посчитаем средний процент свободно пространства в неполных стендах. Для текущей конфигурации он равен 16%. Для Best Fit (Рисунок 21) это значение будет составлять уже 26%. Чем меньше процент свободно места на стенде, тем сложнее будет в дальнейшем подобрать подходящую виртуальную машину. В то же время, когда машины скомпонованы более тесно внутри одного стенда, это освобождает значительно больше места на оставшихся стендах и дает большую вариативность для дальнейшей миграции или размещения новых ВМ.

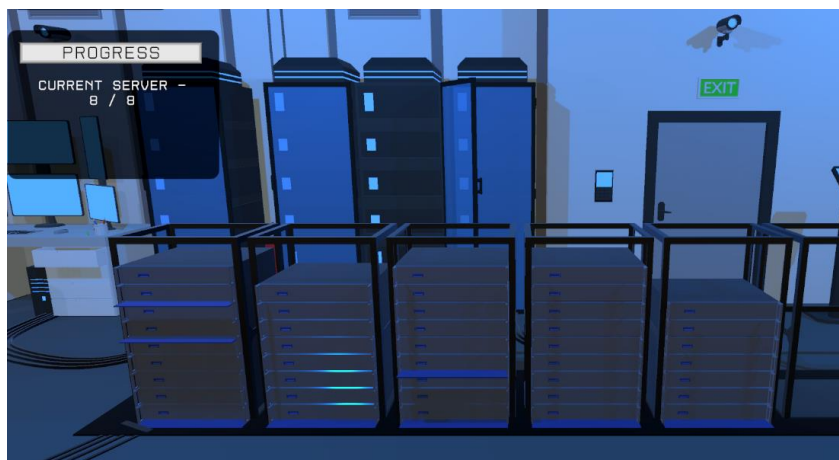


Рисунок 20, Результат работы First Fit



Рисунок 21, Результат работы Best Fit

На полученном в результате визуализации прекрасно прослеживается недостаток алгоритма FF по сравнению с BF – FF не гарантирует оптимальное использование памяти (Fikru Feleke Moges, 2019). Одно из преимуществ FF – более быстрая работа и меньшее использование памяти, в данном случае несут существенны из-за небольшой конфигурации. Протестируем эту же конфигурацию на других алгоритмах.

3.2.2. Анализ SA алгоритма

На продемонстрированных примерах при одной конфигурации серверов меняется начальная температура и коэффициент охлаждения. При очень высокой начальной температуре (см. Рисунок 23) алгоритм начинает «прыгать» между совершенно разными решениями, что приводит к пропуску оптимального решения.



Рисунок 22, Результат SA



Рисунок 23, Результат SA (высокая температура)

Изменение коэффициента охлаждения влияет лишь на скорость сходимости алгоритма, и снижение значения коэффициента приводит к сильному увеличению времени работы. Разница в эффективности решений на приведенных примерах обосновывается случайностью самого SA алгоритма и его перестановок (см. Рисунок 24). Так как мы тестируем процесс на небольшом наборе решений, эта случайность может привести к серьезным отклонениям при запуске одной и той же конфигурации несколько раз.

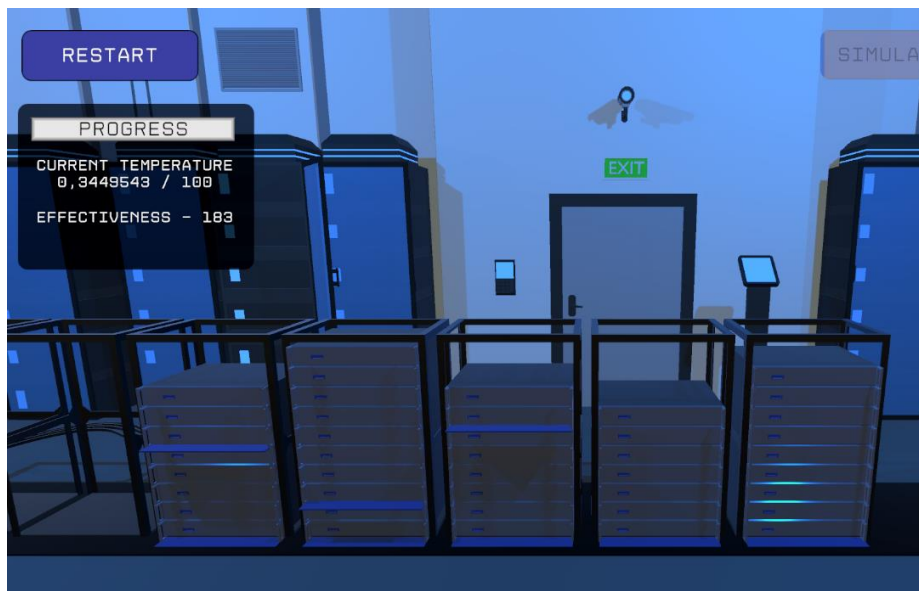


Рисунок 24, Результат SA (низкий коэффициент).

3.2.3. Анализ генетического алгоритма

Рассмотрим генетический алгоритм. При многократных запусках алгоритма можно отметить, что количество итераций и коэффициент мутаций на небольших конфигурациях, создаваемых в приложении, практически не влияют на качество решения, меняя лишь время работы алгоритма. Но этого нельзя сказать про размер популяции. Одним из распространенных способов определения популяции, является формула, предложенная Джоном Холландом (Holland, 1992):

$$N = \frac{(2 * R)}{C},$$

где R – количество итераций, C – коэффициент скрещивания, обычно ~ 0,9. То есть при 50 итерациях нужно иметь около 100 решений одновременно. При слишком низком значении популяции может происходить преждевременная сходимость алгоритма, когда все решения в популяции становятся слишком похожи друг на друга, что затрудняет дальнейшее улучшение их качества. Это прекрасно видно по результатам расстановки серверов на Рисунке 25 GA (низкая популяция) и Рисунке 26 GA (нормальная популяция).

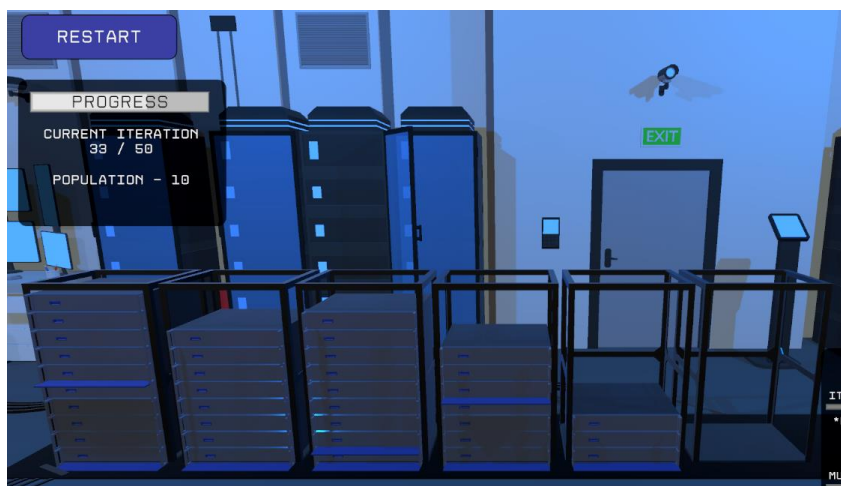


Рисунок 25, GA (низкая популяция)



Рисунок 26, GA (нормальная популяция)

3.3. Удобство использования

3.3.1. Пользовательское тестирование

Стоит отметить, что не все из перечисленных ранее критериев являются объективными. Для составления оценки было проведено тестирование на пользователях.

1. Участники и выборка

С целью получения разнообразных мнений, участниками исследования были студенты из различных факультетов или специальностей. Кроме того, к участию по возможности были призваны студенты разных уровней образования (бакалавриат, магистратура), чтобы учесть различия в опыте и понимании.

Число респондентов составило 11 человек.

2. Продолжительность и задания

Время, отведенное для сбора обратной связи, было однодневным, но могло меняться в зависимости от доступных ресурсов и учебного расписания студентов.

Задания были направлены на оценку трех категорий: понятность, гибкость и наглядность визуализации. Респонденты оценивали визуализацию по пятибалльной шкале и опционально оставляли комментарии.

3.3.2. Финальная оценка

Теперь рассмотрим и проанализируем всё приложение по каждому из перечисленных в разделе 3.1. параметров (Страница 44). Финальный вывод сделан в том числе по собранной в процессе тестирования обратной связи.

1. Эффективность

Так как основная цель визуализации – качественное внешнее представление работы алгоритмов, оценить непосредственное чистое время работы каждого из них не получится, так как необходимы задержки на анимацию и передвижение. С другой стороны, это позволяет наглядно увидеть разницу во времени работы при изменении начальных коэффициентов.

2. Точность

Анализ точности ограничен тем, что рассматриваются решения на небольшом наборе данных. Поэтому сильнее проявляет себя случайность работы алгоритмов SA и GA. Но также важно отметить, что на визуализации отлично прослеживаются необходимые зависимости (от популяции или от охлаждения).

3. Надежность

Система изначально дает сконфигурировать определенный ограниченный набор значений, поэтому может считаться надежной.

4. Понятность

Возможность итерационного просмотра работы и анимации перестановок, замедляющие время работы алгоритмов, позволяют отлично понять и рассмотреть шаги оптимизации. Кроме того, стоит отметить и прогресс-бар, позволяющий легко понять, как долго еще будет работать тот или иной алгоритм.

5. Гибкость

Как было сказано ранее, гибкость приложения ограничена небольшим набором параметров начальной конфигурации, но, с другой стороны,

даже такого набора значений хватает чтобы продемонстрировать качественную разницу.

6. Наглядность

Именно трехмерная визуализация качественно снижает порог вхождения для понимания принципов происходящего. Во введении были рассмотрены разные виды визуализаций, и по опыту использования разработанного приложения, можно сделать вывод что именно по наглядности 3D визуализации сильно опережает любые другие методы.

4. Анализ разработанной визуализации

Данная глава содержит анализ трехмерной визуализации, разработанной на графическом движке Unity Engine, и включает в себя краткий пересказ результатов тестирования, анализ преимуществ и недостатков трехмерных визуализаций и, в частности, разработанного приложения, а также способы дальнейшего развития и улучшения функциональности и интерфейса.

4.1. Анализ результатов тестирования

Ранее проведенное тестирование приложения (см. раздел 3.2.) показало, что реализованные алгоритмы VMP с учетом ограниченного размера конфигурации имеют схожие показатели и зависимости от характеристик с теоретическим описанием. Можно сделать вывод, что визуализация дает корректное представление об их принципе работы, а сравнение результатов оптимизации дает реалистичную картину.

С другой стороны, утверждать о полноте картины и визуализации в частости нельзя, так как в реальных условиях оптимизация включает одновременно тысячи виртуальных машин и на работу оптимизационных алгоритмов могут влиять незначительные параметры, опущенные в данной работе. Например, стоимость электричества в регионе размещения всего или части ЦОД — это вторичный параметр относительно SLA.

4.2. Преимущества трехмерной визуализации

Выделим несколько пунктов, которые можно отнести к разработанному приложению:

1. Лучшее понимание данных

При визуализации алгоритмов размещения виртуальных машин пользователи могут лучше понять, какие ресурсы используются и как они распределены, ведь размерность или потребляемые ресурсы выражены физическим видимым параметром и легко различимы.

2. Более интерактивный опыт

3D визуализация на Unity обеспечивает более интерактивный опыт для пользователей, которые могут взаимодействовать с объектами в трехмерной среде и управлять ими. Это делает работу с данными более простой и удобной. Пользователь сам контролирует процесс, задает параметры и наблюдает за процессом.

3. Улучшение визуальной привлекательности

Посредством трехмерного окружения, плавных анимаций, удобного интерфейса и визуальных эффектов разработанная визуализация делает доносимую информацию более привлекательной и интересной для пользователя. Это приводит к лучшему пониманию и большей вовлеченности.

4. Возможность отображения сложных систем

3D визуализация особенно полезна для отображения сложных систем или процессов, которые могут быть трудно воспроизводимы в других форматах. При визуализации алгоритмов размещения виртуальных машин трехмерная среда может помочь пользователю лучше понять, как эти алгоритмы работают в реальном мире, так как ассоциации с физическими объектами выстраиваются намного лучше, чем с абстрактами фигурами или блок-схемами.

5. Готовность к детальной настройке

Трехмерная визуализация может быть более гибкой и настраиваемой, чем другие форматы визуализации. Это можно использовать для выделения и акцентирования важных моментов, а использование разных настроек графики для разных платформ, предоставляемых графическими движками, позволяет достичь переносимости решения на различные платформы.

4.3. Недостатки трехмерной визуализации

Помимо определённых достоинств, есть и ряд недостатков, присущих как трехмерным визуализациями, так и разработанному приложению в частности:

1. Требования к оборудованию

Любая трехмерная визуализация требует более мощное оборудование, так как использует графические ядра процессора или дискретные графические карты.

2. Возможные проблемы с производительностью

Этот пункт является следствием предыдущего. Современные вычислительные устройства позволяют беспрепятственно воспроизводить видео или stop – motion анимацию, но, к сожалению, не гарантируют бесперебойную работу для полноценных трехмерных графических приложений. Это обязывает указывать, как минимум, системные требования к целевой платформе.

3. Непригодность для определенных задач

Для некоторых задач 3D визуализация может быть не слишком полезной, например, для визуализации больших объемов данных или для простых таблиц с информацией. Мы не можем использовать огромные конфигурации ВМ и физическим стендов, как в реальности, так как, во-первых, пропадет наглядность и доступность, а во-вторых, это потребует на порядок больше вычислительных ресурсов, что лишит нас возможности запуска, к примеру, на web – платформах.

4. Неполная картина происходящего

Несмотря на то, что трехмерная визуализация намного более интерактивна и визуально приятнее, чем статичные схемы и картинки, нельзя забывать, что табличные или другие графические представления могут содержать намного больше информации касательно аспектов работы или сравнительного анализа результатов. Поэтому по разработанной визуализации можно делать выводы об общей картине происходящего, не перекладывая конкретные используемые данные на реальный мир.

4.4. Способы развития и улучшения

В качестве дальнейшего развития можно предложить несколько доработок, исправляющих некоторые недостатки и развивающих систему в целом:

- **Поддержка большего числа алгоритмов**

В приложении можно добавить больше алгоритмов VMP для улучшения его функциональности и гибкости. Приложение уже поддерживает выборность, а архитектура построена таким образом, что каждый алгоритм реализован отдельным модулем. Таким образом, можно беспрепятственно добавлять поддержку многих других алгоритмов.

- **Поддержка интеграции с внешними системами**

На данный момент пользователь сам выбирает и определяет начальную конфигурацию. Так как целевая платформа приложения – Web сайты, получение параметров из внешней базы или системы – органичная функция.

- **Улучшение интерфейса и производительности**

Эти малозначительные улучшения могут сильно повлиять на пользовательский опыт UX при работе с приложением в целом. В интерфейсе можно добавить подсказки и в работе некоторых аспектов,

которые не очевидны, сделать расширенный набор параметров и поддержку большей размерности конфигураций.

- **Новые функции**

Это один из наиболее важных пунктов, ведь он сильнее всего влияет на степень решения изначально поставленной задачи. Вывод статистики использования различных оптимизаций, табличный сравнительный анализ решений и возможность экспорта и импорта полученных решений или начальных конфигураций.

- **Улучшение графики**

Чтобы сделать визуализацию более реалистичной и привлекательной можно повысить общий уровень графики, к примеру отказаться от использования low-poly моделей, сделать light-map и карты нормалей для всех трехмерных моделей.

- **Поддержка большего количества платформ**

Расширить количество поддерживаемых платформ не только по части производительности, но и изменив пользовательский интерфейс для удобного использования, к примеру, на сенсорных экранах.

ЗАКЛЮЧЕНИЕ

1.1. Выполнение поставленных задач

В процессе выполнения выпускной квалификационной работы были решены следующие, поставленные ранее, задачи:

1. Проведено исследование предметной области

Проведен анализ инструментов визуализаций и исследование существующих наиболее популярных оптимизационных алгоритмов размещения виртуальных машин с точки зрения их визуализации. В результате исследования предложены рекомендации по применяемым подходам к визуализации для дальнейшего использования в разработке приложения.

2. Разработан прототип приложения

Для тестирования рекомендованного подхода к визуализации разработан прототип на технологии Unity Engine, а также его web – сборка на WebGL. В процессе разработки прототипа был получен необходимый опыт и сделаны выводы, которые были учтены при разработке полноценного приложения.

3. Разработана трехмерная визуализация принципов работы VMP алгоритмов

На основании анализа предметной области и опыта разработки прототипа, были описаны требования к приложению, описания UI и UX, выбраны наиболее подходящие инструменты разработки и, в конце концов, создано трехмерное приложение для визуализации принципов работы наиболее популярных алгоритмов VMP.

4. Проведено тестирование и анализ разработанной визуализации

Приложение протестировано на различных конфигурациях, выявлены наиболее важные критерии, по которым проведена оценка, в том числе и качество реализации оптимизационных алгоритмов для задачи VMP. Выявлены преимущества и недостатки разработанной визуализации, а также направления будущих улучшений.

1.2. Достижение цели работы

В результате проведённой работы можно с определенной долей уверенности утверждать, что поставленная цель была выполнена –

разработана качественная визуализация, которая обладает необходимыми преимуществами:

1. Понятность и доступность

Позволяет донести сложные идеи и концепции до широкой аудитории. Использование графических элементов и анимации упрощает восприятие информации, делая ее более доступной для людей с различным уровнем знаний и опыта.

2. Интерактивность

Позволяет пользователям взаимодействовать с представленными данными и экспериментировать с ними, что может привести к новым открытиям и повышению эффективности работы.

3. Визуальный анализ

Позволяет проанализировать данные и обнаружить скрытые закономерности, которые могут быть упущены при обычном анализе. Визуализация может помочь выявить аномалии, сделать прогнозы и определить пути улучшения работы системы.

4. Эффективность и экономия времени

Может существенно сократить время, необходимое для анализа и понимания данных. Вместо чтения больших объемов текста пользователь может использовать визуализацию для быстрого ознакомления с данными и выявления главных трендов.

5. Продвижение и презентация

Может быть использована для продвижения и презентации продукта, проекта или идеи. Использование ярких и понятных графических элементов поможет привлечь внимание и запомнить информацию.

1.3. Практическая значимость

Разработка приложения для трехмерной визуализации принципов работы оптимизационных алгоритмов для решения задачи о размещении виртуальных машин имеет практическую значимость для целого ряда отраслей.

В области виртуализации и облачных вычислений размещение виртуальных машин является одной из ключевых задач, которая влияет на эффективность работы облачных систем. Разработанное приложение позволяет визуализировать принципы работы оптимизационных алгоритмов

для решения данной задачи, что может быть полезно для специалистов в области виртуализации и облачных вычислений.

Также данное приложение может использоваться в обучении и воспитании профессиональных навыков студентов и специалистов в области виртуализации и облачных вычислений. Это может помочь повысить квалификацию кадров и улучшить качество работы облачных систем.

Рекомендации, основанные на результате исследования, могут быть следующими:

- Использовать разработанное приложение в обучении студентов и специалистов в области виртуализации и облачных вычислений.
- Использовать разработанное приложение в разработке новых оптимизационных алгоритмов для решения задачи о размещении виртуальных машин.
- Использовать разработанное приложение в процессе тестирования и анализа облачных систем для определения эффективности различных алгоритмов размещения виртуальных машин.

В заключении можно сделать вывод, что проведенное в этой выпускной квалификационной работе исследование и реализованное на его основе приложение имеют реальную практическую значимость и могут быть использованы для решения задач обучения, оптимизации и анализа облачных систем.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Beloglazov, J. A. Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing / Anton Beloglazov, Jemal Abawajy, Rajkumar Buyya // Future Generation Computer Systems. – 2012. – Volume 28. – 755-768 с.
2. Curtis, S. A. The classification of greedy algorithms / Curtis, S. A // Science of Computer Programming. - 2003. - 49. - 125-157 с.
3. Dabiah Ahmed Alboaneen. Metaheuristic approaches to virtual machine placement in cloud computing: a review / Alboaneen, Dabiah Ahmed; Tianfield, Huaglory; Zhang, Yan // 15th International Symposium on Parallel and Distributed Computing (ISPDC). - 2016.
4. Fikru Feleke Moges, S. L. (2019). Energy-aware VM placement algorithms for the OpenStack Neat consolidation framework / Fikru Feleke Moges & Surafel Lemma Abebe // Journal of Cloud Computing. - 2019. - 8.2.
5. Дазарев, Д. Об онлайн-алгоритмах для задач упаковки в контейнеры и полосы, их анализе в худшем случае и в среднем / Д. О. Лазарев, Н. Н. Кузюрин // Труды ИСП РАН. - 2018. - 30. - 209-230 с.
6. Hariharasudhan Viswanathan, E. K. Energy-efficient thermal-aware autonomic management of virtualized HPC Cloud Infrastructure / Ivan Rodero Rutgers, Hariharasudhan Viswanathan, Marc Gamell // Journal of Grid Computing. - 2012. - 10(3). - 447-473 с.
7. Holland, J. Adaptation in Natural and Artificial System / Holland, J. H. // The MIT Press. - 1992.
8. Jonas Mockus, W. E. Bayesian Heuristic Approach to Discrete and Global Optimization / Jonas MOCKUS, William EDDY, Audris MOCKUS // Nonconvex Optimization and Its Applications. - 1997.
9. M Wetter, J. W. Comparison of a generalized pattern search and a genetic algorithm optimization method / Michael Wetter¹ and Jonathan Wright // Eighth International IBPSA Conference. - 2003.
10. Mallawaarachchi, V. Introduction to Genetic Algorithms — Including Example Code / Vijini Mallawaarachchi // Towards Data Science. - 2017.
11. Mikhailyuk, V. A. On the existence of polynomial-time approximation schemes for the reoptimization of discrete optimization problems / V. A. Mikhailyuk // Cybernetics and Systems Analysis. - 2011. - 47. - 368-374 с.

12. P Devarasetty, S. R. Genetic algorithm for quality of service based resource allocation in cloud computing / Prasad Devarasetty, Ch. Satyananda Reddy // Evolutionary Intelligence. - 2021. - 14(2).
13. P. Svärd, W. L. Continuous datacenter consolidation / Petter Svärd, Wubin Li, Eddie Wadbro // IEEE 7th International Conference on Cloud Computing Technology and Science. - 2015.
14. Rania Hassan, B. C. A Comparison of Particle Swarm Optimization and the Genetic Algorithm / Rania Hassan, Babak Cohanin, Olivier de Weck // Structural Dynamics & Materials Conference. - 2005.
15. Rutenbar, R. Simulated annealing algorithms: an overview / Rob. A. Rutenbar // IEEE Circuits and Devices Magazine. - 1989. - 19-26 c.
16. S Jamali, S. M. Improving grouping genetic algorithm for virtual machine placement in cloud data centers. / Shahram Jamali, Sepideh Malektaji // International conference on computer science. - 2014.
17. S.Garg, R. R. GMM-LSTM: a component driven resource utilization prediction model leveraging LSTM and gaussian mixture model. / Sheetal Garg, Rohit Ahuja, Raman Singh, Ivan Perl // Cluster Computing. - 2022.
18. Sourav Kanti Addya. Simulated annealing based VM placement strategy to maximize the profit for Cloud Service Providers / Sourav Kanti Addya, Ashok Kumar Turuk, Bibhudatta Sahoo, Mahasweta Sarkar, Sanjay Kumar Biswash // Engineering Science and Technology, an International Journal. - 2017. -20. - 1249-1259 c.
19. Strunk., A. Costs of virtual machine live migration: A survey / Anja Strunk // IEEE Eighth World Congress. - 2012.
20. Tordsson., L. T. (2014). An autonomic approach to risk-aware data center overbooking / Luis Tomás / IEEE Transactions on Cloud Computing. - 2014. - 2(3).
21. Waskom, M. L. Statistical data visualization / Michael L. Waskom // The Journal of open source software. - 2021. -6.
22. Снитюк, В. Прогнозирование. Модели, методы, алгоритмы / В.Е. Снитюк // Учебное пособие. – К.: «Маклаут». - 2008. - 364 с.

ПРИЛОЖЕНИЕ А

Блоки кода

```
public IEnumerator SortBestFitOne()
{
    if (currentServer >= servers.Length) yield break;
    ui.DisableUI();
    for (var server = currentServer + 1; server < servers.Length; server++)
    {
        serverObjs[server].GetComponent<ServerBlock>().MoveForward(standInbetween);
    }

    var bestFit = -1;
    for (var stand = 0; stand < stands.Length; stand++)
    {
        if (freeSpace[stand] >= servers[currentServer])
        {
            if (bestFit == -1)
                bestFit = stand;
            else if (freeSpace[bestFit] > freeSpace[stand])
                bestFit = stand;
        }
    }

    if (bestFit != -1)
    {
        Vector3 moveTo = standsSpawn.position;
        moveTo.z = serversSpawn.position.z;
        moveTo.x += standInbetween * bestFit;
        mover.MoveServerTo(serverObjs[currentServer].transform, moveTo, 1);
        yield return new WaitForSeconds(1);
        moveTo.y = 0.3f * (standSpace - freeSpace[bestFit]);
        mover.MoveServerTo(serverObjs[currentServer].transform, moveTo, 1);
        yield return new WaitForSeconds(1);
        moveTo.z = standsSpawn.position.z;
        mover.MoveServerTo(serverObjs[currentServer].transform, moveTo, 1);
        yield return new WaitForSeconds(1);

        freeSpace[bestFit] -= servers[currentServer];
        currentServer++;
    }

    ui.EnableUI();
}
```

Блок кода 1, реализация Best Fit на C#

```

public IEnumerator SortFirstFitOne()
{
    if (currentServer >= servers.Length) yield break;
    ui.DisableUI();
    for (var server = currentServer + 1; server < servers.Length; server++)
    {
        serverObjs[server].GetComponent<ServerBlock>().MoveForward(standInbetween);
    }

    for (var stand = 0; stand < stands.Length; stand++)
    {
        if (freeSpace[stand] >= servers[currentServer])
        {
            Vector3 moveTo = standsSpawn.position;
            moveTo.z = serversSpawn.position.z;
            moveTo.x += standInbetween * stand;
            mover.MoveServerTo(serverObjs[currentServer].transform, moveTo,
1);

            yield return new WaitForSeconds(1);
            moveTo.y = 0.3f * (standSpace - freeSpace[stand]);
            mover.MoveServerTo(serverObjs[currentServer].transform, moveTo,
1);

            yield return new WaitForSeconds(1);
            moveTo.z = standsSpawn.position.z;
            mover.MoveServerTo(serverObjs[currentServer].transform, moveTo,
1);

            yield return new WaitForSeconds(1);

            freeSpace[stand] -= servers[currentServer];
            currentServer++;
            break;
        }
    }

    ui.EnableUI();
}

```

Блок кода 2, реализация First Fit на C#

```

public IEnumerator SortSimAnnealingOne()
{
    if (temperature < 1) yield break;
    ui.DisableUI();
    currentServer = servers.Length;
    currentEnergy = GetCurrentEnergy();
    int newEnergy;
    do
    {
        GetRandomNeighbour();
        newEnergy = GetCurrentEnergy();

        var deltaEnergy = newEnergy - currentEnergy;
        if (deltaEnergy < 0)
        {

```

```

        CopyToAnnealing();
        currentEnergy = newEnergy;
    }
    else
    {
        double acceptanceProbability = Mathf.Exp(-deltaEnergy /
temperature);
        if (Random.Range(0.0f, 1.0f) < acceptanceProbability)
        {
            CopyToAnnealing();
            currentEnergy = newEnergy;
        }
    }

    temperature *= 1 - coolingRate;
} while (newEnergy != currentEnergy);

//audioPlayer.PlayOneShot(audioClip2);

for (var stand = 0; stand < stands.Length; stand++)
{
    var crntPos = 0;
    for (var server = 0; server < annealingCrnt[stand]; server++)
    {
        Vector3 moveTo = standsSpawn.position;
        moveTo.x += standInbetween * stand;
        moveTo.y = 0.3f * crntPos;
        serverObjs[sortedAnnealing[stand, server]].transform.position =
moveTo;
        crntPos += servers[sortedAnnealing[stand, server]];
    }
}

yield return new WaitForSeconds(0.5f);
ui.EnableUI();
}

```

Блок кода 3, реализация SA на C#

```

private int GetCurrentEnergy()
{
    int energy = 0;
    for (int stand = 0; stand < stands.Length; stand++)
    {
        energy += (stand + 5) * neighbourSpace[stand];
    }

    return energy;
}

```

Блок кода 4, расчёт текущей энергии

```

private void GetRandomNeighbour()
{
    CopyToNeighbour();
    for (var stand = 0; stand < stands.Length; stand++)
    {
        for (var server = 0; server < neighbourCrnt[stand]; server++)
        {
            if (Random.Range(0.0f, 1.0f) < 0.5f)
            {
                var newStand = Random.Range(0, stands.Length);
                var count = 10;
                while (neighbourSpace[newStand] <
servers[sortedNeighbour[stand, server]] && count > 0)
                {
                    newStand = Random.Range(0, stands.Length);
                    count--;
                }

                if (neighbourSpace[newStand] < servers[sortedNeighbour[stand,
server]])
                    continue;
                sortedNeighbour[newStand, neighbourCrnt[newStand]] =
sortedNeighbour[stand, server];
                neighbourSpace[newStand] -= servers[sortedNeighbour[stand,
server]];
                neighbourCrnt[newStand]++;
                neighbourSpace[stand] += servers[sortedNeighbour[stand,
server]];
                neighbourCrnt[stand]--;
                for (int i = server; i < neighbourCrnt[stand]; i++)
                {
                    sortedNeighbour[stand, i] = sortedNeighbour[stand, i +
1];
                }
            }
        }
    }
    for (var server = 0; server < servers.Length; server++)
    {
        if (notPlaced[server])
        {
            int stand = Random.Range(0, stands.Length);
            int count = stands.Length * 2;
            while (annealingSpace[stand] < servers[server] && count > 0)
            {
                stand = Random.Range(0, stands.Length);
                count--;
            }
            if (annealingSpace[stand] < servers[server])
                continue;
            annealingSpace[stand] -= servers[server];
            sortedAnnealing[stand, annealingCrnt[stand]] = server;
            annealingCrnt[stand]++;
            notPlaced[server] = false;
        }
    }
}

```

Блок кода 5, генерация нового состояния SA

```

public IEnumerator SortGeneticOne()
{
    if (currentIteration >= maxIterations) yield break;
    ui.DisableUI();
    currentServer = servers.Length;

    MutateGeneration();
    CopyBest();

    //audioPlayer.PlayOneShot(audioClip2);

    for (var stand = 0; stand < stands.Length; stand++)
    {
        var crntPos = 0;
        for (var server = 0; server < geneticCrnt[stand]; server++)
        {
            Vector3 moveTo = standsSpawn.position;
            moveTo.x += standInbetween * stand;
            moveTo.y = 0.3f * crntPos;
            serverObjs[sortedGenetic[stand, server]].transform.position =
moveTo;
            crntPos += servers[sortedGenetic[stand, server]];
        }
    }

    currentIteration++;
    yield return new WaitForSeconds(0.5f);
    ui.EnableUI();
}

```

Блок кода 6, реализация генетического алгоритма на C#

```

private void MutateGeneration()
{
    for (int one = Random.Range(0, 1); one < population - 4; one += 4)
    {
        int parent1, parent2;
        int child1, child2;
        if (GetCurrentScore(one) > GetCurrentScore(one + 2))
        {
            parent1 = one;
            child1 = one + 2;
        }
        else
        {
            parent1 = one + 2;
            child1 = one;
        }

        if (GetCurrentScore(one + 1) > GetCurrentScore(one + 3))
        {
            parent2 = one + 1;
            child2 = one + 3;
        }
        else

```



```

        {
            parent2 = one + 3;
            child2 = one + 1;
        }

        RandomMutation(parent1, child1);
        RandomMutation(parent2, child2);
    }
}

private void RandomMutation(int parent, int child)
{
    for (var stand = 0; stand < stands.Length; stand++)
    {
        generationSpace[child, stand] = generationSpace[parent, stand];
        generationCrnt[child, stand] = generationCrnt[parent, stand];
        for (int server = 0; server < standSpace; server++)
        {
            generation[child, stand, server] = generation[parent, stand,
server];
        }
    }

    for (var stand = 0; stand < stands.Length; stand++)
    {
        for (var server = 0; server < generationCrnt[child, stand]; server++)
        {
            if (Random.Range(0.0f, 1.0f) < mutationRate)
            {
                var newStand = Random.Range(0, stands.Length);
                var count = 10;
                while (generationSpace[child, newStand] <
servers[generation[child, stand, server]] && count > 0)
                {
                    newStand = Random.Range(0, stands.Length);
                    count--;
                }

                if (generationSpace[child, newStand] <
servers[generation[child, stand, server]])
                    continue;
                generation[child, newStand, generationCrnt[child, newStand]]
= generation[child, stand, server];
                generationSpace[child, newStand] -= servers[generation[child,
stand, server]];
                generationCrnt[child, newStand]++;
                generationSpace[child, stand] += servers[generation[child,
stand, server]];
                generationCrnt[child, stand]--;
                for (var i = server; i < generationCrnt[child, stand]; i++)
                {
                    generation[child, stand, i] = generation[child, stand, i
+ 1];
                }
            }
        }
    }
}

```

```

    }
}

```

Блок кода 7, генерация мутации

```

private void CopyBest()
{
    var bestId = GetBest();

    for (var stand = 0; stand < stands.Length; stand++)
    {
        geneticSpace[stand] = generationSpace[bestId, stand];
        geneticCrnt[stand] = generationCrnt[bestId, stand];
        for (var server = 0; server < standSpace; server++)
        {
            sortedGenetic[stand, server] = generation[bestId, stand, server];
        }
    }
}

private int GetCurrentScore(int one)
{
    var score = 0;
    for (var stand = 0; stand < stands.Length; stand++)
    {
        score += (stand + 5) * generationSpace[one, stand];
    }

    return score;
}

private int GetBest()
{
    var best = 0;
    var bestId = 0;
    for (var one = 0; one < population; one++)
    {
        var current = GetCurrentScore(one);
        if (current >= best)
        {
            best = current;
            bestId = one;
        }
    }

    return bestId;
}

```

Блок кода 8, расчет приспособленности и продолжение потомства