Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи планування розподілу задач у багатопроцессорних системах

(тема)

Виконав:
здобувач 2 року навчання,
групи СПм-23-5
Олександр ГОЛОВЧЕНКО
(власне ім'я, прізвище)

Спеціальність
123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма
Системне програмування
(повна назва освітньої програми)

Керівник: доц. Дмитро ГОЛУБНИЧИЙ
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Головченку Олександру Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи планування розподілу задач у багатопроцесорних системах

затверджена наказом по університету від " 21 " квітня 2025 р. № 296 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 16 червня 2025 р.

3. Вхідні дані до роботи 1) концепція багатопроцесорної мережі; 2) базові концепції лінійного програмування з булевими змінними; 3) принципові ідеї рангового підходу

4. Перелік питань, що потрібно опрацювати у роботі

1) Балансування навантаження у багатопроцесорних системах: сучасні підходи та математичні моделі;

2) Використання рангового підходу при вирішенні задачі цілочисельного лінійного програмування з булевими змінними;

3) Алгоритми методу відсікань неперспективних варіантів;

4) Програмна реалізація експериментального застосунку;

5) Результати Експериментальних досліджень.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд-презентація – 18 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Розробка алгоритмів методу відсікань	22.04.25-29.04.25	
2	Розробка програмних моделей алгоритмів	30.04.25-05.05.25	
3	Проведення експериментів	06.05.25-09.05.25	
4	Порівняльне тестування ефективності розроб-	10.05.25-21.05.25	
	лених алгоритмів в умовах наближених до		
	реальних		
5	Проведення експериментів	22.05.25-02.06.25	
6	Оформлення матеріалів кваліфікаційної роботи	03.06.25-05.06.25	
7	Подання кваліфікаційної роботи керівникові та	06.06.25-09.06.25	
	її попередній захист		
8	Подання кваліфікаційної роботи на рецензу-	10.06.25-12.06.25	
	вання		

Дата видачі завдання " 21 " квітня 2025 р.

Здобувач Головченко
(підпис)

Керівник роботи (підпис) доц. Дмитро ГОЛУБНИЧИЙ
(посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 87 с., 18 рис., 10 табл., 1 дод., 22 джерела.

АЛГОРИТМ ПОШУКУ, БАЛАНСУВАННЯ НАВАНТАЖЕННЯ, ВЕКТОР, ПРОГРАМНА МОДЕЛЬ, РАНГ, СТРАТЕГІЯ ВІДСІКАННЯ.

Метою даної кваліфікаційної роботи є розроблення та порівняльний аналіз ефективності наближених алгоритмів, що базуються на ранговому методі, для вирішення задачі балансування навантаження. Ця задача може бути формалізована у вигляді задачі цілочисельного лінійного програмування з булевими змінними (ЦЛП з БЗ).

Об’єктом дослідження є алгоритми пошуку рішень, що використовують різні підходи до реалізації стратегій відсікання, а також аналіз їх впливу на загальну продуктивність програмної реалізації.

Предметом дослідження виступає ранговий метод та його застосування у процесі оптимального розподілу навантаження в багатопроцесорних обчислювальних системах.

У ході виконання кваліфікаційної роботи було оглянуто сучасні рішення в сфері балансування навантаження в мережі, наведено доказ зводимості даного завдання до вирішення класичної задача "Про ранець 0-1" і, відповідно, приведено огляд деяких відносно нових рішень даної задачі.

Запропонована програмна реалізація деяких наближених алгоритмів пошуку рішення задачі "Про ранець 0-1", заснованих на моделі рангового підходу зі скороченням кількості аналізованих векторів рішення за методом відсівання безперспективних варіантів. Виявлено потенційні вузькі місця вже відомих стратегій відсікання, та запропоновано модифіковану версію однієї з них.

ABSTRACT

Master’s thesis: 87 pages, 18 figures, 10 tables, 1 appendice, 22 sources.

CUT-OFF STRATEGY, LOAD BALANCING, PROGRAM MODEL, RANK, SEARCH ALGORITHM, VECTOR.

The major goal of this thesis is to develop and comparative analysis of the effectiveness of approximate algorithms based on the rank method for solving the load balancing problem. This problem can be formalized as an integer linear programming problem with Boolean variables (ILP with BV).

The object of the study is solution search algorithms that use different approaches to implementing cutoff strategies, as well as an analysis of their impact on the overall performance of the program implementation.

The subject of the study is the rank method and its application in the process of optimal load distribution in multiprocessor computing systems.

During the qualification work, modern solutions in the field of network load balancing were reviewed, a proof of the reducibility of this problem to the solution of the classical problem "About the knapsack 0-1" was provided, and, accordingly, some relatively new solutions to this problem were reviewed.

A software implementation of some approximate algorithms for finding a solution to the problem "About the knapsack 0-1" was proposed, based on the rank approach model with a reduction in the number of analyzed solution vectors by the method of screening out unpromising options. Potential bottlenecks of already known cut-off strategies were identified, and a modified version of one of them was proposed.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БЗ – булеві змінні

ДО – дискретна оптимізація

ПМ – процесорний модуль

ЦЛП – цілочисельне лінійне програмування

ABC – алгоритм штучної бджолиної колонії (англ., Artificial Bee Colony)

ABHS – гібридна система штучної бджолиної колонії (англ., Artificial Bee Hybrid System)

AMP – асиметрична багатопроцесорна обробка (англ., Asymmetric Multiprocessing)

AMTHA – автоматичне розміщення задач на гетерогенних архітектурах (англ., Automatic Mapping Task on Heterogeneous Architectures)

BABC-DE – бінарна штучна бджолина колонія з диференціальною еволюцією (англ., Binary Artificial Bee Colony – Differential Evolution)

BFPA – бінарний алгоритм запилення квітів (англ., Binary Flower Pollination Algorithm)

BHLSO – гібридний ройовий алгоритм з поведінковими механізмами (англ., Behavioral Hybrid LSO)

CS – пошук зозулі (англ., Cuckoo Search)

DE – диференціальна еволюція (англ., Differential Evolution)

DLS4LB – динамічне планування навантаження для балансування (англ., Dynamic Load Scheduling for Load Balancing)

eLaPeSD – покращене планування з урахуванням навантаження і продуктивності (англ., Enhanced Load-aware Performance Scheduling and Distribution)

ER-LS – покращене ранжування з плануванням у вигляді списку (англ., Enhanced Ranking with List Scheduling)

GA – генетичний алгоритм (англ., Genetic Algorithm)

HEFT – Неоднорідний найбільш ранній час закінчення (англ., Heterogeneous Earliest Finish Time)

HPC – високопродуктивні обчислення (англ., High-Performance Computing)

JSQ – приєднання до найкоротшої черги (англ., Join the Shortest Queue)

KP01 – задача про рюкзак 0-1 (англ., Knapsack Problem 0-1)

MBLSO – модифікований бінарний ройовий алгоритм (англ., Modified Binary LSO)

MBO – оптимізація метелика-монарха (англ., Monarch Butterfly Optimization)

MPAHA – модель паралельних алгоритмів для гетерогенних архітектур (англ., Model of Parallel Algorithms on Heterogeneous Architectures)

NP – задача класу неполіноміальної складності (англ., Non-deterministic Polynomial time)

QoS – якість обслуговування (англ., Quality of Service)

SED – найменша очікувана затримка (англ., Shortest Expected Delay)

SMP – симетрична багатопроцесорна обробка (англ., Symmetric Multiprocessing)

ВСТУП

Під багатопроцесорними системами у загальному випадку розуміють обчислювальні системи, в яких для обробки даних та виконання програм використовується більше одного процесора. Таким чином забезпечується можливість паралельної обробки данних або виконання різних завдань різними процесорами, чим підвищується продуктивність та ефективність системи. Однак, такий підхід породжує проблему розподілу робочого навантаження.

Відповідний розподіл робочого навантаження по різних оброблювальних елементах дуже важливий, оскільки непропорційні робочі навантаження можуть звести нанівець перевагу продуктивності.

Вирішення завдання планування розподілу задач у мережі, на основі моделі ЦЛП з БЗ полягає у знаходженні, відмовостійкого та ефективного способу розподілу ресурсів мережі. Ранговий підхід має потенціал, що дозволяє враховувати як обмеженість ресурсів, так і необхідність резервування для підвищення надійності роботи інфокомунікаційних систем та мереж.

Актуальність роботи зумовлена широким розповсюдженням різних варіацій багатопроцесорних систем (сервери, дата-центри, суперкомп’ютери, багатоядерні процесори, комп’ютерні мережі), а також складністю, а точніше NP-повнотою задачі до якох задача балансування зводиться.

Метою даної кваліфікаційної роботи є розроблення та порівняльний аналіз ефективності наближених алгоритмів, що базуються на ранговому методі, для вирішення задачі балансування навантаження. Ця задача може бути формалізована у вигляді задачі цілочисельного лінійного програмування з булевими змінними (ЦЛП з БЗ).

Об’єктом дослідження є алгоритми пошуку рішень, що використовують різні підходи до реалізації стратегій відсікання, а також

аналіз їх впливу на загальну продуктивність програмної реалізації.

Предметом дослідження виступає ранговий метод та його застосування у процесі оптимального розподілу навантаження в багатопроцесорних обчислювальних системах.

Запропонована програмна реалізація деяких наближених алгоритмів пошуку рішення задачі "Про ранець 0-1", заснованих на моделі рангового підходу зі скороченням кількості аналізованих векторів рішення за методом відсівання безперспективних варіантів. Виявлено потенційні вузькі місця вже відомих стратегій відсікання, та запропоновано модифіковану версію однієї з них.

Експериментальні дослідження виявили, що використання запропонованих алгоритмів дозволяє значно скоротити кількість проаналізованих векторів при цьому похибка відносно точного рішення становить не більше 10%.

Порівняння з іншими алгоритмами по середньому завантаженню процесорів в мережі показало, що запропонований алгоритм показує хороші результати утримуючи завантаження в межах 90-98%.

1 БАЛАНСУВАННЯ НАВАНТАЖЕННЯ У БАГАТОПРОЦЕСОРНИХ СИСТЕМАХ: СУЧАСНІ ПІДХОДИ ТА МАТЕМАТИЧНІ МОДЕЛІ

1.1 Багатопроцесорні системи. Сутність та використання

Багатопроцесорні системи, тобто обчислювальні системи, в яких для обробки даних та виконання програм використовується більше одного процесора можна класифікувати за різноманітними ознаками і таким чином виділити:

а) симетричні (SMP, Symmetric Multiprocessing);

1) всі процесори мають рівні права та доступ до спільної пам'яті;

2) операційна система рівномірно розподіляє завдання між процесорами;

б) асиметричні (AMP, Asymmetric Multiprocessing);

1) один головний процесор керує іншими;

2) використовувалися в ранніх багатопроцесорних системах, але зараз зустрічаються рідко;

в) системи із загальною пам'яттю (Shared Memory Multiprocessing);

1) всі процесори звертаються до однієї спільної пам'яті;

2) Можливі конфлікти доступу, тому використовуються механізми синхронізації;

г) системи з розподіленою пам'яттю (Distributed Memory Multiprocessing);

1) у кожного процесора є своя локальна пам'ять, а обмін даними між процесорами відбувається через мережу;

2) використовується в суперкомп'ютерах та великих кластерах.

Класичними прикладами багатопроцесорних систем є:

- сервери;
- дата-центри;

- суперкомп’ютери (IBM Summit, Fugaku);
- багатоядерні процесори;
- комп’ютерні мережі.

Однак, паралельне вирішення завдань так само як і вирішення множини завдань багатопроцесорною системою породжує проблему розподілу робочого навантаження.

Розподіл робочого навантаження відомий, також, як балансування навантаження. Відповідний розподіл робочого навантаження по різних оброблювальних елементах дуже важливий, оскільки непропорційні робочі навантаження можуть звести нанівець перевагу продуктивності. Отже, балансування навантаження у паралельних системах є критично важливою та складною діяльністю [1].

1.2 Класифікація алгоритмів розподілу задач та балансування навантажень в багатопроцесорних системах

Через свою ефективність багатопроцесорні системи набули широкого розповсюдження, а разом із тим перейняли і тенденцію до постійного розвитку. Чим спровокували постійну розробку нових алгоритмів розподілу та балансування завдань. Постійний потік ідей в цьому напрямку спровокований також і математичною складністю даної задачі, яку буде доведено пізніше, в підпункті 1.4. Складність задачі розподілу навантаження на теперішній час не дозволяє знайти рішення, яке б забезпечувало достатньо рівномірну завантаженість за гарантований час. Таким чином наразі існують десятки різноманітних методів вирішення задачі балансування, які можуть бути класифіковані різним чином в залежності від математико-технічних основ, що лежать у їх основі або і від особистої думки автора, що проводить класифікацію. Деякі з методів в силу своєї унікальності або крайньої гібридизованості класифікації не підлягають.

Тим не менш можна виділити деякі основні напрямки більш

стандартних методів вирішення означеної проблеми, за ознаками.

За способом (часом) прийняття рішення можна виділити:

- статичні алгоритми: Розподіл завдань між процесорами відбувається на етапі компіляції чи початку виконання програми. В таких алгоритмах припускається, що інформація про завдання та ресурси відома заздалегідь;
- динамічні алгоритми: Розподіл завдань відбувається під час виконання програми, що дозволяє адаптуватися до умов системи, що змінюються. Так як коливання навантаження або відмова окремих вузлів. Дослідження у [1] показують що, динамічні алгоритми можуть значно покращити продуктивність за умов невизначеності;
- адаптивні алгоритми розподілу: Принцип заснований на комбінуванні статичного та динамічного підходів. Початковий розподіл фіксований, але під час роботи можливий перерахунок навантаження, що дозволяє уникнути проблем обох підходів;
- розподіл на основі пріорітетів: Засновано на призначенні завдань залежно від їхньої важливості. Важливі завдання одержують більше ресурсів. Такий підхід використовується в системах реального часу та серверах із пріоритетним обслуговуванням.

За способом курування алгоритми розподілу навантаження бувають:

- централізовані алгоритми: Існує єдиний керуючий вузол, який відповідає за розподіл усіх завдань. Такий підхід може призвести до так званої “вузької шийки” в системі та зниження відмовостійкості;
- децентралізовані (розподілені) алгоритми: Кожен вузол системи самостійно приймає рішення про прийняття та виконання завдань, що підвищує масштабованість та надійність системи. Але може знизити загальну продуктивність, оскільки вузли не завжди враховують завантаження інших вузлів.

За врахуванням гетерогенності системи можна виділити дві групи алгоритмів:

- алгоритми для однорідних систем: Передбачають, що всі процесори

мають однакову продуктивність та характеристики;

- алгоритми для гетерогенних систем: Враховують відмінності у продуктивності, архітектурі та інших характеристиках процесорів, що дозволяє ефективніше розподіляти завдання з урахуванням можливостей кожного вузла [2].

Окремо також можна виділити:

- аргоритми балансування навантаження через граф задач (Task Graph Scheduling): Заснований на поданні задач у вигляді графа, де вузли – це завдання, а ребра – залежності між ними. Його сильною стороною є те, що він дозволяє враховувати залежності між завданнями та завантаженість процесорів;

- глобальний динамічний розподіл (Work Stealing): Якщо процесор завершив свої завдання, він "краде" роботу у перевантаженого процесора. Такий підхід можна вважати різновидом децентралізованого підходу. Він дозволяє рівномірно розподіляти навантаження без центрального керування;

- енергозберігаючий розподіл (Energy-aware Scheduling): Використовується переважно у мобільних хмарних системах. Основна ідея полягає в тому, що завдання розподіляються з урахуванням споживання енергії так щоб мінімізувати споживання. Часто робота таких алгоритмів пов'язана із регулюванням тактових частот.

Приведену класифікацію можна зобразити графічно у вигляді дерева як показано на рисунку 1.1.

1.3 Сучасні дослідження та рішення в області розподілу задач та балансування навантажень в багатопроцесорних системах

Як вже було сказано на сьогоднішній день існує безліч алгоритмів та стратегій вирішення завдань розподілу та балансування навантаження у багатопроцесорних системах. У цьому пункті будуть коротко розглянуті деякі дослідження в цій галузі, які проводилися за останнє десятиліття. Ці

дослідження можуть бути основою розробки нових ефективних алгоритмів у цій галузі, або, щонайменше, давати уявлення про сучасні підходи до розподілу завдань у багатопроцесорних системах.

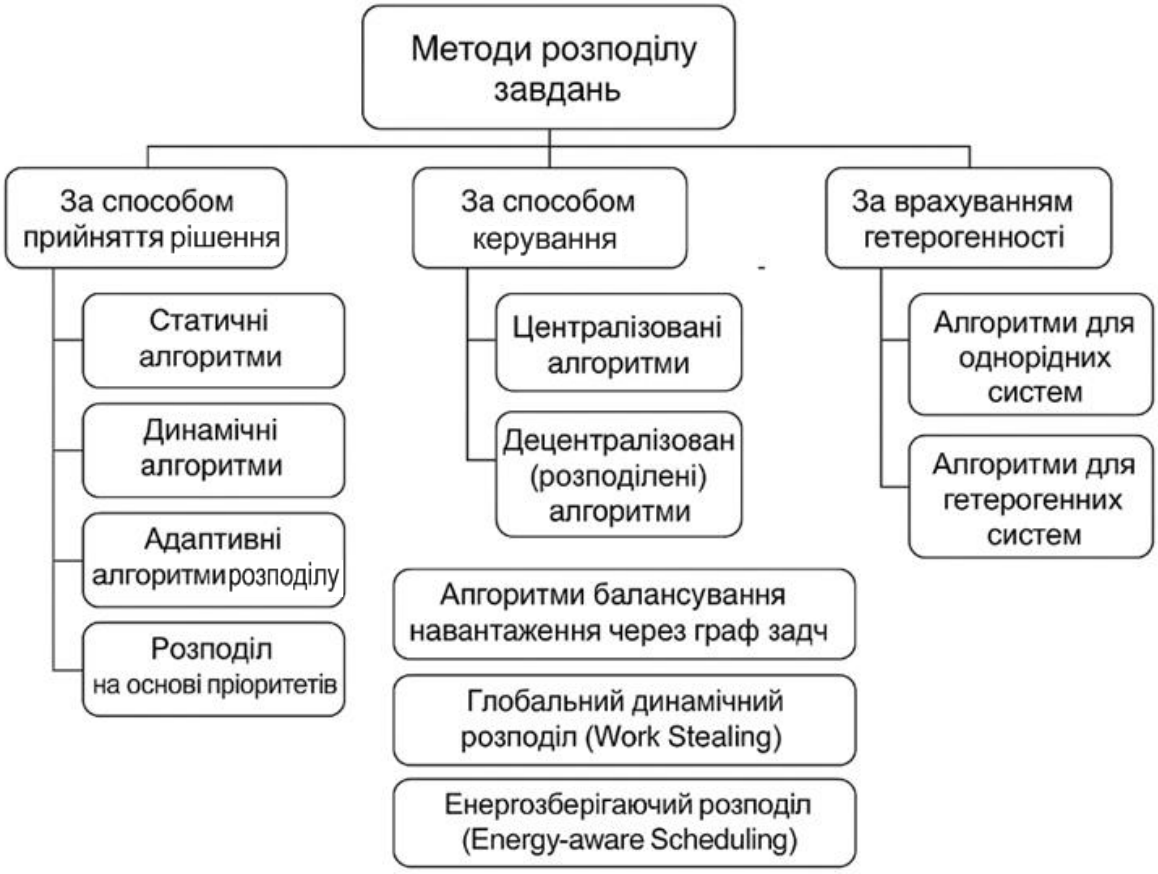


Рисунок 1.1 – Класифікація методів розподілу навантаження у багатопроцесорних системах

Так, наприклад, автори [1] А. Мандал та С. Пал у своєму дослідженні аналізують різні техніки динамічного балансування навантаження, що відносяться як до централізованих так і до розподілених алгоритмів. Наприкінці автори роблять висновок, що за умови, що допускається обмежена масштабованість, централізовані, динамічні методи балансування навантаження забезпечують кращу продуктивність у порівнянні з розглянутими альтернативами, оскільки не споживають накладні витрати пов'язані з обміном профілями. Однак автори також вказують, що якщо

система потребує складної масштабованості, некооперативні розподілені методи динамічного балансування навантаження зазвичай забезпечують краще рішення, оскільки накладні витрати під час обміну інформацією про профілі обмежені в порівнянні з іншими методами в цій групі.

У статті [3] запропоновано алгоритм призначення завдань у неоднорідному обчислювальному середовищі (AMTHA), який дозволяє ефективно розподіляти обчислення між доступними процесорними ядрами з урахуванням комунікації між ними. Також розглядається модель, яка описує виконання паралельних алгоритмів у такому середовищі – MPAHA.

Автори наводять результати порівняльного тестування, щодо реального часу виконання на багатокристальних системах з використанням результатів проботи AMTHA. Показано, що зі зростанням обсягу комунікації (або розміру пакетів, що передаються) між завданнями помилка збільшується, а окрім цього значення помилки залежить від обсягу даних кешу на кожному ядрі. Так, у тестах з вісьмома ядрами помилка була в межах 4%, але при використанні 64 ядер вона зростала до 6% (при тому, що навантаження на обробку значно перевищувало навантаження на комунікацію).

Наприкінці аналізується адаптованість аналізованих моделі та алгоритму до багатокристальних кластерних архітектур. Алгоритм AMTHA був протестований на двох різних багатокристальних архітектурах, продемонструвавши здатність з високою точністю (помилка менше 6%) передбачати час виконання додатків.

Автори М. Амарис, Дж. Лукареллі, К. Моммессін і Д. Трістрам в своїй роботі [2] вивчають проблему виконання додатку, представленого графом завдань пріоритету на паралельній машині, що складається зі стандартних обчислювальних ядер та прискорювачів. Особливістю їх підходу є те, що вони розрізняють фази розподілу та планування і фокусуються на частині розподілу завдань між різнорідними обчислювальними елементами у заздалегідь відомих умовах, і в динамічному середовищі з поступовим надходженням нових задач.

Для варіанту де умови відомі завчасно авторами розглянуто існуючий алгоритм, в основі якого – лінійне програмування, встановили нижні межі його продуктивності після чого покращили фазу планування, замінивши жадібне планування на більш ефективне. Новий алгоритм зберіг той самий ступінь апроксимації, що й попередній метод.

Для динамічно змінюваного середовища розроблено перший алгоритм, що враховує залежності між завданнями – ER-LS. Запропонований алгоритм є збалансованим в контексті якості розв’язку та витрат ресурсів. Експерименти показали, що запропоновані методи перевершують класичний HEFT-алгоритм. Зокрема, алгоритм ER-LS у середньому працює краще, ніж Greedy (на 8-36% залежно від завдання, а в одному випадку – в 11 разів), але поступається EFT, який у середньому знижує час виконання завдань на 11% порівняно з ER-LS, а в деяких випадках – до 60%. Однак, незважаючи на це, ER-LS є більш передбачуваним у гіршому випадку. У майбутньому автори планують враховувати витрати на комунікацію між обчислювальними блоками.

Вже представлені наукові роботи вийшли до 2018 року, проте дослідження у цій сфері продовжуються і зараз.

У квітні 2024 року Міжнародна група дослідників із США та Франції представили статтю: «A Communication- and Memory-Aware Model for Load Balancing Tasks» [4]. Автори пропонують інноваційну модель, що поєднує обчислення, комунікацію та пам'ять для оптимізації розподілу завдань у розподілених системах. Представлений повністю розподілений алгоритм оптимізації балансування навантаження, що демонструє результати близькі до оптимальних та дозволяє оптимізатору досліджувати складні ситуації при розміщенні завдань, такі як підвищений паралелізм за рахунок реплікації даних, що збільшує використання пам'яті.

Запропонований розподілений алгоритм оптимізації балансування навантаження на основі евристики швидко знаходить близькі до оптимального рішення. Метод був протестований при застосуванні

електромагнітного коду, забезпечивши підвищення швидкості виконання при незбалансованому розподілі навантаження до 2,3 рази. Автори підкреслюють важливість порівняння евристичних рішень із доказово оптимальними, за для чого завдання оптимізації формалізовано як завдання змішаного цілочисельного лінійного програмування.

В статті [5], за авторством О. Раторе, А. Басдена, Н. Чанцелора и Х. Касумаатмаджі, розглядається використання квантового відпалу (QA) для балансування навантаження у паралельних додатках високопродуктивних обчислень (HPC – high-performance computing). Авторський підхід застосовується до двох типових алгоритмів HPC: адаптивного уточнення сітки та згладженої гідродинаміки частинок. Хоча методика отримання даних для симуляції залежить від програми, запропонований протокол балансування навантаження залишається універсальним. У контексті сітки квантовий відпал демонструє переваги в порівнянні з класичними методами, такими як круговий протокол, але не має явної переваги перед більш складними методами, такими як найшвидший спуск або симульований відпал, незважаючи на свою конкурентоспроможність. Основною проблемою масштабованості є обмежене з'єднання у поточному квантовому устаткуванні. У разі більш складного завдання на основі частинок, що розглядається як багатозадачна оптимізація, рішення квантового відпалу продемонстрували перевагу якості рішень над сучасними класичними методами, що значно покращує ефективність використання процесорів.

У висновку зазначається, що використання квантового відпалу для розподілу навантаження в паралельних додатках HPC показало поліпшення в порівнянні з більш простими класичними методами, хоча воно не перевершило більш складні класичні підходи. Істотною проблемою для квантового відпалу є обмежений зв'язок між фізичними кубітами, що є критичним бар'єром для масштабованості. Однак у контексті завдання на основі частинок, де граф проблеми не завжди повністю пов'язаний, квантовий відпал стає більш застосовним. Очікується, що з розвитком

квантових відпалювачів вони стануть придатними для поділу графів, що може призвести до появи гібридних архітектур, що використовують як класичні, так і квантові обчислювальні ресурси.

В роботі [6] розглядається проблема балансування навантаження у сучасних центрах обробки даних та хмарних сервісах, які використовують гетерогенні сервери. Авторський підхід формулює це завдання як стохастичну оптимізацію та пропонує ефективний алгоритм на основі детального математичного аналізу. Проведені симуляції показують, що запропоноване рішення перевищує існуючі методи як за якістю розподілу навантаження, так і обчислювальної ефективності.

Авторами підкреслюється актуальність проблеми для розподілених обчислювальних систем та необхідність враховувати їхню гетерогенність. Запропонований алгоритм SCD спеціально адаптований до таких умов та показує кращі результати порівняно із сучасними методами балансування навантаження. Він працює за час O(n log n), що робить його застосовним у реальних системах, не поступаючись традиційним підходам, таким як JSQ та SED.

У роботі під заголоком Two-level Dynamic Load Balancing for High Performance Scientific Applications [7] автори досліджують вплив дисбалансу навантаження на продуктивність наукових додатків, вказуючи, що хоча системи високопродуктивних обчислень (HPC) стали більшими і складнішими, пропонуючи паралелізм на декількох рівнях, більшість існуючих методів балансування навантаження не вирішують проблему дисбалансу навантаження на декількох рівнях одночасно.

Хоча сама робота більше націлена на дослідження впливу дизбалансу на продуктивність системи, проте в роботі також пропонується використовувати новий дворівневий підхід до динамічного балансування, що враховує потоки та процеси. Пропонований дворівневий підхід до балансування навантаження з використанням DLS4LB та eLaPeSD є універсальним і може застосовуватись до будь-якого додатку MPI+OpenMP.

Результати показують, що певні поліпшення продуктивності можуть бути досягнуті за допомогою однорівневого балансування навантаження на будь-якому рівні, однак найкраща продуктивність програми може бути досягнута тільки шляхом вирішення проблеми дисбалансу навантаження спільно на двох рівнях (підвищення продуктивності програми до 21%).

Було помічено, що найкраща дворівнева комбінація не завжди є комбінацією двох найкращих методів DLS за продуктивністю тільки на одному рівні.

1.4 Задача 0-1 рюкзак як класичний приклад задачі ЦЛП з БП

Пізніше у пункті 1.5 буде доведено, що завдання розподілу навантаження у багатопроцесорній системі зводиться до задачі ЦЛП з БЗ. Тому логічно спочатку розглянути математичну модель завдання ЦЛП з БЗ.

Цілочисельне лінійне програмування (ЦЛП) – це різновид лінійного програмування (ЛП), у якому змінні можуть набувати лише цілих значень. Зокрема, якщо змінні обмежені значеннями 0 або 1, таке завдання називається булевою (бінарною, з бінарними змінними) цілою лінійною програмою (0-1 ЦЛП, або ЦЛП з БЗ).

Задачею ЦЛП з БЗ є у знаходження певного набору значень бінарних змінних x_i, які:

- максимізують чи мінімізують лінійну цільову функцію:

$$\max/\min f(x) = c_1 \times x_1 + c_2 \times x_2 + ... + c_n \times x_n,$$

де c_i – задані коефіцієнти;

- задовольняють системі лінійних обмежень виду:

$$a_{11} \times x_1 + a_{12} \times x_2 + ... + a_{1n} \times x_n \le b_1,$$
$$a_{21} \times x_1 + a_{22} \times x_2 + ... + a_{2n} \times x_n \le b_2,$$
$$\vdots$$
$$a_{m1} \times x_1 + a_{m2} \times x_2 + ... + a_{mn} \times x_n \le b_m,$$

де a_{ij} та b_i – задані коефіцієнти.

Орієнтуючись на дане визначення, можна дійти висновку, що задача 0-1 ранець, постановка якої буде приведена далі – це окремий випадок задачі ЦЛП з БЗ.

Задача knapsack problem 0-1 є однією із найвідоміших задач в галузі дискретної оптимізації. Вона привертає увагу дослідників ще з моменту свого формулювання у 1972 році, і активно вивчається з точки зору теорії та практичних методів розв'язку. Така популярність зумовлена широким спектром прикладних задач, які можуть бути зведені до цієї моделі – від розподілу ресурсів у мережі до завантаження транспортних засобів.

Постановка даної задачі в загальному вигляді є наступною. Знайти певний бінарний вектор $\vec{x}$, для якого наступна функція набуває максимального значення:

$$f(\vec{x}) = \sum_{j=1}^{n} c_j \times x_j, \tag{1.1}$$

за умови, що виконуються:

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i, \tag{1.2}$$

$$x_j \in \{0,1\}, \qquad i = \overline{(1,m)}; \qquad j = \overline{(1,n)}. \tag{1.3}$$

З метою формального опису розглянемо одномірну постановку задачі,

в якій необхідно максимізувати цільовий функціонал:

$$f(\vec{x}) = c_1 x_1 + c_2 x_2 + ... + c_n x_n, \tag{1.4}$$

за обмежень:

$$\sum_{j=1}^{n} a_{1j} x_j \le b, \tag{1.5}$$

$$c_1 \ge c_2 \ge ... \ge c_n; \qquad a_{ij} > 0; \qquad c_j > 0 \qquad j = \overline{(1,n)}. \tag{1.6}$$

Є сенс зауважити, що на даний момент наукова спільнота не дійшла єдиної думки про співвідношення класів P і NP. Цей аспект суттєвий, оскільки задача 0-1 Рюкзак є NP-повною: її розв'язок можна перевірити за поліноміальний час, але якщо P ≠ NP не існує алгоритму, який завжди знаходить точне рішення за поліноміальний час. У протилежному випадку (якщо P = NP), існування такого алгоритму є можливим. Ця невизначеність залишає простір для подальших досліджень у сфері пошуку оптимальних рішень, оскільки будь-який запропонований алгоритм потенційно може стати тим, що вирішить питання про належність завдання до класу P.

1.5 Зведення задачі розподілу навантаження в мультипроцесорних системах до задачі цілочисельного лінійного програмування з булевими змінними

Використання математичної моделі ЦЛП з БЗ в контексті розв'язку завдання розподілу задач у мережі, за умови забезпечення відмовостійкості має декілька особливостей [8-14]:

- використання двійкових змінних: у задачах цілочислового лінійного програмування з бінарними змінними рішення подається у вигляді «0» або «1», що відображає виключення або включення певного завдання чи ресурсу

з процесу планування;

- формалізація резервування: використання двійкових змінних дає змогу чітко задавати резервування ресурсів у мережі вже на етапі побудови моделі;

- урахування вимог до надійності: завдяки системі лінійних обмежень у моделі ЦЛП можна реалізувати політику надмірності, для забезпечення стійкості мережі до відмов;

- використання ресурсів: у моделі ЦЛП можуть задаватися обмеження до ресурсів на кшталт доступної кількості вузлів чи пропускної здатності. Такий підхід забезпечує більш раціональне використання ресурсів, та забезпечити рівномірне навантаження;

- планування розподілених обчислень: модель ЦЛП з двійковими змінними дозволяє ефективно визначати, як розподілити задачі в межах мережі з урахуванням часу обробки, затримок і доступності ресурсів. Булевими змінними зручно описувати ситуації, коли завдання закріплюється за певним вузлом, або не виконується зовсім;

- QoS: Розширивши модель ЦЛП кількома додатковими обмеженнями, можна домогтися включення вимог якості обслуговування (QoS);

- Комбінаторна складність: запровадження булевих змінних надає моделі комбінаторного характеру, як наслідок задача потрапляє до класу NP-складних, тобто зі збільшенням кількості змінних обчислювальна складність стрімко підвищується, і для вирішення задач такого масштабу доцільно використовувати евристичні методи чи наближені алгоритми.

- аналіз надмірності: використання булевих змінних дає змогу моделювати надлишковість мережі, зокрема визначати, які вузли або канали можуть бути вимкнені до появи збоїв функціонування системи;

- гнучкість моделювання: булеві змінні у ЦЛП дають змогу формулювати як цільову функцію, так і обмеження з урахуванням витрат, відмовостійкості, безпеки та розміщення завдань у мережі.

Ефективність процесу управління розподілом задач у мережі

оцінюється за допомогою показників, наведених у [13].

Коефіцієнт функціональної потужності мережі:

$$E_\nu(\vartheta_\mu^i) = \sum_{i=1}^{n_\nu} \sum_{\mu=1}^{m_i} \beta_{\mu i} ,$$

де n_ν – кількість процесорних модулів (ПМ) у мережі, чий стан S_ν;

m_i – кількість завдань, яку і-й ПМ у стані S_ν здатен вирішувати;

$\beta_{\mu i}$ – коефіцієнт μ-ї задачі в і-му процесорному модулі, який визначає її важливість/пріоритет для системи.

Загальний час, необхідний для отримання даних у мережі, визначається конфігурацією розміщення сегментів бази даних на фізичних вузлах.

$$T = \sum_{i=1}^{m} \sum_{j=1}^{n} K_j p_i \tau_{ij} x_{ij} ,$$

де $\tau_{ij} = \dfrac{V_i^{(b)}}{\lambda_{ij}}$ – середній час на вибірку 1 кілобайта інформації з і-го фрагмента, що зберігається на j-му вузлі;

λ_{ij} – коефіцієнт швидкості вибірки та обробки одного мегабайтуб даних при звернennв до і-го фрагмента БД на вузлі j;

$V_i^{(b)}$ – обсяг зовнішньої пам’яті, потрібний для збереження *i*-го фрагмента бази даних;

p_i – характеристика, що відображає частоту доступу до *i*-го фрагмента БД під час функціонування системи управління базами даних (СУБД);

K_j – коефіцієнт швидкості доступу до даних на j-му вузлі мережі;

$T_в$ – середній час, необхідний для відновлення працездатності мережі після збою.

$$x_{ij} = \begin{cases} 1, \text{якщо фрагмент БД з номером } i \text{ розміщений у } j-\text{му вузлі мережі;} \\ 0, \text{ у протилежному випадку;} \end{cases};$$

Нехай, задана система з n управляючий процесорний модуль (ПМ) M_i, де $i = \overline{(1,n)}$, кожний з яких керує об'єктами Oε. Вважатимемо незалежними збої системи зв'язку і відмови ПМ. Станом системи s(t) в момент часу t називатимемо множину станів усіх модулів у момент t, тобто $s(t)=\sigma_1,\ldots,\sigma_n$, де $\sigma_i \in \{0,1\}$ та

$$\sigma_i = \begin{cases} 1, \text{якщо } M_i - \text{відмовивший модуль } (o-\text{ПМ}); \\ 0, \text{якщо } M_i - \text{працездатний модуль } (p-\text{ПМ}). \end{cases}$$

Вихідний стан системи задається як $s(t = 0) = s_0 = 00\ldots0$. Позначимо:

- Λ_n – множина всіх станів системи;
- $D = \{M_1,\ldots M_n\}$ – множина модулів з яких складається система;
- $\Omega_v = \{U_1, \ldots, U_L\}$ – множина задач, що підлягають розв'язанню в мережі при досягненні стану S_v;
- підмножина завдань які модуль M_i здатен опрацьовувати у відповідному стані системи S_v:

$$\Omega_i^v = \{U_1^i, \ldots, U_\mu^i, \ldots, U_m^i\}$$

Для початкового стану s_0 задано множину задач $\Omega_0 = \{U_j\}$, а також визначено початковий розподіл цих задач між модулями системи, тобто підмножини Ω_i^v, які відповідають окремим модулям.

Позначимо:

множину працездатних процесорних модулів для стану S_v як $Dr_v = \{M_i\}_v$; множину тих ПМ, що відмовили – $Df_v = \{M_i\}_v$. Через Af_v позначимо

усі задачі закріплені за модулями, які відмовили, в стані системи S_ν; $Ar_\nu = \Omega_0/Af_\nu$ – множина всіх задач закріплених за процесорним модулем для стану S_ν. Кожна з задач $U_\mu^i \in \Omega_i^\nu$ має визначений ступінь важливості, що відображає цілі функціонування системи управління і задається ваговим коефіцієнтом $\{\beta_{\mu i}\}$. Відповідно, результат розподілу задач у мережі оцінюється через корисний ефект E_ν (функціональна потужність мережі), який розраховується як сумарна вага задач, що виконуються у поточному стані системи S_ν.

Математична модель задачі перерозподілу описується так:

$$E_\nu = \sum_{i\=1}^{n_\nu} \sum_{\mu=1}^{m_i} \beta_{\mu i} X_{\mu i} \to \max, \tag{1.7}$$

за наступних обмежень:

$$\sum_{i\=1}^{n_\nu} \sum_{\mu=1}^{m_i} \Delta T_{\mu i} X_{\mu i} \le \Delta T_\nu^{доп}; \tag{1.8}$$

$$\sum_{i\=1}^{n_\nu} \sum_{\mu=1}^{m_i} V_{\mu i} X_{\mu i} \le V_\nu^{доп}; \tag{1.9}$$

$$\sum_{i\=1}^{n_\nu} \sum_{\mu=1}^{m_i} t_{\mu i} X_{\mu i} \le T_\nu^{доп}; \tag{1.10}$$

$$\sum_{\mu=1}^{m_i} X_{\mu i} \le 1, \tag{1.11}$$

де $X_{\mu i} = \begin{cases} 1, \text{якщо } \mu - \text{задача вирішується в } i - \text{му ПМ}; \\ 0, \text{ в іншому випадку}. \end{cases}$ (1.12)

$\Delta T_\nu^{доп}$, $V_\nu^{доп}$, $T_\nu^{доп}$ – гранично допустимі значення для середнього часу обслуговування задачі і-м процесорним модулем, об'єму пам'яті, необхідному для розв'язку завдання, часу на перерозподіл даних по всій мережі.

Обмеження 1.12 говорить про те, що завдання може бути призначеним для вирішення тільки на один з вузлів мережі, тоді завдання перерозподілу що описується (1.7-1.12) може бути вирішене як задача цілочисельного лінійного програмування з двійковими змінними. За умови деградування мережі в реальному часі.

1.5 Сучасні методи вирішення задачі ЦЛП з БЗ на прикладі задачі 0-1 рюкзак

Як уже зазначалося, проблема 0-1 рюкзак активно досліджується з моменту свого формулювання і дотепер.

У березні 2023 року була опублікована стаття [15] якій був представлений варіант нового метаевристичного алгоритму оптимізації, заснованого на рішенні з галузі фізики - оптимізатор спектру світла (LSO).

Автори пропонують використовувати свій, бінарний варіант, для розв'язання задач про ранець як в одно-, так і в багатовимірній варіаціях. Крім того, вони пропонують кілька покращених варіантів, заснованих на експлуатації бінарного модифікованого LSO (MBLSO) та інтеграції поведінки інтелекту рою, з модифікованим LSO (BHLSO).

Експериментальні результати показали перевагу BHLSO як для KP01, так і для MKP у порівнянні з кількома відомими на той момент алгоритмами з точки зору часу ЦП, швидкості збіжності та точності.

У тому ж році, у жовтні було представлено роботу [16] Nature-inspired algorithms for 0-1 knapsack problem: A survey, у якій авторами було проведено огляд і порівняльний аналіз деяких існуючих на той час метаевристичних алгоритмів розв'язання задачі 0-1 рюкзак, натхненних природними процесами. Таким чином увага авторів була зосереджена навколо таких алгоритмів як:

- генетичні алгоритми (GA);
- мурашині алгоритми (ACO);

- алгоритм рою частинок (PSO);
- імітація відпалу (SA);
- пошук із заборонами (Tabu Search);
- бджолина колонія (Bee Colony).

Автори також пропонують класифікацію задач 0-1 рюкзака на шість типів, залежно від методу кодування рішень.

У розділі експериментальних результатів показано, що різні методи мають свої переваги і недоліки. Головний висновок дослідження полягає в тому, що універсального алгоритму, який однаково добре вирішує всі варіанти задачі про ранець (No Free Lunch Theorem), не існує.

Запропонований у дослідженні [17] алгоритм адаптивної бінарної штучної бджолиної колонії (AİYAK) призначений для вирішення багатовимірної задачі про рюкзак (MKP). Автори загострюють увагу на тому, що ефективність та результативність алгоритмів оптимізації залежать від їх здатності активно шукати та швидко переміщатися у просторі рішень. Для цього AİYAK використовує кілька бінарних операторів сусідства. На відміну від традиційних метаевристичних алгоритмів, обмежених одним або генетичними операторами, AİYAK використовує адаптивний механізм вибору операторів, балансуючи «дослідження» та «експлуатацію» рішення. Проведене параметричне дослідження дозволило підібрати параметри алгоритму, такі, що порівняльний аналіз із чотирма сучасними методами (BGWO, BFOA, HHS, QPSO) на трьох наборах еталонних завдань показав, що AİYAK демонструє більш високу точність і надійність.

Ще один натхненний природою алгоритм було запропоновано авторами [18]. Оптимізація метелика-монарха (MBO) – це метаевристичний алгоритм, натхненний поведінкою метеликів-монархів під час міграції. Початковий MBO підходить для вирішення завдань безперервної оптимізації, але в цій статті представлена нова варіація MBO з оператором оновлення глобальної позиції (GMBO), який може вирішувати задачу 0-1 KP. Оператор оновлення включений, щоб допомогти всім метеликам-монархам швидко

переміщатися до глобальної, кращої позиції. Для подальшої оптимізації рішень впроваджений спеціальний двоетапний оператор відновлення. GMBO було перевірено та порівняно з уже відомими алгоритмами ABC, CS, DE, GA та MBO на великомасштабних екземплярах 0-1 KP. Експериментальні результати показують, що GMBO перевершує ці алгоритми за точністю рішення, швидкістю збіжності та чисельною стійкістю.

У роботі [19] запропоновано гібридний алгоритм гармонійного пошуку з оцінкою розподілу (HHSEDA) для вирішення задачі 0-1 рюкзак.

Особливість HHSEDA – використання ймовірнісної оцінку розподілу для адаптивного пошуку, покращений процес імпровізації для посилення пошуку, а також новий метод ініціалізації, що забезпечує допустимі початкові рішення. Як результат – перевага над традиційними методами, схильними до потрапляння в локальні мінімуми. Крім того, запропоновано покращений оператор виправлення, який коригує неприпустимі рішення, та підвищує точність та швидкість збіжності алгоритму.

Проведені експерименти показали, що HHSEDA перевершує інші методи (ABHS, BABC-DE, BFPA), особливо на великомасштабних задачах. Переваги включають генерацію коректних початкових рішень, стійкість до потрапляння до локальних мінімумів та ефективну стратегію виправлення рішень. Перспективи застосування охоплюють багатовимірні та дисконтовані рюкзаки, а також завдання маршрутизації.

Автори [20] пропонують нову ієрархічну класифікацію алгоритмів обчислювального інтелекту, поділяючи їх у дві основні групи: моделювання людського розуму і натхненні природою. Акцент зроблено на, поки що слабо вивчені, геонаукові методи, які на думку авторів мають потенціал для вирішення динамічних завдань, пов'язаних із умовами навколишнього середовища, що змінюються. Стаття не має прямого зв'язку з задачею про ранець, проте, наведені, здебільшого, можуть використовуватися для її рішення. Таким чином, стаття може дати уявлення про різноманіття алгоритмів оптимізації.

2 ВИКОРИСТАННЯ РАНГОВОГО ПІДХОДУ ПРИ ВИРІШЕННІ ЗАДАЧІ ЦІЛОЧИСЕЛЬНОГО ЛІНІЙНОГО ПРОГРАМУВАННЯ З БУЛЕВИМИ ЗМІННИМИ

2.1 Графічна інтерпретація задачі ЦЛП з БП

Поставимо у відповідність задачі (1.4) – (1.6) граф G (рисунок 2.1), який є зображенням усіх рішень, число яких дорівнює 2^n у вигляді двійкового дерева [21; 22].

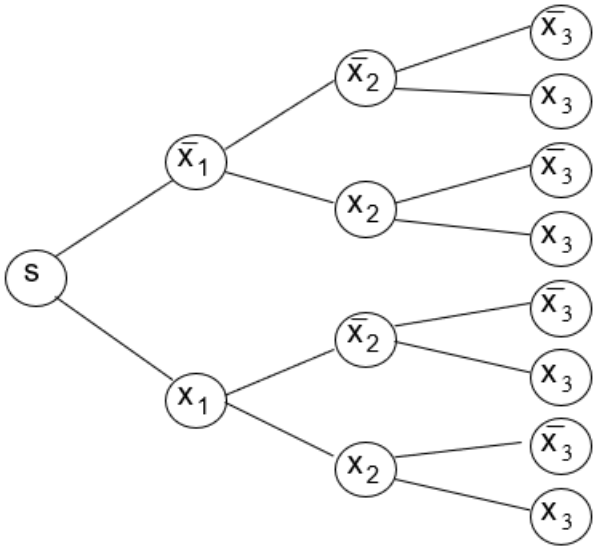


Рисунок 2.1 – Граф G

Множину яка містить усі можливі рішення, тобто усі двійкові вектори розмірності n, що представляють шляхи графа G, позначимо через X. Тоді $X = \{x_j\}$, де всі компоненти набувають значення 0 або 1, тобто $x_j \in \{0,1\}$.

Також існує деяка підмножина множини X яка є множиною припустимих рішень. Позначимо її V. Вектори, які входять до підмножини V обов'язково задовольняють обмеженням (1.5)-(1.6).

Нарешті, має існувати множина оптимальних рішень $H \subset V$, для якої значення функціоналу (1.4) для будь-якого вектора $x \in H$ дорівнює

екстремуму цього функціоналу.

Вся множина можливих рішень X може бути представлена групами векторів розділеними на категорії за їх вмістом:

- вектори, в яких тільки один компонент $x_j = 1$, при $j = \overline{(1,n)}$ і всі інші компоненти рівні 0;
- вектори, в яких два компоненти $x_j = 1$ в усіх можливих комбінаціях, а решта компонентів, дорівнюють 0;
- вектори, в яких три компоненти $x_j = 1$ в усіх можливих комбінаціях, а решта компонентів, дорівнюють 0;
- і так далі до вектора з n компонентів $x_j = 1$.

Позначивши підмножини векторів означених груп як m_r, при $r = \overline{(1,n)}$, множина X може бути представлена як об'єднання підмножин m^r:

$$X = \bigcup_{j=1}^{n} m^r. \tag{2.1}$$

В роботі [21] показано, що відповідно до графу G може бути побудований граф G' (рисунок 2.2). Множина усіх шляхів графу G' тотожна підмножинам векторів, які описувалися раніше співвідношенням (2.1). При цьому можна ввести поняття рангу шляху за яким: ранг шляху відповідає числу ребер, які утворюють цей шлях на графі G'.

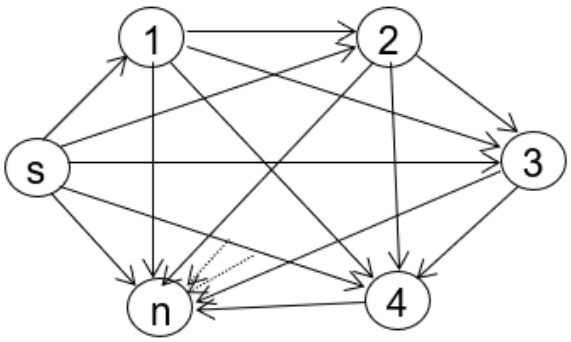


Рисунок 2.2 – Граф G'

Для побудови такого графу зобразимо початкову вершину s, що позначає стан в якому вектор містить лише 0. Від вершини s до вершин 1, 2, ..., n зображуємо спрямовані ребра. Кожну з вершин і з'єднаємо з вершинами i+1, i+2, ..., n. З вершини n не виходить жодного ребра, але ця, остання, вершина є кінцевою точкою ребер, від решти вершин.

Для того, щоб краще розуміти сутність ранговогу підходу, можна представити граф G' у вигляді дерева шляхів DΔ. Для його побудови уявимо площину розділену по горизонталі на яруси від нульового до n-го. Вершина s, що є початковою вершиною і фактично позначає нульовий вектор, розміщується згори в нульовому ярусі. Усі вершини графу G' до яких ведуть ребра, з початкової вершини s розташуємо на першому ярусі згори вниз у порядку зростання порядкових номерів і з'єднуємо їх з s. Утворена при цьому множина шляхів є множиною шляхів першого рангу. Аналогічно на другому ярусі розміщуємо ті вершини які на графі G' мають вхідні ребра з вершин розташованих на першому ярусі. Природно, що оскільки вершина 1 має вхідне ребро тільки з вершини s, якої немає в першому ярусі, то на другому ярусі її не буде. Таким чином в кожному наступному ярусі буде на одну вершину менше. Решта ярусів будуються аналогічно до тих пір поки не буде досягнуто ярусу з єдиною вершиною n. На рисунку 2.3 наведено приклад, коли n = 4, а на рисунку 2.4 його просторове уявлення.

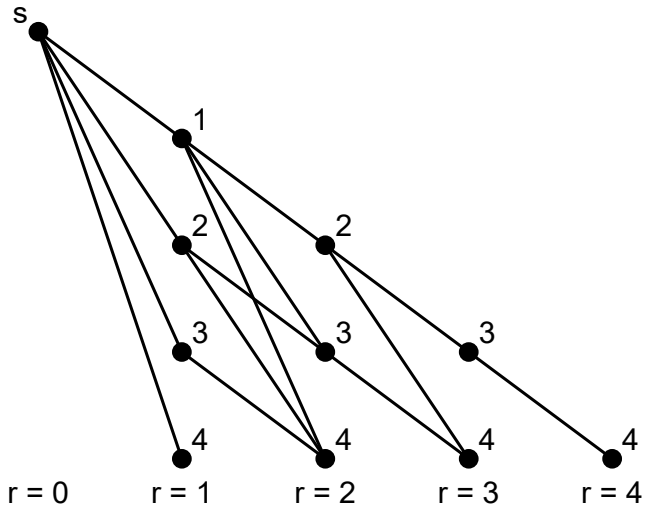


Рисунок 2.3 – Граф DΔ

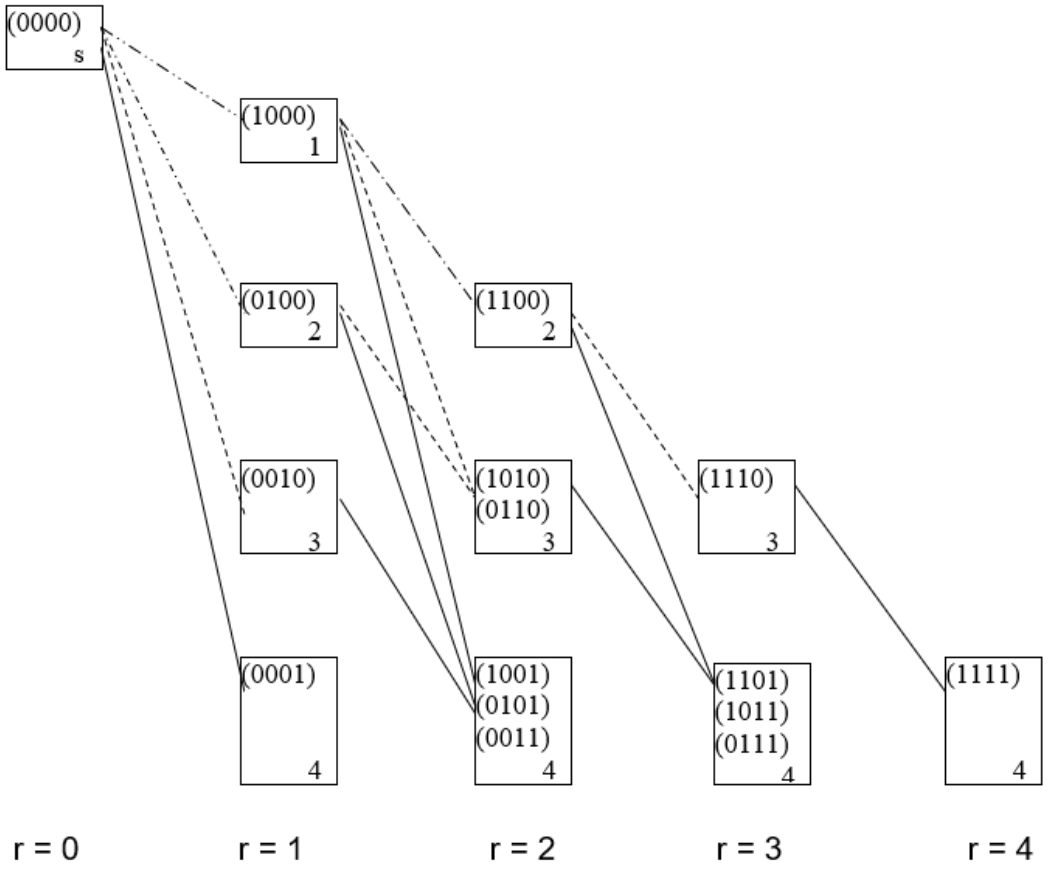


Рисунок 2.4 – Просторове уявлення графа DΔ

Геометрично будь-яка вершина k означеного графа певного рангу r є множиною векторів $\vec{x}$ $(x_1,x_2,...,x_k,...,x_n)$, у яких елемент x_k має значення одиниці, а між першою та k-ю (включно) позиціях перебувають r одиниць.

Тоді, можна казати, що кожному ребру, яке входить до вершини k даного графа можна поставити у відповідність канонічний вектор з одиницею в k-й координаті, одиничного куба B^n, $\vec{e}_k$ (0, 0, …,0, 1, 0, … 0).

У такому випадку кожному шляху рангу r між початковою точкою (s) та j у графі DΔ відповідає певний вектор $\vec{x}$, що дорівнює сумі базисних векторів, які відповідають ребрам, з яких утворено цей шлях.

Наприклад, шляху $\mu_{s123}^{r=3}$, відповідає вектор $\overrightarrow{x_{s123}}$, що утворився як сума початкового вектора, всі компоненти якого дорівнюють нулю $\vec{0}$ {0000} і базисних векторів e_1= {1000}, e_2 = {0100}, e_3 = {0010} тобто:

$$\overrightarrow{x_{s123}} = 0\{0000\} + \overrightarrow{e_1}\{1000\} + \overrightarrow{e_2}\{0100\} + \overrightarrow{e_3}\{0010\}$$

Припустимо, що кожному ребру графа DΔ, а отже і відповідному одиничному вектору, яке входить у деяку вершину j, при тому, що $j = \overline{(1,n)}$, відповідають дві ваги:

- вага c_j, яка дорівнює коефіцієнту при x_j у функціоналі (1.4);
- вага a_{1j}, яка дорівнює коефіцієнту при x_j в обмеженні (1.5).

У такому випадку, будь який шлях μ_{sj}^r певного рангу r, що починається у вершині s і закінчується вершиною j, може бути також охарактеризований двома довжинами:

- ваги функціоналу $d_c\left(\mu_{sj}^r\right)$;
- ваги обмежень $d_a\left(\mu_{sj}^r\right)_1$.

Означені довжини являють собою суму ваг функціоналу або обмежень відповідно, тих ребер з яких цей шлях утворено.

Множину шляхів $m_s^r(j)$ графу DΔ які ведуть від s до j, рангів $r = \overline{(1,n)}$, можна подати у вигляді:

$$m_s^r(j) = m_{sj}^{r=1} \cup m_{sj}^{r=2} \cup ... \cup m_{sj}^{r=n}, \qquad j = \overline{(1,n)}, \tag{2.2}$$

де m_{sj}^r – це множина шляхів μ_{sj}^r графу DΔ від початкової точки (s) до вершини j, розташованої на r-му ярусі означеного графа.

Оскільки множині $m_{sj}^{r=k}$ у графі DΔ у відповідність ставиться множина векторів $\{\vec{x}_1, \vec{x}_2, ..., \vec{x}_v\}$, у складі яких – k одиниць. Отже, $\left|m_{sj}^r\right| = C_n^{r=k}$, тобто кожному з векторів $(x_1, x_2, ..., x_n)$ у відповідність ставиться один із шляхів множини $m_{sj}^{r=k}$. З множини (2.2) слідує, що:

$$\left|m_s^r(j)\right| = C_n^{r=1} + C_n^{r=2} + ... + C_n^{r=n} = 2^n - 1.$$

З цього слідує що, граф DΔ відповідає впорядкованому варіанту гіперкуба B^n, а вузлам – вищезазначені шляхи $\mu_{sj}^r \in m_{sj}^r$.

Значення функціоналу кожного із шляхів відповідає величині функціонала відповідної вершини гіперкуба B^n, у той час як довжина цього шляху за вагою обмеження визначає чи належить дана вершина гіперплощині що описується рівнянням обмеження (1.5).

Якщо виконується умова $d_a\left(\mu_{sj}^r\right)_1 \le b$, то вершина представлена означеним вектором належить гіперплощині (1.5). Саме ж по собі виконання даної умови означає, що шлях μ_{sj}^r належить множині допустимих рішень і надалі казатимемо, що при виконанні умови $d_a\left(\mu_{sj}^r\right)_1 \le b$, шлях μ_{sj}^r задовольняє властивості ν.

У зворотному випадку, тобто за умови що $d_a\left(\mu_{sj}^r\right)_1 > b$, вузол гіперкуба, вважається таким, що не входить до гіперплощини (1.5), а шлях μ_{sj}^r – таким, що не виконує умову властивості ν.

Логічно, що оптимальним рішенням задачі (1.4) – (1.6) є вектор, що відповідає найдовшому за вагою функціоналу шляху графа DΔ, з поміж тих, що задовольняють властивості ν.

Якщо представити n-мірний гіперкуб B^n, як деревоподібний граф DΔ то можна розділити множину усіх шляхів, з вузла s на певну кількість локальних областей, яка відповідає кількості вершин графа і яку позначимо Ω. Число вершин означеного графа – сума чисел натурального ряду:

$$\Omega = n + (n-1) + ... + 1 = \frac{n \cdot (n+1)}{2} \approx \frac{n^2}{2}.$$

Відмітимо що локальні області отримані в графі Дельта впорядковані по рангах і у порядку зростання порядкового номера останнього елементу відповідних векторів. При цьому шлях будь-якого рангу може бути отриманий приєднанням нового ребра (j,p) до одного певного шляху чий ранг нижчий на одиницю:

$$m_{sp}^{r=r+1} = \left\{\left(\forall\left(\mu_{sj}^{r} \in m_{sj}^{r}\right)\right) \cup (j,p)\right\}. \qquad (2.3)$$

Допускаємо, що задані певні правила відсіювання $\{L_w\}$ тих шляхів μ_{sj}^{r} множин m_{sj}^{r}, на основі яких не може бути побудовано оптимальний шлях, таким чином у множинах залишаються тільки ті шляхи які одночасно задовольняють цим правилам та властивості ν (1.5). Тоді, оптимізаційний процес у напрямку до вершини p графу DΔ передбачає формування множин шляхів наступного, відносно поточного, рангу $m_{sp}^{r=r+1}$, таких, що задовольняють заданим правилам відсікань. формування зазначених шляхів відбувається за принципом, що описується рівнянням (2.3), за умови що це дозволить отримати у множині $m_{sp}^{r=r+1}$ такі шляхи, які відповідають правилам $\{L_w\}$ на основі такого співвідношення:

$$\forall\left(\mu_{sj}^{r} \in m_{sj}^{r}\right)\left[\mu_{sp}^{r=r+1} = L_w\left\{\mu_{sj}^{r} \cup (j,p)\right\}\right], \quad p = (\overline{r+1,n}); \;\; j = (\overline{r,n}), \qquad (2.4)$$

де $\mu_{sj}^{r} \cup (j,p)$ – шлях у графі DΔ з початкової вершини у p, транзитом через вузол j. Описаний шлях обов’язково відповідає правилам встановленим {Lw}. Тут і далі, для спрощення, вважатимемо, що певний шлях автоматично відповідає властивості ν, за умови, що він відповідає правилам встановленим в, {Lw}.

Для вирішення задачі представленої виразами (2.1) – (2.2) вибудуємо деяку процедуру A_0. Її задачею використовуючи співвідношення (2.4) та задані правила відсівання $\{L_w\}$, здійснити побудову множини локальних екстремумів функціоналу Ω з подальшим виділенням з них глобального. Простіше – вирішувати завдання (1.4) – (1.6). Також зазначимо, що оскільки шляхи, фактично, будуються послідовно, тобто номер кожної додаваної вершини більший, за номер тієї, яка додавалася раніше, то доцільно перед початком роботи відсортувати вершини за значенням функціоналу в порядку зменшення.

2.2 Узагальнена процедура A_0

Узагальнена процедура A0 складається з чотирьох кроків:

Крок 1. З початкового вузла s формуються множини шляхів $m_{sj}^{r=1}$, $j=\overline{(1,n)}$, які задовольняють властивості ν. В означених множинах виділяються шляхи $\mu_{sj}^{*\,r=1}$, які є локальними максимумами областей Ω_j, за функціональною довжиною.

Крок 2. Будуються множини $m_{sp}^{r=r+1}$ $p=\overline{(r+1,n)}$ шляхів рангу на одиницю більшиого за поточний, беручи за основу шляхи множини m_{sj}^{r} згідно зі співвідношенням (2.4). При цьому всі ті шляхи, що не виконують умови властивості ν відкидаються. Серед утворюваних шляхів здійснюється відсівання згідно з обраними правилами відсікань {Lw} і виділяються шляхи $\mu_{sp}^{*\,r=r+1}$, що являють собою локальні екстремуми в областях Ω_p.

Крок 3. Виконуємо перевірку, чи усі множини наступного рангу $m_{sp}^{r=r+1}$ є порожніми і при підтвердженні переходимо до кроку 4, інакше перевіряємо виконання рівності: r = (n - 1). Якщо рівність виконується йдемо до

четвертого кроку, якщо ні – до кроку 2 збільшуючи ранг на одиницю.

Крок 4. З поміж областей локальних екстремумальних значень Ω_j , де $j = \left(\overline{1, n^2/2}\right)$, виділяємо один, або декілька глобальних екстремумів. Після чого процедура A_0 завершує свою роботу.

Побудована таким чином узагальнена процедура A_0 дає можливість при кожному виконанні кроку 2 визначати місцеві екстремуми в кожному рангу графу DΔ, а при виконанні кроку 4 знайти з поміж них один, або декілька глобальних екстремумів, що сформовані на основі принципу 2.11 з використанням правил $\{L_w\}$.

Виходячи з математичної моделі n-вимірного куба з одиничню довжиною ребра B^n, представленого у вигляді деревоподібного графа DΔ, та з урахуванням вищеозначеного принципу оптимізації, що базується на ранговому підході, можна сформулювати три завдання:

- виведення стратегій $\{L_w\}$ для відсікання безперспективних, тобто таких які не можуть бути основою для оптимального, шляхів множинах m^r_{sj}.

- побудова алгоритмів, для знаходження наближених та точних рішень одномірної задачі ЦЛП з БЗ. Алгоритми будуватимуться на основі виведених правил відсікань безперспективних шляхів та використанні узагальненої процедури A_0.

- створення обчислювачів паралельної дії в якості спеціалізованих засобів для прискорення вирішення задач ДО, які зводяться до задачі ЦЛП з БЗ.

2.3 Метод відсіювання безперспективних варіантів

Фундамент методу відсіювання неперспективних варіантів розв’язку (шляхів) графу DΔ при вирішенні задачі ЦЛП з БЗ становить множина стратегій $\{L_w\}$ згадувана раніше. Застосування стратегій відсікання в процедурі A_0 дозволить вивести ряд алгоритмів рішення задачі ЦЛП з БЗ.

Суть стратегії відсікань L_1 полягає у тому, що шляхи наступного рангу формуються на основі тих шляхів поточного рангу які задовольняють властивості ν і мають максимальну довжину за вагою функціоналу в кожній з множин m_{sj}^{r} поточного рангу.

У відповідності до стратегії L_1 співвідношення (2.4) набуває наступного вигляду:

$$\mu_{sp}^{r=r+1} = \max_{c_j} \left\{ \mu_{sj}^{r} \cup (j,p) \right\}; \quad p = (\overline{r+1,n}); \quad j = (\overline{r,n}), \tag{2.5}$$

Розберемо практичний приклад.

Приклад 2.1. Максимізувати функціонал:

$$f(\vec{x}) = 20x_1 + 18x_2 + 18x_3 + 15x_4 + 10x_5 + 4x_6 + 1x_7,$$

при обмеженні:

$$70x_1 + 2x_2 + 5x_3 + 2x_4 + 3x_5 + 1x_6 + 4x_7 \le 12,,$$

використовуючи стратегію L_1.

На рисунку 2.5 проілюстрована робота стратегії L_1.

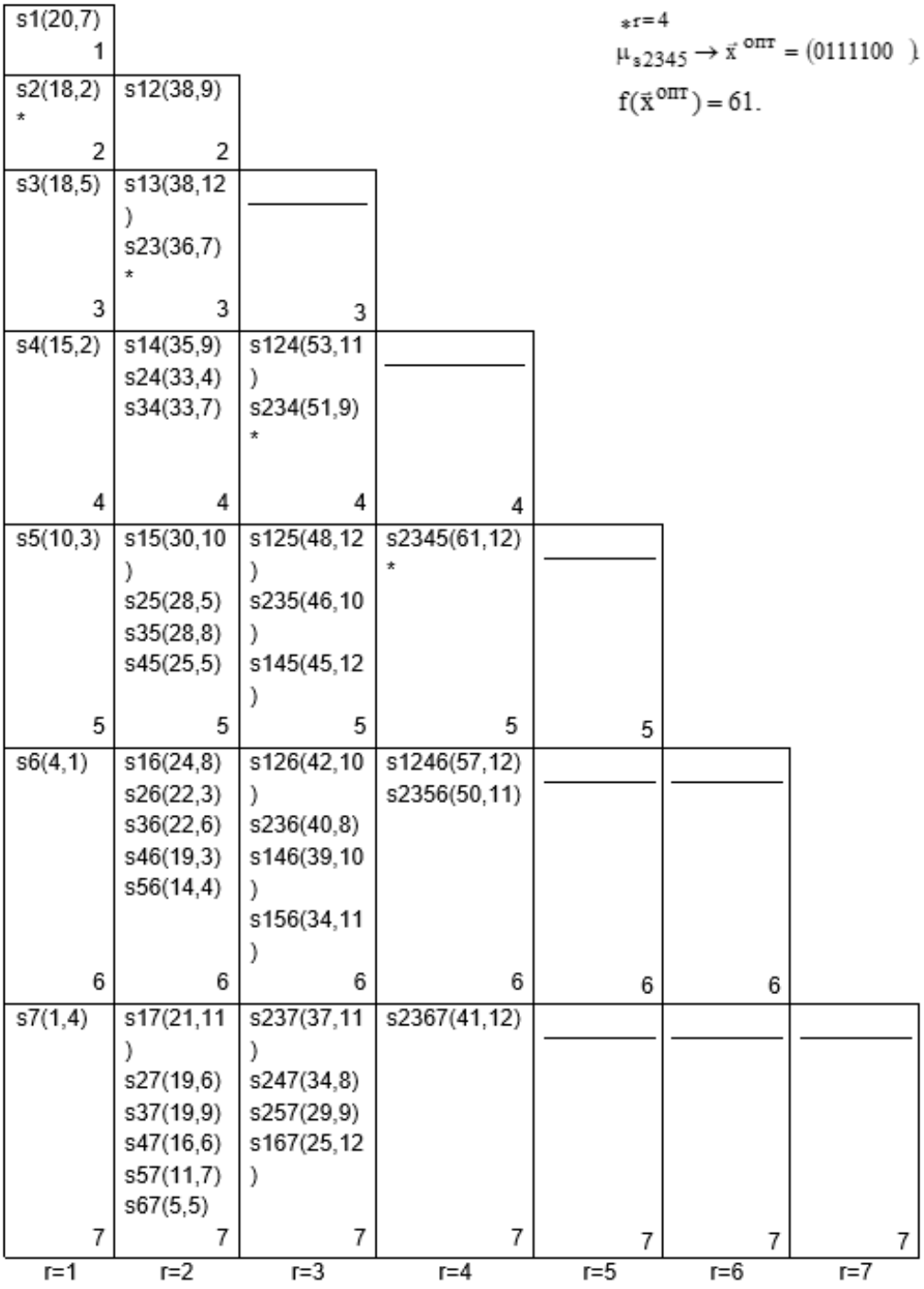


Рисунок. 2.5 – Ілюстрація роботи стратегії L_1 (оптимальний шлях позначено «*»)

Першим кроком є побудова відповідного задачі графа DΔ, при цьому не зображуючи вершину s, оскільки для практичного рішення вона не є суттєвою. Граф побудуємо у вигляді що близький до геометричної інтерпретації графа DΔ, тобто як скошену таблицю у комірках якої

розмістимо номери вершин які вони символізують і групи множини шляхів, в неї входять. Кожний шлях μ_{sj}^{r}, $j=(\overline{r,n})$ позначається ідентифікатором, який говорить про те через які вузли попередніх рангів він побудований. Так, якщо ідентифікатор шляху – s127, це означає, що шлях $\mu_{s127}^{r=3}$, проходить по вузлах $s\to1\to2\to7$. У дужках, що стоять після ідентифікатора приведено по два значення. Перше значення відповідає функціональній довжині шляху. Друге значення в дужках позначає довжину шляху за вагою обмеження.

Пошук рішення є ітеративним.

Ітерація 1. Формуємо шляхи рангу один, що задовольняють ν. На першому ранзі їх сім: $\mu_{s1}^{r=1}$-$\mu_{s7}^{r=1}$ з яких обираємо один з найбільшою функціональною довжиною: $\mu_{s1}^{r=1}$, який позначаємо як $\mu_{s1}^{*\,r=1}$.

Ітерація 2. Формуємо шляхи де r = 2 у відповідності до принципу (2.5). Наприклад, шлях $\mu_{s13}^{r=2}$ будується з кінцевого вузла шляху $\mu_{s1}^{r=1}$, додаванням ребра (1,3). Функціональна довжина $d_c\left(\mu_{s13}^{r=2}\right)=38$, за обмеження $d_a\left(\mu_{s13}^{r=2}\right)_1=12$, тобто утворений $\mu_{s13}^{r=2}$ відповідає властивості ν.

Серед утворених шляхів виділяємо ті, що мають максмальниу функціональну довжину $d_c\left(\mu_{sj}^{r=2}\right)$. Це шляхи $\mu_{s12}^{*\,r=2}$ і $\mu_{s13}^{*\,r=2}$.

Ітерація 3. Формуємо шляхи третього рангу (r = 3) на основі шляхів рангу r = 2 використовуючи (2.5).

Додаючи ребро ребро (2,3) до шляху $\mu_{s12}^{r=2}$ можна утворити $\mu_{s123}^{r=3}$, однак, його обмеження $d_a\left(\mu_{s123}^{r=3}\right)_1=14$, але ця величина виходить за рамки обмеження, тому такий шлях відкидається.

Якщо при формуванні нового шляху він не відповідає властивості ν, то береться наступний, менший за функціоналом шлях з тієї ж підмножини. На r = 2 рисунка 2.5 можна спостерігати такий розвиток подій під час формування

шляху з підмножини $m_{s3}^{r=2}$ в $m_{s4}^{r=3}$. Серед шляхів третього рангу обираємо найдовший за вункціоналом $d_c\left(\mu_{sj}^{r=3}\right)$ шлях. На третьому ранзі це шлях: $\mu_{s124}^{*\,r=3}$.

Ітерація 4. Утворюємо шляхи четвертого рангу (r = 4), аналогічно шляхам попереднього рангу. Серед отриманих шляхів вибираємо шлях з найбільшим значенням $d_c\left(\mu_{sj}^{r=4}\right)$, тобто шлях $\mu_{s2345}^{*\,r=4}$.

Ітерація 5. Утворюємо шляхи п'ятого рангу (r = 5). Однак як бачимо з рисунка 2.5 на цьому ранзі немає шляхів які б задовольняли властивості ν, тому серед шляхів $\mu_{sj}^{*\,r}$, $r=(\overline{1,4})$, $j=(\overline{r,n})$ – вибираємо найбільший: $\mu_{s2345}^{*\,r=4}$. Тоді вектор $\vec{x}^{\,опт}=(0111100)$, який відповідає обраному шляху $\mu_{s2345}^{*\,r=4}$ є оптимальним розв'язком поставленої задачі.

Зазначимо, що використання стратегії L_1 дає оптимальне рішення задачі не в кожному випадку, причиною цього є той факт, що у кожній підмножині алгоритм обирає шлях найдовший за функціоналом, який, однак, може набирати більшу вагу обмеження. Тобто, може статися що шляхи наступники обраного шляху не можуть бути побудовані через невідповідність обмеженню, а шляхи, менші за довжиною обмежень, але що могли потенційно перевищити за довжиною функціонала при продовженні, вже відсіяні за стратегією L_1.

Стратегія відсівання безперспективних шляхів L_2 схожа за логікою роботи на вже розглянуту L_1. Відмінність полягає у тому що замість найдовших шляхів за вагою функціоналу, в якості основи для векторів наступного рангу обираються шляхи з найменшим обмеженням. Завдяки цьому стратегія L_2 дозволяє отримувати шляхи найбільшого можливого рангу.

Поза як це є єдиною відмінністю в роботі стратегії від вже описаної L_1,

то приклад наразі не наводитимемо.

Без шкоди для двох уже описаних стратегій, можна запропонувати додаткову стратегію завчасного відсівання шляхів m_{sj}^{r} певного рангу r. Пропонована стратегія L_3, спирається на той факт, що шляхи більш високих рангів будуються на основі шляхів нижчих рангів згідно з (2.3), а максимальний потенційний приріст функціональної довжини γ_j для будь-якого шляху у такому випадку визначається шляхом підсумовування коефіцієнтів c_j усіх наступних вузлів і без врахування обмеження:

$$\gamma_j = c_{j+1} + c_{j+2} + ... + c_n, \qquad \gamma_n = 0; \qquad j = \overline{(1, n-1)} \tag{2.6}$$

На основі даного факту висунемо наступне твердження 1.

Твердження 1. Якщо сумарно функціональна довжина $d_c\left(\mu_{sj}^{r}\right)$ шляху у вузол w та максимальний потенційний приріст функціоналу γ_w цього вузла менші за уже отриманий функціонал іншого шляху того ж рангу то шляхи побудовані на основі даного не можуть бути оптимальним рішенням завдання.

Твердження 1 є обґрунтованим, оскільки процедура A0 обирає глобальний екстремум серед локальних а вже сформований принаймні один шлях який за функціональною довжиною переважає той, що будується і всіх його послідовників, тобто жоден з них не зможе бути обраний глобальним екстремумом.

Тоді, рішення про відкидання шляху може бути прийняте при виконанні умови:

$$d_c\left(\mu_{sp}^{r}\right) + \gamma_p < \max_{\{c_j\}}\left\{ d_c\left(\overset{*}{\mu}{}_{sp}^{r} \right) \right\}, \tag{2.7}$$

де $d_c\left(\mu_{sp}^r\right)$ – довжина шляху μ_{sp}^r до вузла p рангу r з коефіцієнтом c_j.

Розберемо практичний приклад.

Приклад 2.2. Знайти вектор $\vec{x}$ при якому свого максимального значення набуває функція:

$$f(\vec{x}) = 20x_1 + 18x_2 + 18x_3 + 15x_4 + 10x_5 + 4x_6 + 1x_7,$$

Враховуючи обмеження:

$$70x_1 + 2x_2 + 5x_3 + 2x_4 + 3x_5 + 1x_6 + 4x_7 \le 12,$$

використовуючи стратегії L_1 і L_3.

Розв'язання даного прикладу в графічній формі наведено на рисунку 2.6. Підкреслений шлях на рисунку – це шлях, який береться за основу для побудови множин наступного рангу. якщо шлях не підкреслено це означає що він відкидається. Крайній лівий стовпець на рисунку відведено для значень максимального потенційного приросту функціональної довжини γ_j, $j = \overline{(1,7)}$ для вершини в тому ж рядку.

Пусті множини m_{sj}^r на рисунку 2.6 з'явилися внаслідок відсікання за умовою (2.7).

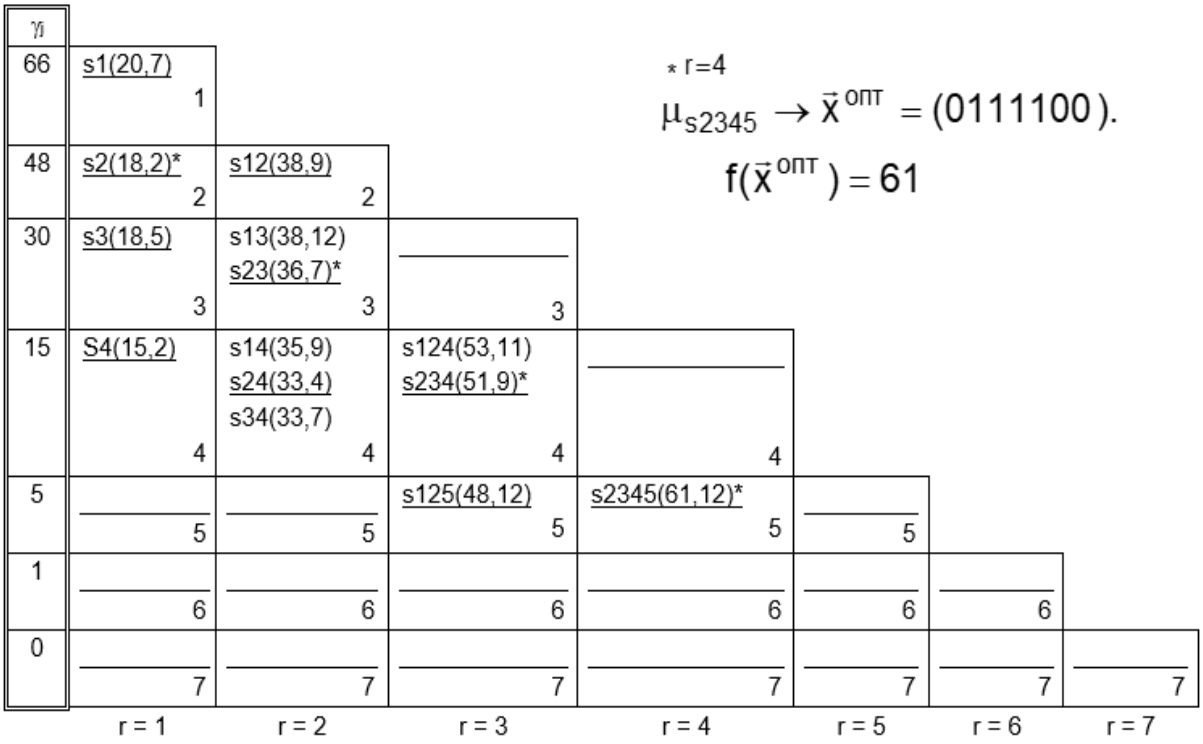


Рисунок 2.6 – Ілюстрація роботи стратегії L_1 і L_3

Проілюструємо роботу цієї статегії побудувавши шлях $\mu_{s15}^{r=2}$. На момент його побудови максимальна довжина функціоналу серед шляхів даного рангу $\max\left\{d_c\left(\mu_{s12}^{*\,r=2}\right)\right\}=38$, а довжина функціоналу даного шляху $d_c\left(\mu_{s15}^{r=2}\right)=30$. максимальний приріст функціональної довжини для п'ятого вузла дорівнює п'яти, тобто $d_c\left(\mu_{s15}^{r=2}\right)+\gamma_5=35$. Таким чином спираючись на твердження 1 побудований шлях а також його наслідувачі не можуть бути оптимальним рішенням, а отже відкидаються згідно зі стратегією L_3. У випадку якщо б довжина за функціоналом цього шляху перевищувала функціональну довжину локального екстремуму цього рангу то подальші дії були б аналогічні тим що наводилося у прикладі 2.1.

З графічного подання прикладу 2.2 видно що використання стратегії L_3 дозволяє значно знизити число векторів серед яких обиратиметься основи для векторів наступних рангів.

Стратегія L_3, хоча і дозволяє суттєво скоротити кількість векторів, але вимагає перевіряти за нею кожен з векторів. Цю процедуру можна скоротити модифікувавши дану стратегію.

Модифікація L3 основана на тому ж принципі відсівання шляхів що і стандартна. Однак вона дозволяє відсівати більшу кількість шляхів одномоментно завдяки змінам що наведені нижче.

Якщо розглядати вектори першого рангу з прикладної точки зору то в них враховується лише з один елемент. Пам'ятаючи, що, перед початком роботи A_0, вхідна множина цих елементів відсортовується в порядку зменшення функціоналу можна вважати, що коли знайдено одноелементний вектор для якого виконується (2.7), то ані цей вектор, ані жодний з наступних одиничних векторів чи їх послідовників не можуть бути оптимальним рішенням.

Така поведінка зумовлена тим, що за умови відсортованості елементів $d_c\left(\mu^1_{sj}\right)$ завжди більша за $d_c\left(\mu^1_{sj+1}\right)$, так само як і γ_j за будь яких умов більша за γ_{j+1}.

Розглядаючи вектори другого рангу слід аналізувати чи містить вектор елемент з порядковим номером один. Якщо це так то такий вектор в будь-якому разі функцією має більшу функціональну довжину ніж будь-які вектори що можуть бути побудовані в тому ж ранзі після нього.

Наприклад вектор s14 в будь-якому разі має більшу функціональну довжину ніж вектори s24, s34, s15 Тому якщо такий вектор відсіюється то це є підставою для переривання формування векторів другого рангу.

Якщо вектор другого рангу не містить, в своєму складі, першого елемента то відсіювання такого вектора на підставі (2.7) не є достатньою умовою для переривання рангу, оскільки за ним можуть слідувати вектори, що мають у своєму складі більш вартісні за функціоналом елементи.

Однак відкидання такого вектора є підставою для переривання формування векторів поточної підмножини, оскільки жоден з векторів другого рангу що мають в якості кінцевого елемента ту саму вершину але

менш вартісний елемент в якості початкового – не можуть перевершити навіть відкидуваний.

Починаючи з третього рангу у векторах також необхідно аналізувати наявність першого елементу вхідного набору, однак вплив відсівання вектора, який його містить зменшується.

Так, якщо відкидається вектор в складі якого є елемент з порядковим номером один, то можна завершити формування векторів тільки поточної підмножини.

Якщо ж відсівається вектор, який не містить елемент з порядковим номером один то це ніяк не впливає на решту векторів, тобто відкидається тільки цей вектор та його нащадки.

Як видно з даного опису така модифікація дозволяє відкидати значну частину безперспективних векторів До того як вони будуть сформовані і, відповідно, зможуть бути перевірені.

3 АЛГОРИТМИ МЕТОДУ ВІДСІКАНЬ НЕПЕРСПЕКТИВНИХ ВАРІАНТІВ

На основі висунутих раніше, стратегій відсікання $\{L_w\}$ та процедури А0, необхідно розробити алгоритми вирішення задачі ЦЛП з БЗ. Так як було розглянуто 3 стратегії, а саме L_1, L_2, L_3, і дві з них мають протилежні ідеї, то ми не можемо створити з них багатоетапний алгоритм, отже, будемо розглядати тільки такі алгоритми які знаходять вирішення за один прохід, тобто одноетапні. Розглядана задача оптимізації приймалася одномірною, отже і алгоритми будуть для задачі з одним обмеженням, а так як розглянуті стратегії (окрім L_3) в звичайному випадку не дозволяють отримати точне рішення то і алгоритми будуть наближеними.

Одноетапність, одновимірність та наближеність рішення, в даному випадку, обрані як ключові класифікаційні ознаки (кількість етапів, кількість обмежень, точність рішення) для розглянутих далі алгоритмів, і дозволить легко зіставити їх з іншими алгоритмами, що будуть розглядатися в майбутньому.

На разі опишемо тільки два наближені алгоритми для одновимірного рюкзака (1.4)-(1.6), а саме алгоритми A_1 та A_2 чия робота заснована на використанні розглянутих раніше стратегій відсіювання неоптимальних варіантів рішень L_1, L_2 та L_3 з основою у вигляді процедури A_0.

3.1 Алгоритм A_1 для вирішення задачі з ЦЛП з БЗ

Алгоритм A_1 робота якого вже фактично розглядалася під час огляду L_3 (рисунок 2.6). Алгоритму лежить процедура А0 на якій використовується стратегії L1 та L3. Алгоритм працює в п’ять кроків:

Крок 1. Будується множина шляхів $m_{sj}^{r=1}, j=\overline{(l,n)}$, для r=1, що починаються у вершині s і задовольняють властивості ν. У побудованій множині визначається шлях максимальної функціональної довжини. В

рамках стратегії L_3 Для кожної цільової вершини j за правилом (2.6) визначається вага γ_j.

Крок 2. З множини поточного рангу r, тобто m_{sj}^r, виключаються шляхи $\{\mu_{sp}^r\}$, $p=(\overline{r,n})$, які задовольняють нерівності (2.7). Простіше кажучи виключаються шляхи які не можуть бути основою для жодного шляху чия функціональна довжина перевищила б поточний локальний екстремум рангу.

Крок 3. На основі множини шляхів m_{sj}^r, за співвідношеням (2.5) формується множина шляхів наступного, більш високого рангу $m_{sp}^{r=r+1}$, $p=(\overline{1,n})$. В множині утримуються тільки ті шляхи, які задовольняють властивості ν. Виділяються шляхи, чия функціональна довжина є максимальною. Якщо доводиться обирати серед декількох таких шляхів то обирається той вага обмеження якого є найменшою, а якщо характеристика шляхів є однаковими то обирається будь-який з них.

Крок 4. Перевіряється чи є можливість формування шляхів наступного, (r+1)-го рангу. І якщо така можливість є, то ранг збільшується на одиницю і виконується перехід до кроку два, в іншому разі виконується перехід до кроку п'ять.

Крок 5. Серед екстремумів всіх рангів виділяємо один або декілька (якщо характеристика функціоналу однакова) глобальних. Алгоритм закінчує роботу.

3.2 Алгоритм A_2 для вирішення задачі з ЦЛП з БЗ

Алгоритм A_2 використовує в якості основи процедуру A_0 та стратегії L_2, L_3. Даний алгоритм орієнтується на формування шляхів найбільш високих з можливих рангів. Для цього, природно, в кожній підмножині обираються шляхи з найменшим значенням обмеження. Як і A_1, A_2 складається з п'яти кроків:

Крок 1. Будується множина шляхів $m_{sj}^{r=1}, j=\overline{(1,n)}$ рангу один, які починаються у вершині s і задовольняють властивості ν. У побудованій множині визначається шлях максимальної функціональної довжини. В рамках стратегії L_3 Для кожної цільової вершини j за правилом (2.6) визначається вага γ_j.

Крок 2. З поточного рангу r, відкидаються шляхи $\{\mu_{sp}^{r}\}$, $p=\overline{(r,n)}$, для яких виконується нерівність (2.7).

Крок 3. На основі множини шляхів m_{sj}^{r} у відповідності до (2.5) формується множина шляхів наступного, більш високого рангу $m_{sp}^{r=r+1}$, $p=\overline{(1,n)}$, при цьому шляхи, які не виконують умови властивості ν відкидаються. Для побудови шляхів наступного рангу у підмножинах виділяються шляхи що мають найменше значення обмеження. За умови, що таких шляхів декілька то вибирається той функціональна довжина якого є найбільшою.

Крок 4. Перевіряється чи є можливість формування шляхів наступного, (r+1)-го рангу. І якщо така можливість є, то ранг збільшується на одиницю і виконується перехід до кроку два, в іншому разі виконується перехід до кроку п'ять.

Крок 5. Серед екстремумів всіх рангів виділяємо один або декілька (якщо характеристика функціоналу однакова) глобальних. Алгоритм закінчує роботу.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ЕКСПЕРИМЕНТАЛЬНОГО ЗАСТОСУНКУ

4.1 Програмна реалізація узагальненої процедури A_0

Першим завданням програмної реалізації A_0 є створення деякої структури даних в якій зберігатимуться вектори рішень. Така структура була реалізована як динамічний масив, що містить посилання на списки. Розмір масиву відповідає кількості елементів, що надійшли на вхід задачі. Самі списки містять посилання на об'єкти класу «Node». Кожен об'єкт класу «Node» зберігає в собі вектор (тому далі для спрощення йтиметься саме про вектори, а не об’єкти), його параметри та набір конструкторів необхідних для формування вектору. Таким чином основними полями об'єкту є:

- functional і obmejenya типу int – для зберігання значень функціоналу та обмеження вектора відповідно;
- nabir – динамічний масив типу int для зберігання самого вектора у форматі набору десяткових цифр;
- функції конструктори – для створення нових об'єктів класу.

Сам програмний еквівалент A_0 являє собою потрійний ієрархічний цикл. Цикл першого і найвищого рівня виконує перебір рангів, тобто обирає у вищезазначеному масиві список який буде заповнюватися. Вкладений в нього цикл другого рівня перебирає вершини призначення, причому починаючи з номер, що дорівнює поточному рангу.

За такої структури усі вектори одного рангу фактично вміщується в одну спільному списку, через це відпадає можливість перебору векторів певної підмножини без створення окремої логіки, так як будь які фізичні кордони між векторами відсутні.

На рисунку 4.1 приведено спрощений алгоритм програмної реалізації узагальненої процедури A_0.

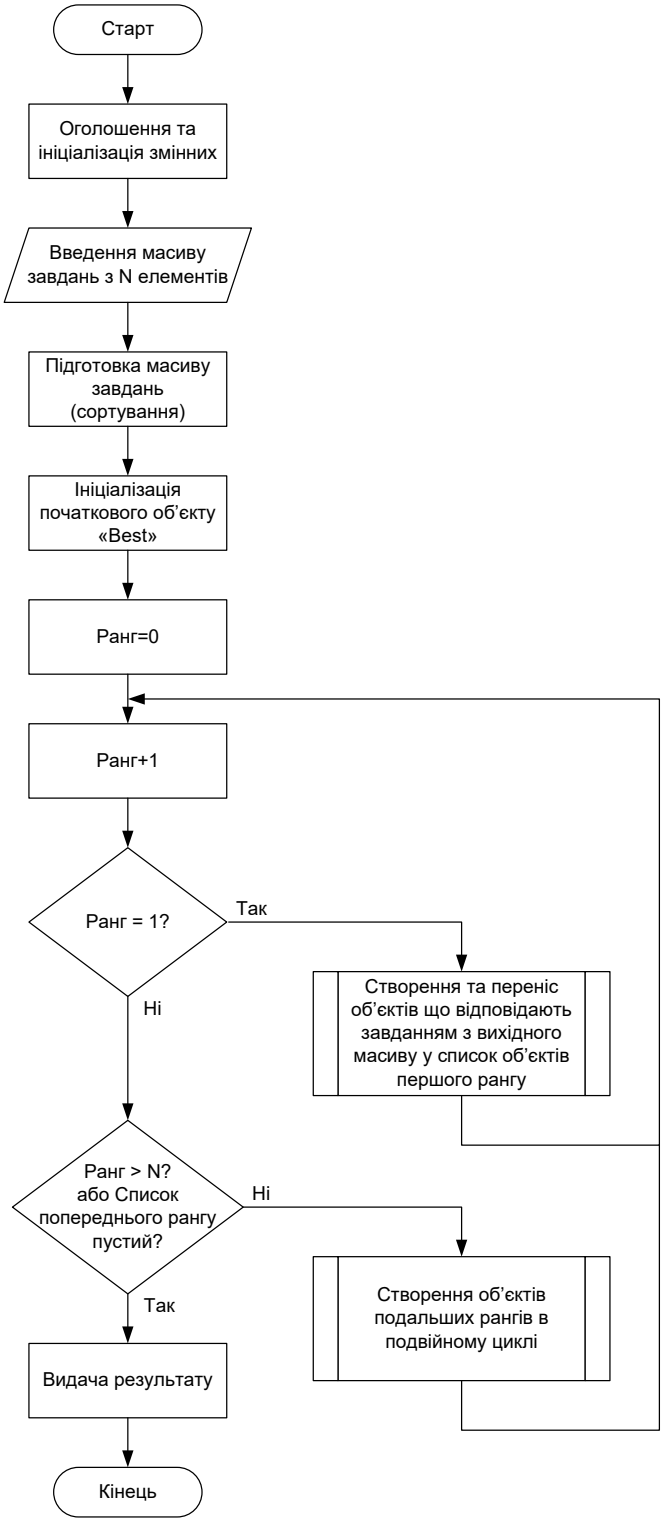


Рисунок 4.1 – Спрощений алгоритм програми A_0

На рисунку 4.1 чітко видно дві підпрограми. Доцільно спершу розглянути другу підпрограму «Створення об'єктів подальших рангів в подвійному циклі». На рисунку 4.2 зображено спрощений алгоритм цієї підпрограми.

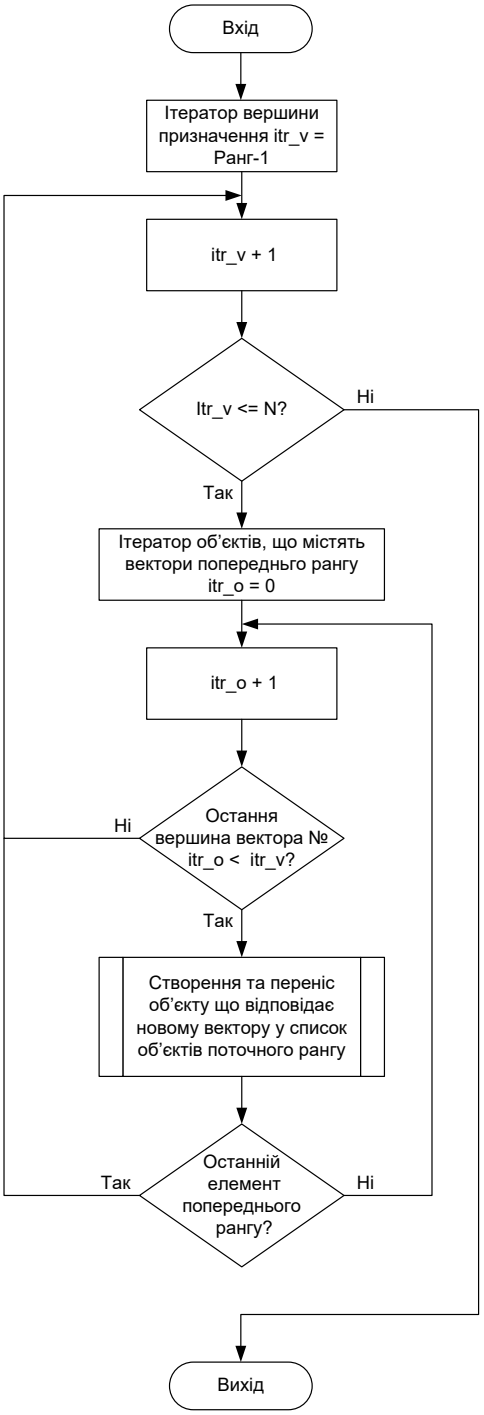
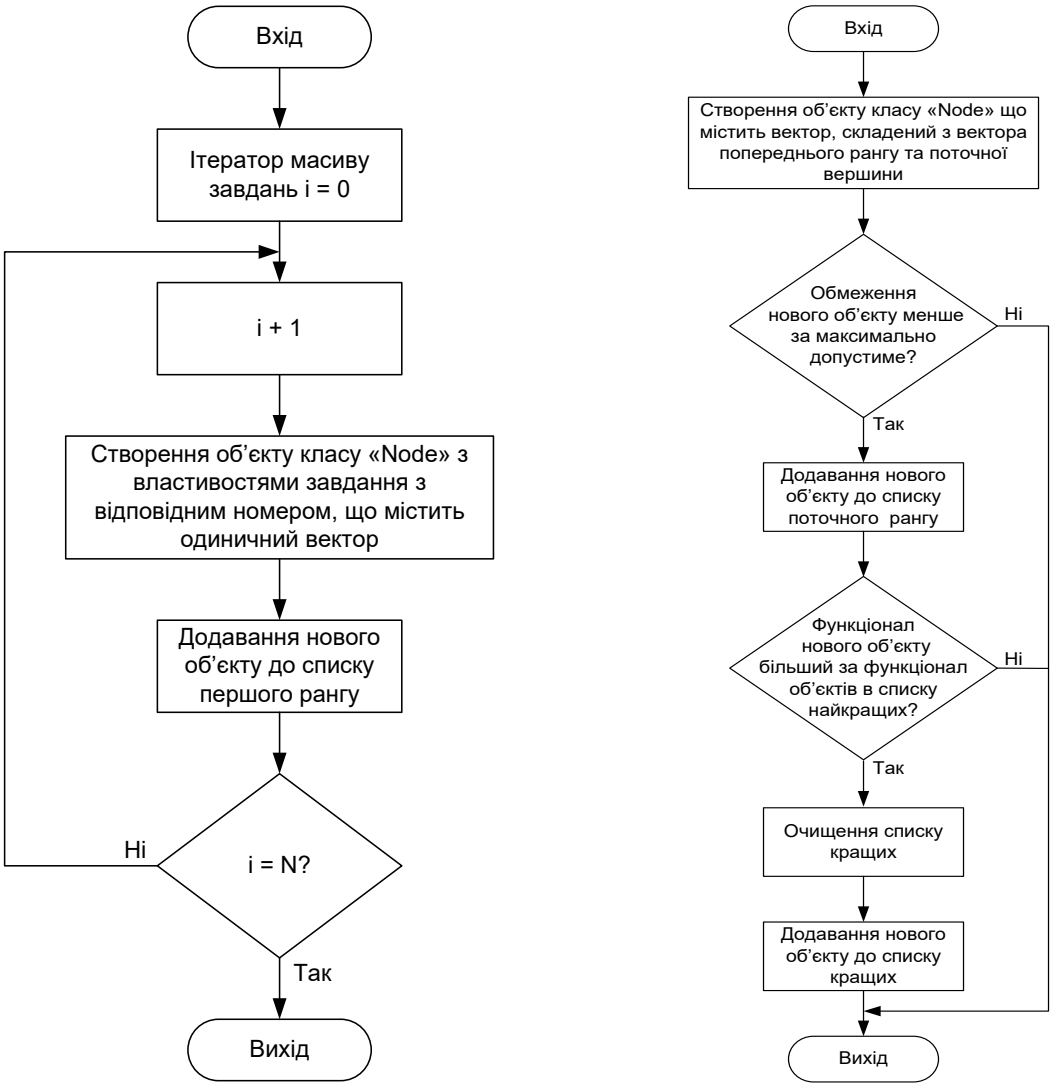


Рисунок 4.2 – Підпрограма узагальненої процедури A_0 – «Створення об'єктів подальших рангів в подвійному циклі»

Таким чином виділяється і ще одна підпрограма. Як і перша підпрограма в A_0 нова підпрогорама «Створення та переніс об'єкту що відповідає новому вектору у список об'єктів поточного рангу» відповідає за формування нових об'єктів, що містять можливі вектори рішень. В

класичному випадку вони є доволі простими і реалізуються як зображено на рисунку та 4.3а та 4.3б.



а) б)

Рисунок 4.3 – Підпрограми узагальненої процедури A_0: створення та перенесення: а) об'єктів що відповідають завданням з вихідного масиву у список об'єктів першого рангу; б) об'єкту що відповідає новому вектору у список об'єктів поточного рангу

Зауважимо, що стратегії відсікання програмно не є ані окремими функціями, ані окремими програмами, замість цього вони представлені як видозмінені блоки програмної реалізації самої A_0, або її підпрограм.

4.2 Програмна реалізація стратегії відсікання L_1

Стратегія відсікання L_1 заснована на виборі в якості основи для нового вектору лише вектору з найбільшим значенням функціоналу з кожної підмножини нижчого рангу. При цьому значення обмеження враховується Або за умови що є декілька векторів з однаковим функціоналом, і тоді слід вибрати той у якого довжина за обмеження є меншою, або вже після формування нового вектору для перевірки виконання умови властивості ню. Враховується що за умови коли новий вектор був відкинутий допускається вибір іншого вихідного вектору з меншим значенням функціоналу.

Зазначимо, що з моменту формування процедури A_0 вважалася справедливим твердження, що завдяки відсортованості елементів вихідного масиву вектори в кожній підмножині одразу відсортовані за функціональною довжиною в порядку її зменшення. В реальності дане твердження є частково справедливим у випадку використання чистої процедури A_0. І навіть тоді воно справедливе тільки для підмножин другого рангу, у підмножинах наступних рангів, починаючи з третього можливі стрибки в бік збільшення функціональної довжини, як правило, за рахунок використання в якості векторів-попередників, тих векторів, що належать до різних підмножин. наприклад в підмножині третього рангу: s125, s135, s235, s145 цілком можливий такий стрибок функціоналу для вектору s145, оскільки s14 може мати більшу функціональну довжину за s23.

Оскільки вищезазначене твердження виявилося хибним (або принаймні потребує окремого розбору і тестування для підтвердження або спростування), то виникає проблема визначення вектору за стратегією L_1. Розглянемо два підходи:

- вибір на основі сортування;
- вибір на основі простого порівняння (далі лінійний пошук).

Реалізація підходу на основі сортування використовує стандартну функцію мови c++ Stable sort, яка заснована на алгоритмі сортування

злиттям. Перевагою цього підходу є тим що при відкиданні сформованого вектора немає необхідності виконувати повторний пошук нової основи, недоліком – додаткові витрати на зберігання відсортованої колекції даних та виконання зайвих дій у випадку якщо перший же вектор буде підходящим.

Як зазначалося раніше стратегія відсікання реалізується як видозміни підпрограми представленої на рисунку 4.3б, за таким алгоритмом:

- вектори однієї підмножини копіюються до окремого вектору, де в подальшому буде виконуватися сортування.

- в отриманому векторі відбувається два послідовних сортування, перше сортує вміст у порядку зростання за значенням обмеження, друге - за функціональною довжиною у порядку її зменшення, зберігаючи вихідну послідовність за умови що ця довжина однакова.

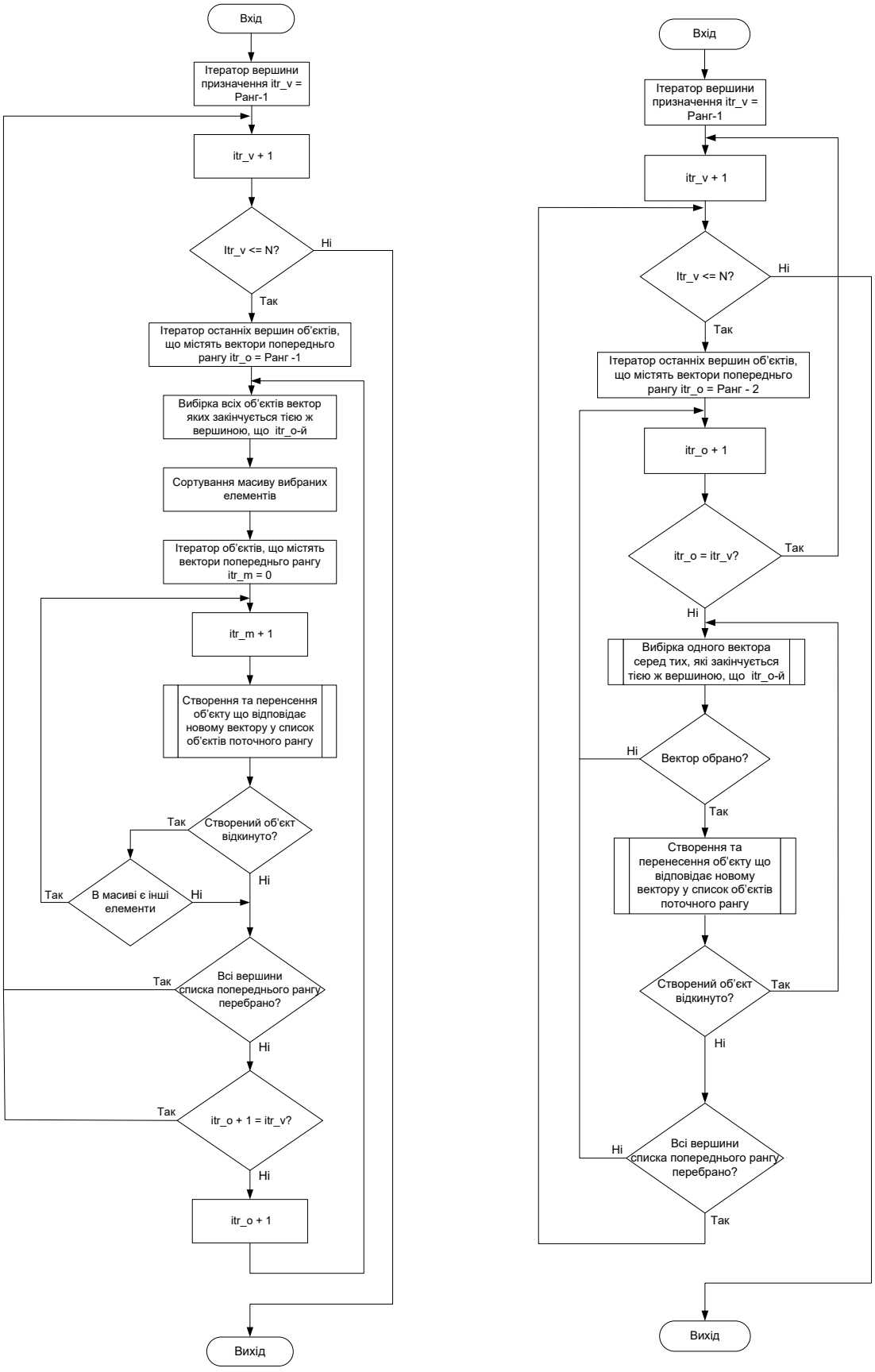
Підхід заснований на лінійному пошуку гарно показує себе, якщо підходящий вектор вдається віднайти протягом перших 2-3 спроб. Працює він циклічно:

- під час першого проходу перебираються усі вектори однієї підмножини. Після того як обраний один з векторів додається додаткова умова яка дозволяє вибрати інший вектор такої ж функціональної довжини з меншим обмеженням або вектор більшої функціональної довжини. Як тільки перебрано всі вектори цієї підмножини на основі обраного вектора будується новий;

- якщо новостворений вектор відкидається через перевищення обмеження то відбувається новий перебір векторів тієї ж підмножини, але додаються нові умови згідно з якими не може бути обраний вектор тієї ж або більшої (для третьої і подальших спроб) функціональної довжини.

Спрощений алгоритм модифікацій підпрограми формування нових векторів з рисунка 4.3б для реалізації L_1 приведений на рисунку 4.4.

-



а) б)

Рисунок 4.4 – Підпрограма «Створення об’єктів подальших рангів в подвійному циклі» з реалізацією L_1 через: а) сортування; б) лінійний перебір

Можна помітити, що на рисунку 4.4 з’явилася нова підпрограма: вибірка одного вектора. Сутність цієї підпрограми це код, що згідно до описаних вище правил вибирає з підмножини підходящий вектор. Ця підпрограма не розкриватиметься детально, оскільки її реалізація технічно складна, і при цьому не має ключового значення для розуміння загальної структури алгоритму.

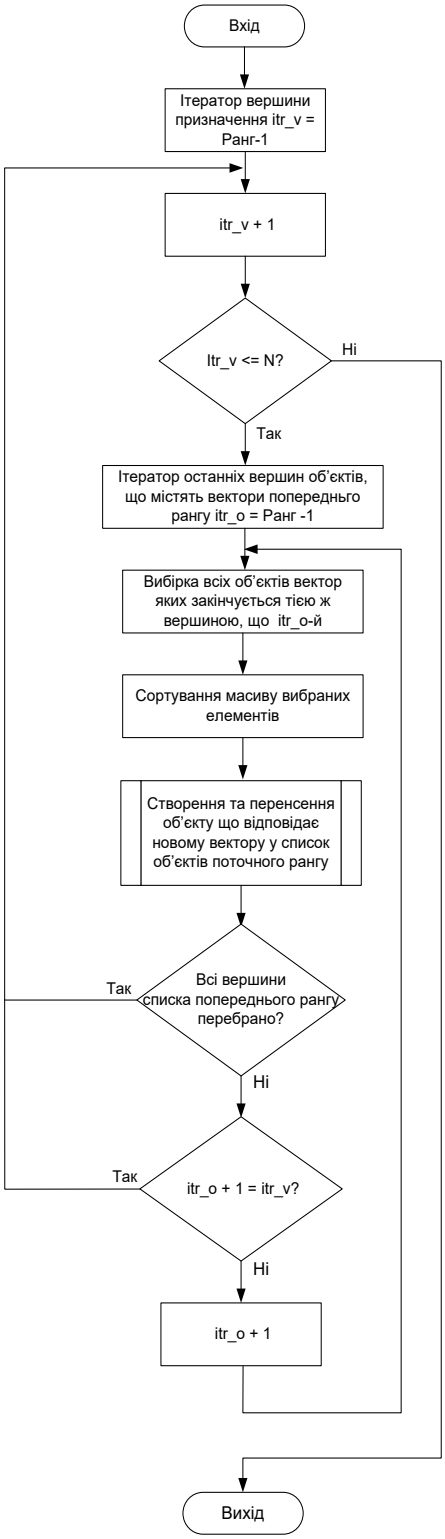
4.3 Програмна реалізація стратегії відсікання L_2

Як вже вказувалося в підрозділі 2.3, стратегія L_2 заснована на логіці зворотній до L_1. Програмно вона також реалізується аналогічно в обох варіаціях за винятком того, що:

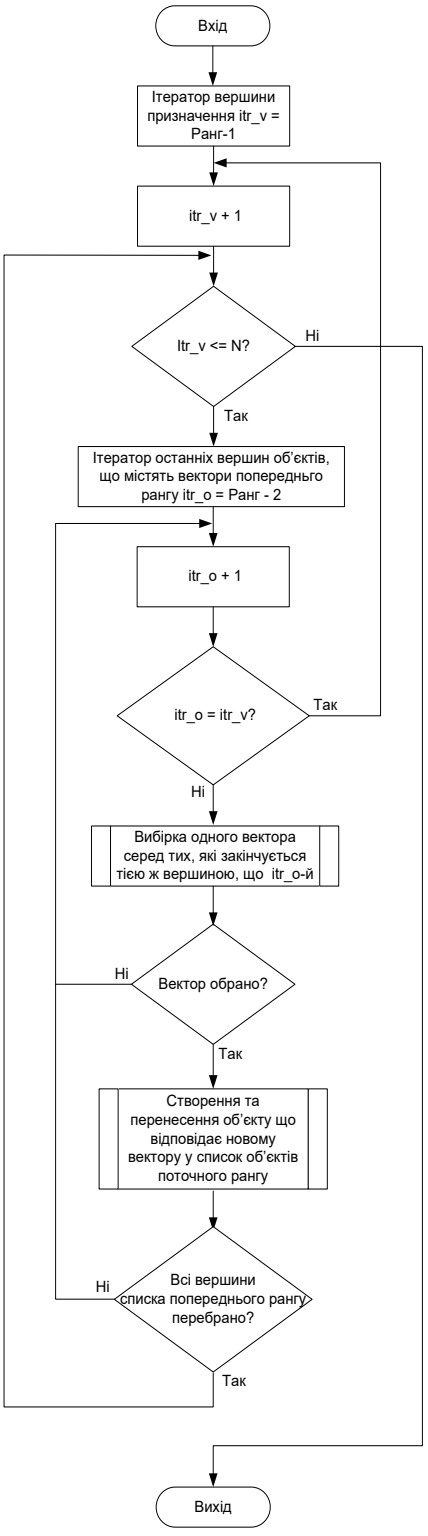
- у варіанті з сортуванням під час першого сортування вектори сортуються на основі функціональної довжини у порядку її зменшення. Під час другого сортування за обмеженнями але вже у порядку зростання.
- при використанні лінійного пошуку перебір векторів підмножини відбувається лише один раз. При цьому коли вибраний один будь-який вектор інший може бути переобраний замість нього тільки у тому випадку якщо його обмеження менше або таке саме але при більшому функціоналі.

Таким чином використання даної стратегії дає лише один шанс на знаходження підходящого вектора для приєднання нового елементу в кожній підмножині. це у свою чергу виявляє найбільший недолік реалізації через сортування, оскільки під час нього завжди виконується зайві операції.

На рисунку 4.5 приведено спрощений алгоритм реалізації стратегії відсікання L_2.



а)

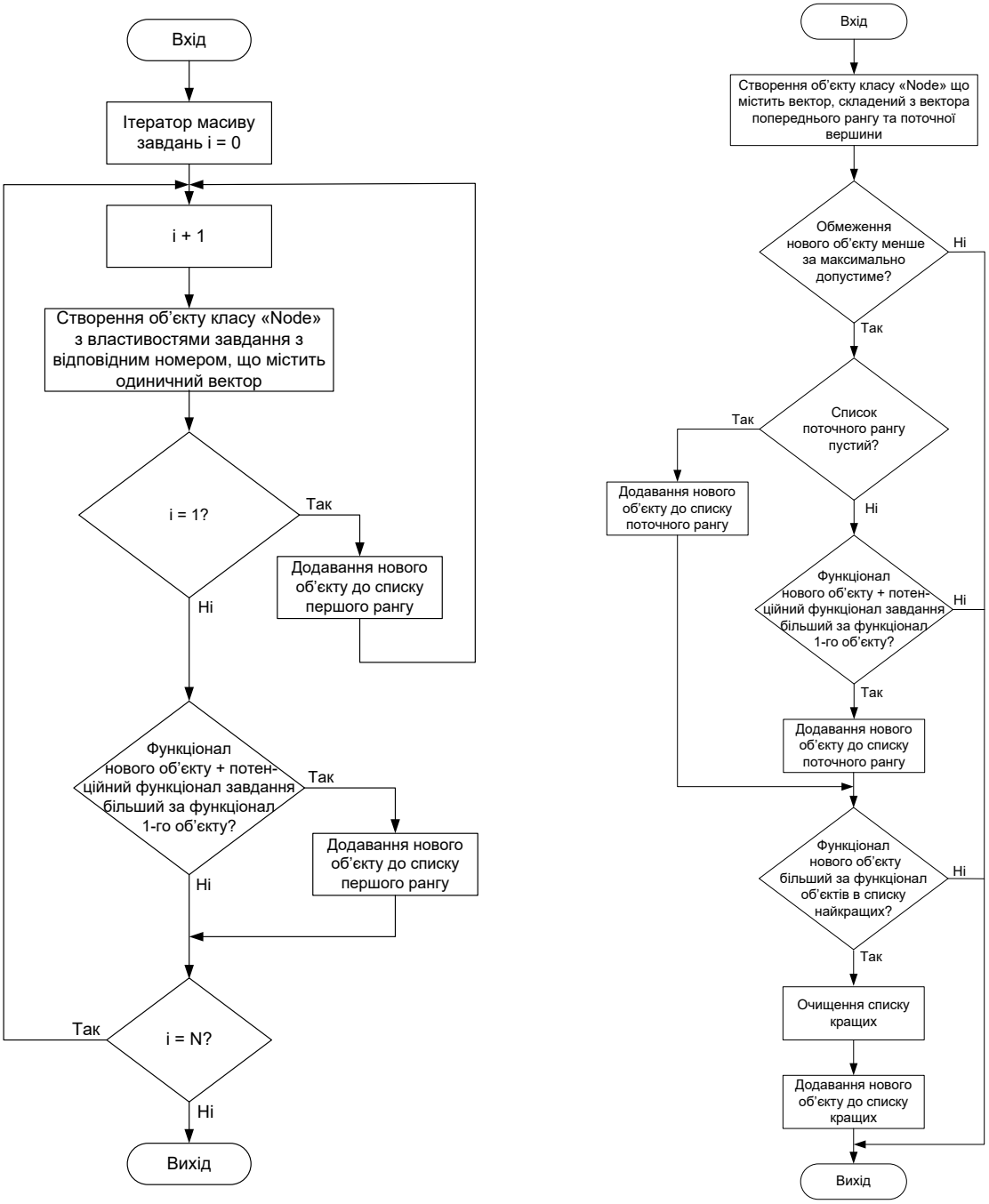


б)

Рисунок 4.5 – Підпрограма «Створення об’єктів подальших рангів в подвійному циклі» з реалізацією L_2 через: а) сортування; б) лінійний перебір

4.4 Програмна реалізація стратегії відсікання L_3

Спрощений алгоритм реалізації L_3 показано на рисунку 4.6.



а) б)

Рисунок 4.6 – Підпрограми A_0 з імплементованою стратегією L_3: Створення та перенесення: а) об'єктів у список першого рангу; б) об'єкту що відповідає новому вектору у список об'єктів поточного рангу

Стратегія L_3 незалежно від реалізації зводиться до розрахунку максимального потенційного приросту функціоналу для кожного елемента вихідного масиву та введення додаткової перевірки (2.7) при формуванні вектора.

Є принаймні два способи якими можна розраховувати згаданий максимальний приріст потенціалу, а саме починаючи з першого елементу, або починаючи з останнього елементу. Другий варіант є більш вигідним оскільки дозволяє розраховувати максимальний приріст для кожного наступного елемента однією операцією.

У своєму звичайному вигляді L_3 реалізується як модифікація підпрограм з рисунку 4.3а і 4.3б. Що і було показано рисунку 4.6а та 4.6б показано спрощений алгоритм роботи цих підпрограм, модифікованих з L_3.

Логіка описувана раніше у підрозділі 2.3 для модифікованої стратегії L_3 реалізується за допомогою додаткових перевірок складу вектора на етапі його створення та механізмів переривання циклів.

4.5 Програмна реалізація додатку для тестування алгоритму пошуку рішення в умовах практичного застосунку

Розгляд алгоритмів пошуку рішення завдання про рюкзак 0-1, до якого зводиться завдання балансування навантаження в мережі, у вакуумі не має сенсу. Тож у рамках дослідження також було змодельовано спрощену багатопроцесорну систему. Система задля простоти та економії ресурсів обрана невеликою: п’ять робочих пристроїв (процесорів) та один керуючий «центр». Спрощений алгоритм якої показано на рисунку 4.7.

В системі задається деяка кількість задач. Хоча в реальних системах звісно відбувається постійна зміна їх кількості, проте в моделі ця кількість є кінцевою задля економії пам’яті пристрою на якому відбувається моделювання.

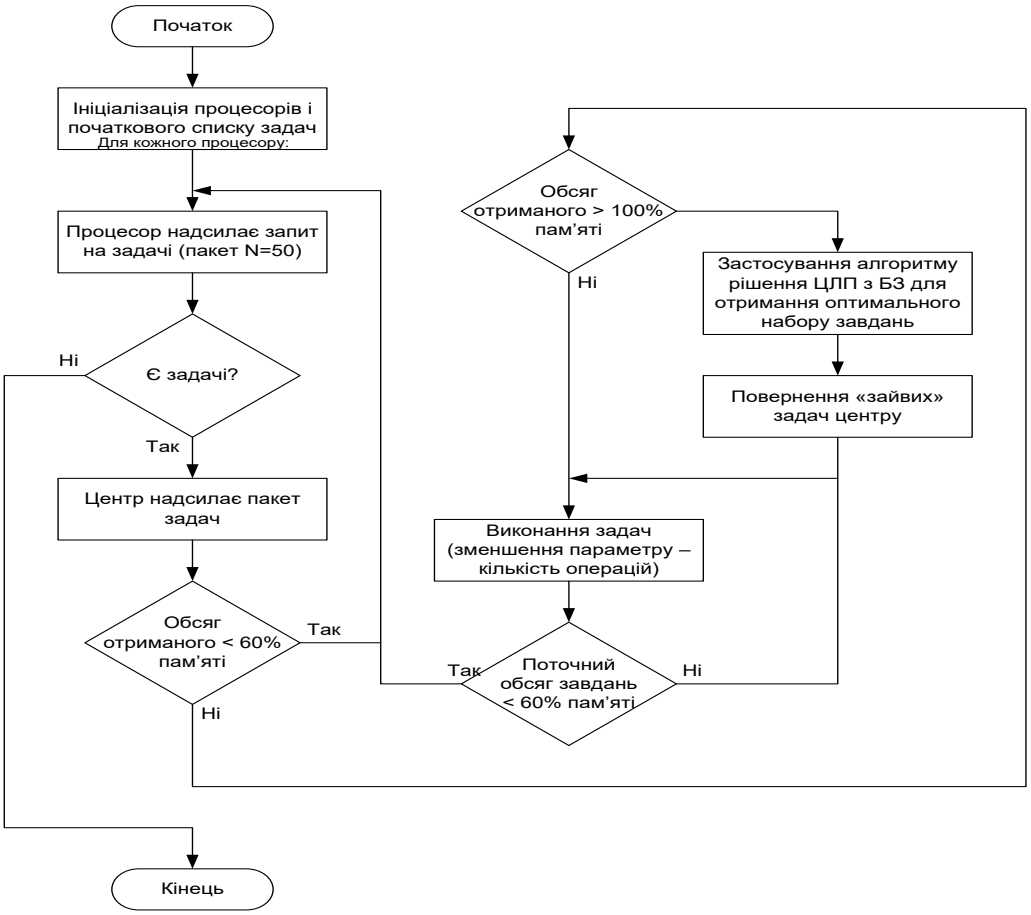


Рисунок 4.7 – Спрощений алгоритм роботи моделі багатопроцесорної системи для тестування ефективності запропонованих алгоритмів

Кожна задача описується через:

- необхідний об'єм пам'яті – цей параметр відповідає вазі обмеження а в реальності може вважатися пам'яттю що необхідно виділити процесу;
- вага функціоналу – цей параметр можна вважати своєрідним еквівалентом пріорітету, у тому сенсі що його виконання може вважатися більш цінним, однак за умов обмеженості ресурсів, може бути вигіднішим вирішити інші завдання;
- кількість операцій – цей параметр відповідає за кількість часу що буде необхідна процесору на вирішення задачі. Коли кількість операцій сягає нуля, завдання вважається виконаним і покидає чергу.

Кожен робочий пристрій/процесор має два параметри: кількість операцій за секунду – ops (від англ. operations per second). В даній моделі цей параметр визначає, на яке число буде зменшено кількість операцій задач що перебувають на виконанні в цьому пристрої за квант часу; об'єм пам'яті –

загальний об’єм пам’яті пристрою що може бути виділена під процеси.

Ціль кожного процесора полягає в максимально ефективному використанні своїх ресурсів, не перевищуючи заданого обмеження.

Алгоритм роботи даної моделі можна описати наступним чином.

Крок 1. Задаються параметри кожного пристрою/процесора.

Крок 2. Задаються параметри задач.

Крок 3. Кожному пристрою, який надсилає запит, надсилається деяка кількість задач. За замовчуванням 50.

Крок 4. На кожному пристрої незалежно виконується фільтрація. Якщо:

- задачі займають менше 60% пам’яті пристрою до центру надсилається запит на ще один пакет завдань, інакше процесор працює з тими задачами, що отримав. Якщо на центрі не залишилося задач продовження роботи допоки не виконано всі поточні задачі і перехід в режим активного очікування;

- загальна пам’ять необхідна для задач отриманих від центру перевищує максимальну для цього пристрою, на пристрої виконується відбір задач за одним з алгоритмів вирішення задачі ЦЛП з БЗ. Всі завдання що не увійшли до вектору рішення надсилаються назад на центральний пристрій;

- від центру, або в результаті роботи алгоритму пошуку, загальна пам’ять необхідна для задач отриманих від центру перевищує 60% обмеження, але не перевищує 100% пристрій починає «виконання», тобто починає зменшувати параметр «кількість» операцій кожної задачі раз у деякий проміжок часу.

У даній моделі припускається, що задачі, які вже потрапили на пристрій, продовжують виконання до повного завершення і не можуть бути повернуті назад до центрального керуючого пристрою. Таким чином, при падінні завантаження нижче 60% додаткові задачі отримані за запитом розміщуються тільки у межах вільного обсягу пам’яті. Це спрощення дозволяє уникнути необхідності враховувати часткове виконання задач при повторному їх розміщенні.

5 РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

Метою данного розділу є порівняльне тестування цих варіацій алгоритмів.

Через те, що більшість описаних стратегій L_w Мають по два варіанти реалізації а пропоновані алгоритми включають в себе по дві стратегії, то загалом було сформовано вісім варіантів тестової програми. Для зручності вони зведені у таблицю 5.1.

Таблиця 5.1 – Варіації реалізації експериментальної програми

A1 / A2			
L3(без модифікацій)		L3(модифікована)	
L1(з сортуванням)	L1(без сортування)	L1(з сортуванням)	L1(без сортування)

Тестування проводилися за наступних умов:

- в якості середовища тестування ноутбук Acer Aspire 3, cpu Intel Core i3 2,30 GHz, RAM 8 Gib з операційною системою Windows 10,;
- об'єм вхідного масиву 150 елементів;
- кількість тестів - 100 штук, для кожної варіації програми.

Далі на рисунках будуть подані результати тестування за такими метриками як похибка обчислення відносно точної відповіді, та кількість проаналізованих векторів. На кожному рисунку вміщено по чотири графіки відповідно для кожної реалізації алгоритму A_1 або A_2.

Результати експериментів проведених для визначення похибки обчислення в залежності від реалізації стратегії відсікання в алгоритмі A_1 наведено на рисунку 5.1. Видно що для алгоритму A_1 величина похибки залежить від реалізації тільки в одному випадку, який в цілому можна розглядати як статистичну аномалію. Можна також помітити що значення

похибка набуває значень від нуля до 7,6%.

Результати експериментів проведених для визначення похибки обчислення в залежності від реалізації стратегії відсікання в алгоритмі A_1 наведено на рисунку 5.1. Видно що для алгоритму A_1 величина похибки залежить від реалізації тільки в одному випадку, який в цілому можна розглядати як статистичну аномалію. Можна також помітити що значення похибка набуває значень від нуля до 7,6%.

Аналогічні дані експериментів для алгоритму A_2 приведено на рисунку 5.2. Величина похибки для всіх реалізацій цього алгоритму не набуває значень більше 6,3%. При цьому близько половини тестів показали, що використання стратегії L_2 з сортуванням показує дещо більш точні результати, але різниця не перевищує 1,5%.

Рисунок 5.3 демонструє результати експериментів, в яких аналізувалися кількість переглянутих векторів для усіх варіацій алгоритму A_1. Видно що порівняно з методом грубої сили будь-яка реалізація A_1 значно виграємо за кількістю аналізованих векторів, дозволяючи працювати максимум з 173000. Відмітимо що модифікована L_3, дозволяє додатково зменшити цю кількість приблизно на 40000 в кожному випадку.

На рисунку 5.4 демонструється результати таких же експериментів для варіації алгоритму A_2. Одразу можна помітити що кількість проаналізованих векторів в даному алгоритмі є меншою у середньому на 10000 векторів порівняно з A_1. У цьому випадку використання модифікації також дозволяє у середньому знизити кількість опрацьованих векторів на 40000.

Вплив конкретної варіації L_1, або L_2 на метрику кількості проглянутих векторів в масштабах задачі є мало помітним і залишається у межах однієї тисячі векторів.

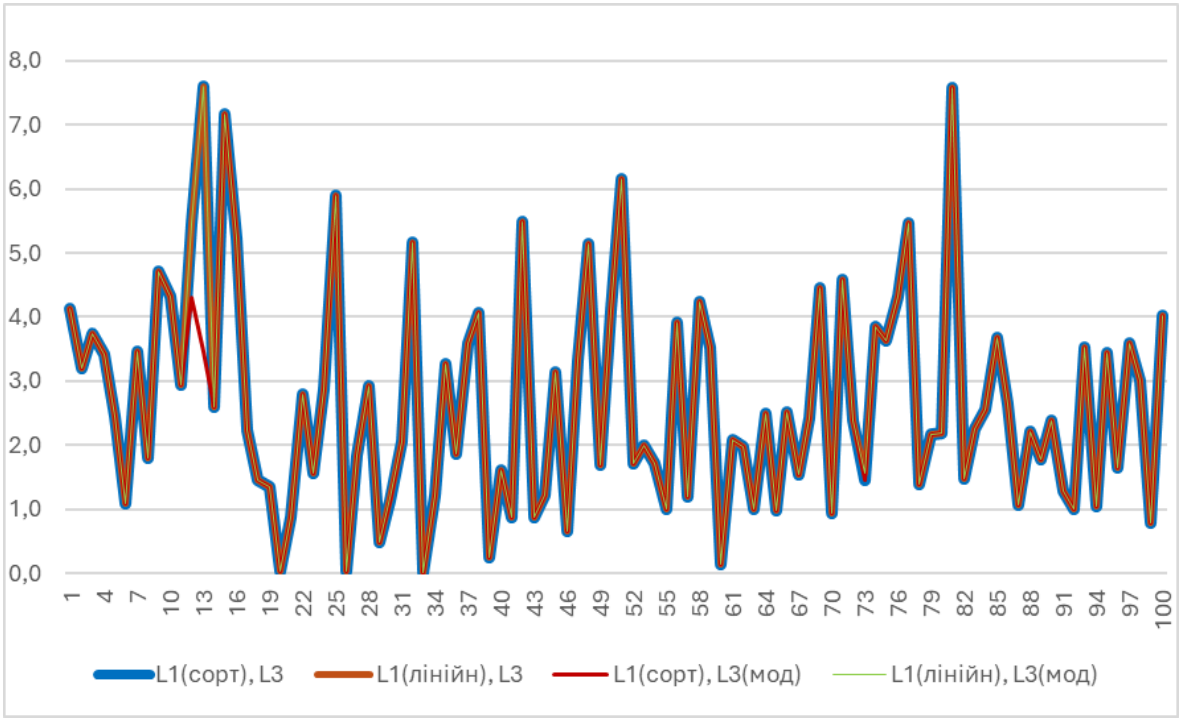


Рисунок 5.1 – Експериментальна оцінка похибки результату алгоритму A_1 в залежності від варіантів реалізації стратегій L_w

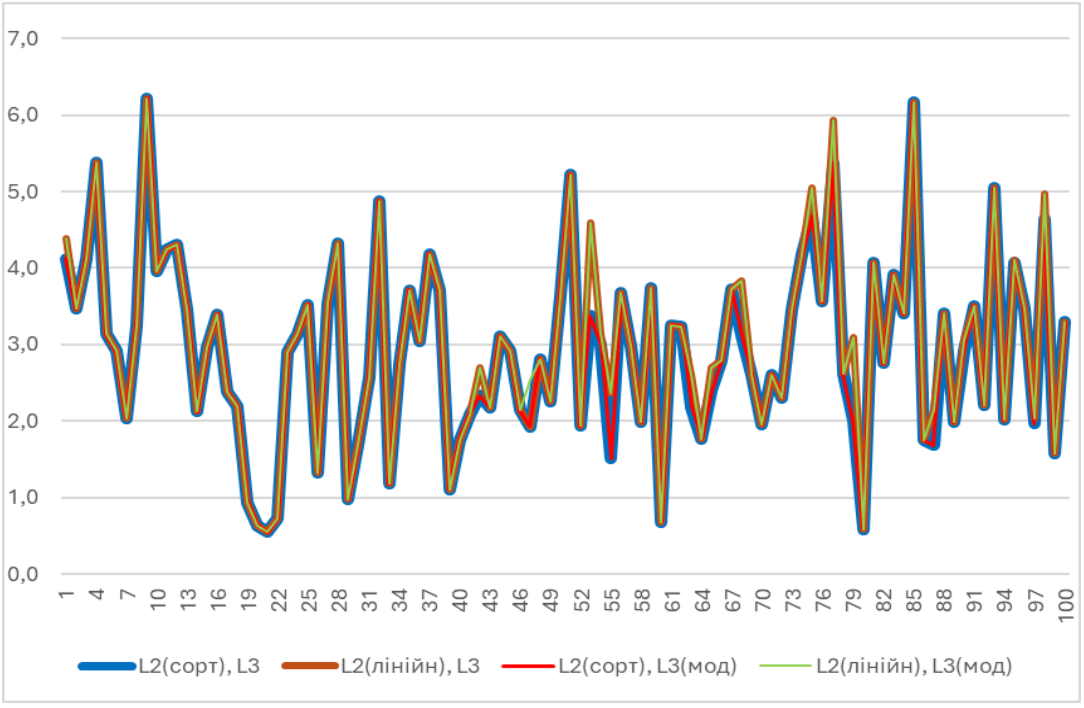


Рисунок 5.2 – Експериментальна оцінка похибки результату алгоритму A_2 в залежності від варіантів реалізації стратегій L_w

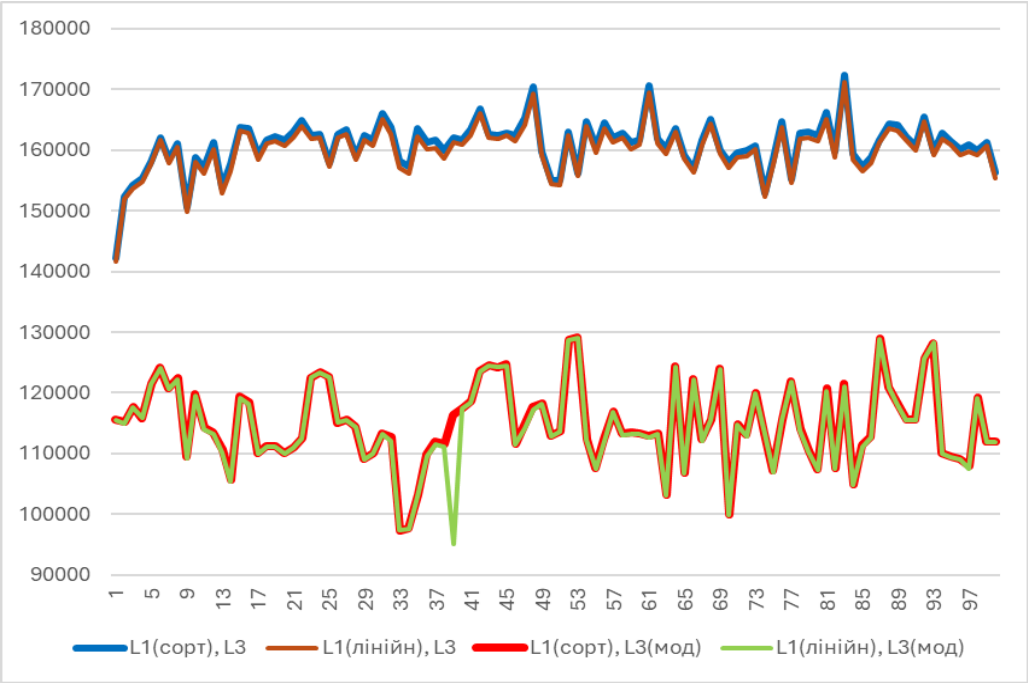


Рисунок 5.3 – Експериментальна оцінка кількості опрацьованих алгоритмом A_1 векторів в залежності від варіантів реалізації стратегій L_w

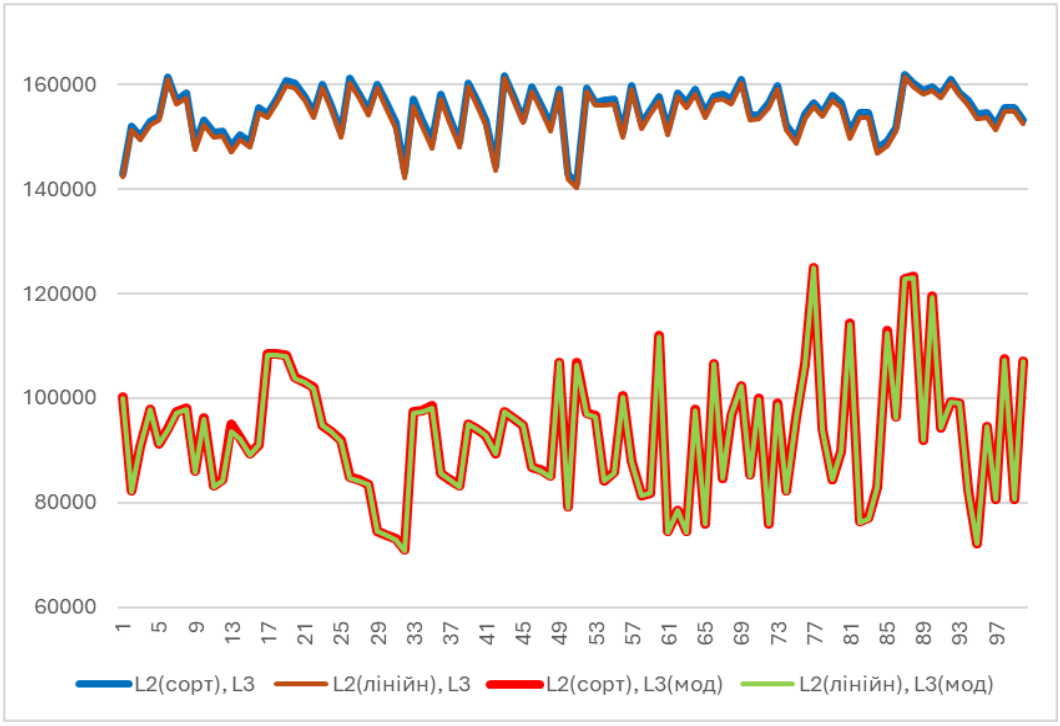


Рисунок 5.4 – Експериментальна оцінка кількості опрацьованих алгоритмом A_2 векторів в залежності від варіантів реалізації стратегій L_w

Друга частина експерименту зосереджена на порівнянні ефективності роботи алгоритму A_2 через лінійний пошук з модифікованою L_3 (як найвигіднішого) з жадібним алгоритмом, жадібним з добором, та алгоритмом Round Robin. Для останнього програма спрощується і задачі розкидаються на пристрої по колу, або просто підряд поки вміщаються (якщо запит прийшов тільки від одного пристрою).

Умови проведення:

- персональний компьютер з cpu Intel Core i3-9100 3,60 GHz, RAM 32 GiB;
- розмірність вхідного масиву – 1000;
- кількість тестувань – 50, для кожного алгоритму;
- умовний об’єм пам’яті кожного пристрою 150 Mb.

Задля оцінки ефективності використання методів вирішення задачі ЦЛП з БЗ використовуватимемо такі метрики:

- середнє завантаження: відсоток використання ресурсів на всіх процесорах за час симуляції. Вирахувані як середній показник з трьох різних замірювань;
- стандартне відхилення завантажень: корінь з дисперсії завантажень між процесорами. Чим воно нижче, тим більш рівномірним є розподіл.

Сандартне відхилення завантаження розраховується як:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2},$$

де:

x_i – завантаження *i*-го пристрою (у відсотках),

$\bar{x}$ – середнє завантаження всіх пристроїв,

n – кількість пристроїв.

Результати дослідження приведено у зведених графіках.

На рисунках приводяться чотири графіки – по одному на кожний з

алгоритмів. На рисунку 5.5 наведено результати експерименту, щодо середнього завантаження.

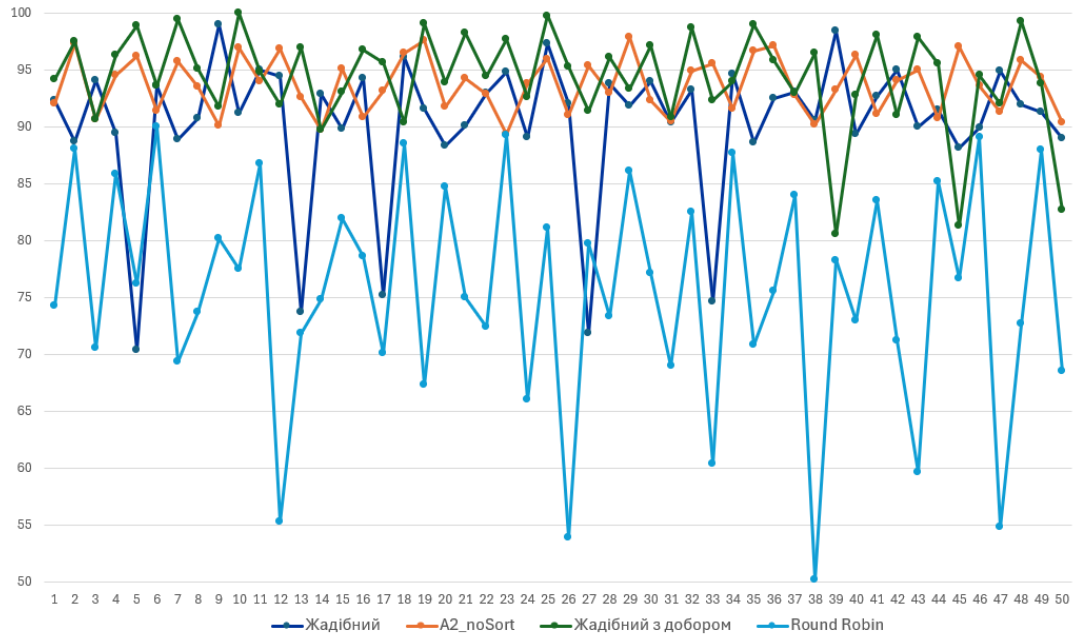


Рисунок 5.5 – Середні значення завантаження

На рисунку 5.6 приводяться дані експерименту по вирахування стандартного відхилення завантаження.

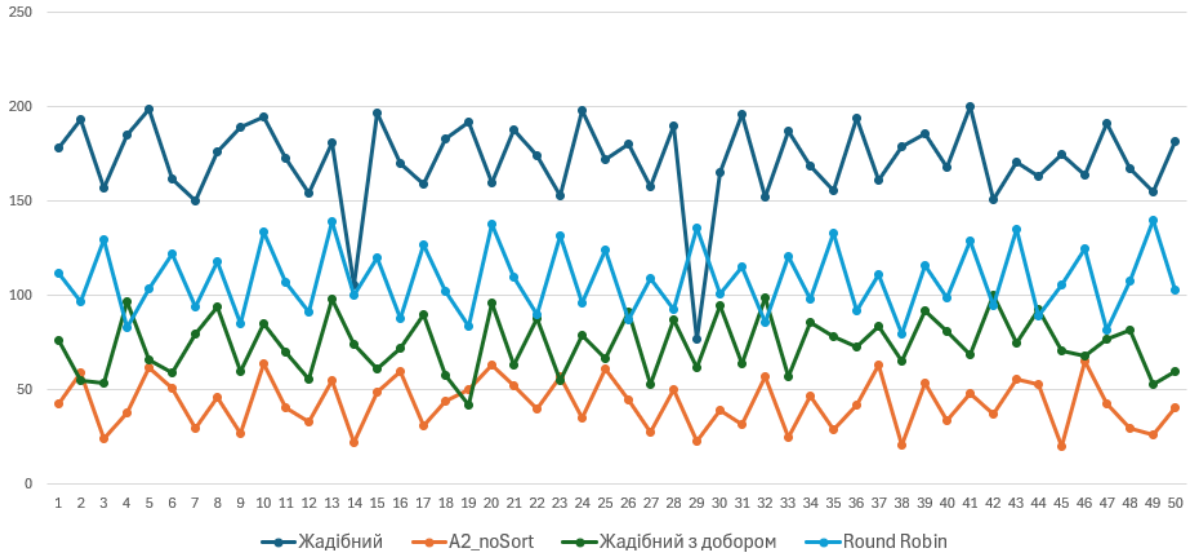


Рисунок 5.6 – Відхилення завантаження (стандартне відхилення)

На рисунку 5.5 в цілому видно, що запропонований алгоритм дає задовільні результати утримуючи планку завантаження на рівні 87-98%, що досягається завдяки орієнтованості на знаходження найдовшого рішення, в той час як решта алгоритмів більше залежать від випадку.

З рисунку 5.6 видно, що за метрикою стандартного відхилення запропонована реалізація методу на основі рангового підходу показує кращі результати порівняно з іншими використаними алгоритмами.

Причина такої поведінки та ж що і в попередньому експерименті. Орієнтованість запропонованого алгоритму на знаходження найдовшого з можливих векторів рішень, дозволяє ефективно завантажувати кожний з пристроїв, в той час як жадібні алгоритми можуть залишати значні прогалинин пристроях де «не пощастило» з набором завдань. В той же час можна побачити, що найбільш близьким за значеннями до запропонованого є жадібний алгоритм з донабором, який «добирає» задачі для заповнення місця залищеного вільним жадібним алгоритмом.

ВИСНОВКИ

В ході дослідження наводиться доказ того, що завдання планування розподілу ресурсів у багатопроцесорній мережі зводиться до задачі ЦЛП з БЗ.

Показано, що задача 0-1 рюкзак - класична варіація задачі ЦЛП з БЗ. Тобто рішення, застосовувані до неї можуть застосовуватися і до цільової проблеми дослідження. Означена задача рюкзак на момент проведення даного дослідження належить до класу NP повних. Однак виявлено що наразі наукова спільнота не дійшла висновку щодо еквівалентності класів P і NP. ці два фактори визначають актуальність пошуків будь-яких методів як точних так і наближених для її оптимального розв'язку.

В даній роботі увага була зосереджена на наближених методах пошуку рішення заснованих на ранговому підході та методі відсіювання неперспективних рішень. В ході дослідження було:

- запропоновано ряд програмних реалізацій складових частин для двох наближених алгоритмів пошуку рішення задачі про ранець;
- виявлено помилковість твердження згідно з яким стверджується, що при формуванні векторів будь-якої підмножини використовуючи процедуру A_0 вони будуть автоматично відсортовані за функціональною довжиною у порядку її зниження. Помилковість даного твердження не заважає роботі процедури чи побудованих на її основі алгоритмів, але може призводити до збільшення похибки відносно точного рішення;
- запропоновано модифікацію для стратегії завчасного відкидання варіантів, яка не впливаючи на точність рішення дозволяє суттєво скоротити кількість виконуваних операцій обробки векторів.

Аналіз результатів експериментальних досліджень для різних варіацій реалізації алгортмів A_1 та A_2 дає можливість дійти наступних висновків:

- похибка відносно точного рішення про застосування алгоритму A_1 приймає значення від 0 до 7,6. тобто максимальне значення похибка даного алгоритму на 1,3% вища за той же показник для алгоритму A_2 де максимальна похибка 6,3%. Приблизно в половині випадків виявлено що реалізація стратегії L_2 алгоритму A_2 впливає на значення похибки так, що при використанні сортування вона меншає в рамках півтора відсотка;

- будь-яка реалізація запропонованих алгоритмів серйозно зменшує кількість аналізованих векторів порівняно з методом грубої сили. При розмірності задачі 150, максимальна кількість векторів що розглядаються дорівнює 173000. При цьому використання алгоритму A_2 дозволяє знизити цю кількість ще на 10000, а використання модифікації L_3 - завчасно відкинути ще 40000;

- не виявлено значного впливу вибору стратегії L_1 або L_2 на роботу стратегії L_3, та на кількість аналізованих векторів. Різниця між варіантами із сортуванням та лінійним пошуком складає всього близько 1000 векторів.

Загалом запропоновані алгоритми рішення здатні знижувати обчислювальну складність задач, дозволяючи оброблювати меншу кількість векторів рішення при цьому забезпечуючи похибку меншу за 10%.

З огляду на результати дослідження рекомендується використовувати алгоритм A_2 з реалізацією через L_1 без сортування та модифікованою L_3, оскільки він розглядає найменшу кількість векторів при прийнятному значенні похибки до 6,3%. Однак якщо прогнозується ситуація коли при пошуку підходящого вектора-попередника буде відкинута значна кількість варіантів алгоритму але через сортування може виявитися кращим.

Рекомендований було порівняно з жадібним, жадібним з донабором, та циклічним алгоритмами, на спрощеній моделі мультипроцесорної системи. Виявлено що рекомендований алгоритм дозволяє підтримувати значення завантаження в рамках 90-98%, що робить його другим після жадібного з донабором, при цьому запропонований алгоритм має найкращі показники відхилення завантаження в рамках 60 мебібайт.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Mandal A., Pal S. An Empirical Study and Analysis of the Dynamic Load Balancing Techniques Used in Parallel Computing Systems // Proceedings of ICCS-2010, 19–20 листопада 2010 р. – 6 с. – [Електронний ресурс]. – Режим доступу: https://arxiv.org/abs/1109.1650?

2. Amaris M., Lucarelli G., Mommessin C., Trystram D. Generic algorithms for scheduling applications on heterogeneous multi-core platforms // Concurrency and Computation: Practice and Experience. – 2017. – Vol. 31, No. 3. – DOI: 10.1002/cpe.4647

3. De Giusti L., Chichizola F., Naiouf M., De Giusti A., Luque E. Automatic Mapping Tasks to Cores – Evaluating AMTHA Algorithm in Multicore Architectures // International Journal of Computer Science Issues (IJCSI). – 2010. – Vol. 7, No. 2. – March. – 6 p. – Режим доступу: https://arxiv.org/abs/1004.3254

4. Lifflander J., Pebay P. P., Slattengren N. L., Pebay P. L., Pfeiffer R. A., Kotulski J. D., McGovern S. T. A Communication- and Memory-Aware Model for Load Balancing Tasks // arXiv. – 2024. – 14 p. – DOI: 10.48550/arXiv.2404.16793.

5. Rathore O., Basden A., Chancellor N., Kusumaatmaja H. Load Balancing For High Performance Computing Using Quantum Annealing // Phys. Rev. Research. – 2025. – 19 p – DOI: 10.1103/PhysRevResearch.7.013067. – Режим доступу: arXiv:2403.05278

6. Goren G., Vargaftik S., Moses Y. Stochastic Coordination in Heterogeneous Load Balancing Systems // arXiv. – 2021. – 28 p. – DOI: 10.48550/arXiv.2105.09389.

7. Mohammed A., Cavelan A., Ciorba F. M., Cabezon R. M., Banicesu I. Two-level Dynamic Load Balancing for High Performance Scientific Applications // Proceedings of the 2020 SIAM Conference on Parallel Processing for Scientific Computing [Текст]. – 2020. – С. 69–80. – DOI: 10.1137/1.9781611976137.7.

8. Laabadi S., Naimi M., El Amri H., Achchab B. The 0/1 Multidimensional Knapsack Problem and Its Variants: A Survey of Practical Models and Heuristic Approaches // American Journal of Operations Research. – 2018. – Vol. 8. – P. 395–439.

9. Listrovoy S. V., Tretiak V. F., Listrovaya A. S. Parallel algorithms of calculation process optimization for the boolean programming problems // Engineering Simulation. – 1999. – Vol. 16, No. 5. – P. 569–579.

10. Третяк В., Голубничий Д., Коломійцев О., Мегельбей Г., Возний О., Філіпенков О. Математична модель рангового підходу // Збірник наукових праць ΛΌГОΣ. – 2020. – С. 116–122. – DOI: 10.36074/25.12.2020.v1.40. – Дата звернення: 18.01.2025.

11. Коломійцев О., Голубничий Д., Рибальченко А., Третяк В., Осієвський С., Возний О., Балабуха О., Качуровський Г., Грічанюк О., Галашевський Г., Сокова Т., Любченко О., Любченко О. Використання методів рангового підходу в моделі транзакційної системи з реплікацією фрагментів бази даних для розгортання у хмарному середовищі // Scientific Collection «InterConf+». – 2023. – № 38(175). – С. 326–341. – DOI: 10.51582/interconf.19-20.10.2023.030. – Дата звернення: 18.01.2025.

12. Голубничий Д., Коломійцев О., Осієвський С., Третяк В., Рибальченко А., Любченко О., Головченко О. Метод відсікання безперспективних варіантів для задач цілочисельного лінійного програмування з булевими змінними з використанням рангового підходу // Науковий збірник «InterConf+». – 2024. – № 41(185). – С. 526–555. – DOI: 10.51582/interconf.19-20.01.2024.064. – Дата звернення: 18.01.2025.

13. Коломійцев О., Голубничий Д., Рибальченко А., Третяк В., Осієвський С., Возний О., Балабуха О., Качуровський Г., Грічанюк О., Галашевський Г., Сокова Т., Любченко О., Любченко О. (2023). Використання методів рангового підходу в моделі транзакційної системи з реплікацією фрагментів бази даних для розгортання у хмарному середовищі // Scientific Collection «InterConf+». – 38(175), 326–341. – DOI:

10.51582/interconf.19-20.10.2023.030. – Дата звернення: 18.01.2025.

14. Голубничий Д. Ю., Головченко О. С. Застосування алгоритмів рангового підходу при плануванні розподілу задач в багатопроцесорних системах. Збірник науково-технічних праць «Радіотехніка». 2024. №219. С. 16-27. – DOI: 10.30837/rt.2024.4.219.02. – Дата звернення: 03.06.2025.

15. Abdel-Basset, M., Mohamed, R., Abouhawwash, M., Alshamrani, A. M., Mohamed, A. W., & Sallam, K. (2023). Binary light spectrum optimizer for knapsack problems: An improved model. Alexandria Engineering Journal, 67, 609-632. https://doi.org/10.1016/j.aej.2022.12.025. – Дата звернення: 21.02.2025.

16. Zhou, Y., Shi, Y., Wei, Y., Luo, Q., & Tang, Z. (2023). Nature-inspired algorithms for 0-1 knapsack problem: A Survey. Neurocomputing, 554(19), 126630. https://doi.org/10.1016/j.neucom.2023.126630. – Дата звернення: 21.02.2025

17. Durgut, R., & Aydin, M. E. (2020). Adaptive binary artificial bee colony algorithm. Applied Soft Computing, 101(1), 107054. https://doi.org/10.1016/j.asoc.2020.107054. – Дата звернення: 21.02.2025

18. Feng, Y., Yu, X., & Wang, G.-G. (2019). A Novel Monarch Butterfly Optimization with Global Position Updating Operator for Large-Scale 0-1 Knapsack Problems. MDPI Mathematics, 7(11), 1056. https://doi.org/10.3390/math7111056. – Дата звернення: 21.02.2025.

19. Liu, K., Ouyang, H., Li, S., & Gao, L. (2022). A Hybrid Harmony Search Algorithm with Distribution Estimation for Solving the 0-1 Knapsack Problem. Mathematical Problems in Engineering, 2022, 8440165. https://doi.org/10.1155/2022/8440165. – Дата звернення: 21.02.2025.

20. Goel, L. (2020). An extensive review of computational intelligence-based optimization algorithms: trends and applications. Soft Computing, 24(2). https://doi.org/10.1007/s00500-020-04958-w. – Дата звернення: 04.04.2025.

21. Голубничий Д. Ю., Листровой С. В. Алгоритм решения одномерной задачи (0,1) – рюкзак: сб. науч. трудов «Информационные системы». – Харьков: НАНУ, ПАНИ, ХВУ, 1995. – С. 59–62.

22. Golubnichy, D. Ur. Solution Method on the Basis of Rank Approach for Integer Linear Programming Problems with Boolean Variables / D. Ur. Golubnichy, S. V. Listrovoy, E. S. Listrovaya // Engineering Simulation. – 1999. – Vol. 16. – P. 707–725.