

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО

Факультет інформатики та обчислювальної техніки
(назва факультету, інституту)

Кафедра автоматизованих систем обробки інформації і управління
(назва кафедри)

"На правах рукопису"
УДК 519.854.2

«До захисту допущено»
Завідувач кафедри

О.А.Павлов
(підпис) (ініціали, прізвище)
“ ” 20 18 р.

МАГІСТЕРСЬКА ДИСЕРТАЦІЯ
на здобуття ступеня магістра

за спеціальністю 122 Комп'ютерні науки та інформаційні технології
(код та назва спеціальності)

спеціалізацією Інформаційні управляючі системи та технології
(код та назва спеціалізації)

на тему: Формалізація і розв'язання задач оптимального планування робіт
за наявності різної продуктивності пристроїв

Виконав: студент VI курсу групи ІС-63м
(шифр групи)

Галкіна Галина Андріївна
(прізвище, ім'я, по батькові)

(підпис)

Науковий керівник проф., д.т.н., с.н.с. Гуляницький Л. Ф.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант к.т.н., доц. Жданова О.Г.
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації
немає запозичень з праць інших авторів без
відповідних посилань.

Студент _____
(підпис)

РЕФЕРАТ

Магістерська дисертація: 107 с., 12 рис., 10 табл., 7 додатків, 82 джерела.

Актуальність. Планування виконання командою наявних завдань є важливим процесом в багатьох галузях, наприклад, у розробці програмного забезпечення. На сьогодні спостерігається зростання популярності покрокового (ітеративного) підходу до виконання робіт у різних сферах нашого життя. Скрам є одним із найбільш поширених гнучких підходів на сьогоднішній день. Ідея методології Скрам полягає у роботі за ітераціями, тобто за деякими фіксованими проміжками часу. У Скрамі ітерації називаються Спринтами. Для ітерації необхідно підібрати набір завдань, які може виконати за цей проміжок часу команда, причому саме такий набір, який принесе найбільшу цінність продуктові, що розробляється. Але обговорення завдань та вирішення, які саме завдання можна взяти на виконання з урахуванням різної продуктивності та досвідченості виконавців, є складним процесом, який займає досить багато часу.

Саме тому актуальним є дослідження проблеми оптимального планування виконання завдань, формальна постановка якої призводить до складних оптимізаційних задач. В свою чергу це потребує розробки наближених алгоритмів розв'язування задачі виконання завдань виконавцями з різною для досягнення найбільшої сумарної цінності виконаної роботи. Враховуючи наявну в теорії складання розкладів термінологію та специфіку задачі, вживатимемо терміни “пристрої” та “виконавці” як взаємозамінні.

Мета дослідження – підвищення ефективності виконання завдань декількома виконавцями (пристроями) з різною продуктивністю за рахунок зменшення витрат часу на планування їх виконання.

Для досягнення мети необхідно виконати наступні **завдання**:

- виконати огляд відомих результатів з поставленої задачі;
- виконати формалізацію задачі планування роботи із врахуванням різної продуктивності пристроїв;
- розробити наближені алгоритми для розв'язування поставленої задачі;

- розробити програмну реалізацію алгоритмів та моделей;
- виконати аналіз отриманих результатів.

Об’єкт дослідження – процес планування виконання завдань пристроями з різною продуктивністю.

Предмет дослідження – методи планування виконання завдань пристроями з різною продуктивністю.

Наукова новизна отриманих результатів полягає у формалізації задачі планування роботи на ітерацію у методології Скрам як задачі оптимального планування робіт за наявності різної продуктивності пристроїв; розробці алгоритму для її розв’язування шляхом розбиття на підзадачі; розробці жадібного алгоритму знаходження початкового розв’язку другої підзадачі, процедури генерації точок околу в просторі розв’язків та розробці алгоритмів на основі схеми алгоритмів локального пошуку.

Публікації. Матеріали роботи опубліковані у міжнародному журналі «Науковий огляд», №3, 2018 [1, 2].

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалась у філії кафедри автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського» в рамках науково-дослідної теми Інституту кібернетики ім. В. М. Глушкова НАН України: «Розробити математичний апарат, орієнтований на створення інтелектуальних інформаційних технологій розв’язування проблем комбінаторної оптимізації та інформаційної безпеки» (шифр теми: ВФ.180.11).

ОПТИМАЛЬНЕ ПЛАНУВАННЯ, ГНУЧКІ МЕТОДОЛОГІЇ, СКРАМ, ПРИСТРОЇ З РІЗНОЮ ПРОДУКТИВНІСТЮ, ДЕТЕРМІНОВАНИЙ ЛОКАЛЬНИЙ ПОШУК, АЛГОРИТМ ІМІТАЦІЙНОГО ВІДПАЛУ, G-АЛГОРИТМ

ABSTRACT

Master's thesis: 107 pages, 12 figures, 10 tables, 7 appendix, 82 references.

Relevance. Tasks scheduling for a team is an important process in many spheres like software development. Nowadays the iterative approach to work is gaining more and more recognition in different spheres. Scrum is one of the most used agile approaches today. The main idea of Scrum is splitting the work into iterations, where iterations are time spans of fixed length. In Scrum, these iterations are called Sprints. For each iteration, it is necessary to choose such a subset of tasks for the team that the work's result will have the biggest possible value for the product. But tasks discussion and making decisions about which tasks to include in the current iteration in accordance with the different productivity and experience of team members is a sophisticated process that takes a lot of time.

That's why the research of the optimal scheduling problem formal model of which results in difficult optimization problems is relevant. That requires development of approximate algorithms for solving the scheduling problem with performers having different productivity with the goal of maximizing the work's result value. Taking into account the terminology of scheduling problems sphere and the problem's specifics we will use the terms "performer" and "machine" as synonyms.

Purpose and objectives of the study. Increasing the effectiveness of tasks finishing by several unrelated performers(machines) by reducing the time spent on the planning of work.

To achieve this purpose it is needed to complete these tasks:

- perform a review of the known results for the problem that is considered;
- perform formalization of the optimal scheduling problem for the unrelated machines;
- develop approximate algorithms for solving the problem considered;
- develop a software implementation of the algorithms and models;
- perform the analysis of the results.

The object of study is the scheduling process for the unrelated machines with different productivity.

The subject of study are scheduling methods for the unrelated machines with different productivity.

Scientific novelty of the results. Formalization of the Sprint planning problem as a scheduling problem for machines with different productivity is performed, an approach to solving this problem based on splitting the problem into two subproblems is suggested; a greedy algorithm, local search algorithms and neighborhood generation procedure for the second subproblem are developed.

Publications. Materials were published in the international journal “Naukoviy ohlyad”, №3, 2018 [1, 2].

Connection of the thesis with scientific programs, plans, topics. The thesis was written at the branch of The Department of Department of Computer-aided management and data processing systems of the National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute” at the V. M. Glushkov Institute of Cybernetics of the National Academy of Sciences of Ukraine under the topic “To develop a mathematical apparatus focused on the creation of intelligent information technologies for solving combinatorial optimization and information security problems”(the topic’s index is BΦ.180.11).

OPTIMAL SCHEDULING, AGILE METHODOLOGIES, SCRUM, UNRELATED MACHINES, LOCAL SEARCH, SIMULATED ANNEALING, G-ALGORITHM

ЗМІСТ

Вступ	9
1 Огляд сучасного стану проблеми	11
1.1 Ітеративна методологія Скрам та задачі планування	11
1.2 Існуючі алгоритми розв’язання задачі планування Спринту.....	15
1.3 Задачі комбінаторної оптимізації, елементи яких складають задачу про планування Спринту	18
1.3.1 Задача пакування декількох рюкзаків.....	18
1.3.2 Узагальнена задача про призначення	22
1.3.3 Задача складання розкладів для паралельних пристроїв з різною непропорційною продуктивністю та наявністю директивного терміну.....	25
Висновок до розділу.....	30
2 Постановка та формалізація задачі	31
2.1 Системний аналіз проблематики	31
2.2 Змістовна постановка задачі	32
2.3 Математична модель задачі.....	33
2.4 Декомпозиційний підхід.....	34
Висновок до розділу.....	40
3 Алгоритми розв’язування задачі	42
3.1 Алгоритм розв’язування підзадачі про розподіл обов’язкових завдань..	42
3.2 Алгоритми розв’язування підзадачі планування необов’язкових завдань	43
3.2.1 Жадібний алгоритм знаходження початкового припустимого розв’язку	43
3.2.2 Алгоритм детермінованого локального пошуку.....	45
3.2.3 Алгоритм імітаційного відпалу	56
3.2.4 G-алгоритм	59
Висновок до розділу.....	62
4 Опис розробленого програмного забезпечення.....	64

4.1 Призначення програмного забезпечення	64
4.2 Засоби розробки.....	64
4.3 Опис програмної реалізації	67
Висновок до розділу.....	75
5 Дослідження ефективності алгоритмів.....	77
5.1 Процес проведення експериментів	77
5.1.1 Постійні параметри вхідних даних	78
5.1.2 Змінні параметри вхідних даних	78
5.1.3 Параметри алгоритмів	79
5.1.4 Характеристики апаратного забезпечення	80
5.2 Співвідношення між кількостями обов'язкових та необов'язкових завдань	80
5.3 Кількість впорядкованих завдань.....	83
5.4 Розкид тривалості завдань	86
Висновок до розділу.....	90
Висновки.....	91
Перелік посилань	93
ПЛАКАТ 1 Блок-схеми алгоритмів локального пошуку.....	101
ПЛАКАТ 2 Блок-схема процедури генерації розв'язку з околу з використанням зсувів ввєрх та вниє	102
ПЛАКАТ 3 Схема структурна послїдовностї	103
ПЛАКАТ 4 Схема структурна класїв.....	104
ПЛАКАТ 5 Схема структурна пакетїв.....	105
ПЛАКАТ 6 Результати експериментїв для жадїбного алгоритму	106
ПЛАКАТ 7 Результати експериментїв для алгоритмїв локального пошуку ..	107

ВСТУП

Задачі планування проектів на сьогоднішній день є одними з найбільш розповсюджених через теоретичну важливість та практичне застосування. Одним із різновидів підходів до планування проектів є гнучкі методології, які нині набувають все більшої популярності. Вони необхідні у сучасному світі, оскільки дуже часто неможливо запланувати наперед всю роботу, яку потрібно виконати на проекті. В сучасних умовах гнучкість є вже необхідною умовою ефективної роботи.

Сенс гнучкого підходу до роботи над проектом полягає в тому, що сама робота проводиться ітеративно, тобто її розбивають на деякі етапи, які виконуються поступово. Це дозволяє зменшити вартість виправлення помилок та додавання нових завдань, тобто адаптуватися до нинішніх умов. Але жодні переваги не приходять без недоліків, і одним із головних недоліків такої гнучкості є великі затрати часу на адаптацію. Адаптація до мінливих умов може бути подана у вигляді регулярних зустрічей команди, постійних переформулювань існуючих завдань, тощо. Все це потребує великої кількості часу, а час зазвичай є важливим ресурсом, який є обмеженим. Саме тому, якщо існують способи збереження цього ресурсу, необхідно ними користуватися, а також розробляти нові способи. Ця проблема є дуже актуальною на сьогоднішній день, вона є темою багатьох наукових досліджень та прикладних проектів.

У гнучкому підході для збереження часу корисним буде розв'язування задачі про планування робіт по проекту на деякий обмежений час (ітерацію). Ця задача має велику кількість різноманітних варіацій, які залежать як від теоретичного представлення, так і від конкретної практичної ситуації, до якої її буде застосовано. Враховуючи існуючу в теорії розкладів термінологію і специфіку нашої задачі, будемо вживати терміни виконавці та пристрої, як взаємозамінні

В дисертації розглядається спеціальний вид такої задачі. Він включає в себе наявність поділу завдань на обов'язкові та необов'язкові, відношення часткового передування між завданнями, загального для всіх завдань директивного терміну та

різної непропорційної продуктивності виконавців. Всі ці параметри мають практичний зміст, вони впливають з практичних потреб.

Такий варіант задачі планування роботи по проекту на обмежений час містить в собі елементи задачі складання розкладів, задачі пакування декількох рюкзаків, задачі про призначення, тощо. Зазначена постановка відрізняється від відомих у науковій літературі задач, тому її розв'язування є актуальним з наукової точки зору. В роботі запропоновано алгоритми розв'язування задачі, зокрема алгоритми локального пошуку, які є відомим класом алгоритмів для розв'язування задач комбінаторної оптимізації.

1 ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ

1.1 Ітеративна методологія Скрам та задачі планування

Ітеративні методології розробки програмного забезпечення стають дедалі більш розповсюдженими у сучасному світі. Всі такі методології базуються на підході, який включає в себе виконання обмеженого обсягу завдань для проекту за певний час, що називається ітерацією.

Розглянемо ітеративну методологію, що має назву Скрам.

Скрам – це підхід, в рамках якого можливо вирішити складні адаптивні проблеми, і в той же час продуктивно та із застосуванням творчого підходу розробити продукт найвищої якості. Скрам складається зі Скрам Команд (Scrum Teams), в яких розподілено відповідні ролі (roles), а також церемоній (events), артефактів (artifacts) та правил (rules). Серцем Скрам є Спринт, з часовими рамками в місяць або менше, в результаті якого створюється “завершений”, цінний та потенційно готовий до випуску Інкремент продукту. Тривалість Спринту є постійною протягом усього періоду розробки. Кожен Спринт може вважатися проектом із часовими рамками в межах одного місяця. Як і інші проекти, Спринт використовується для досягнення певних цілей. Кожен Спринт складається із визначення того, що потрібно розробити, дизайну та гнучкого плану, які є орієнтирами при розробці, роботи та власне продукту, що стане результатом цієї роботи [3].

Скрам ґрунтується на теорії управління емпіричними процесами, або емпіризмі. Емпіризм стверджує, що знання приходить із досвідом та прийняттям рішень на підставі того, що є відомим. Скрам використовує ітеративний, інкрементальний підхід для оптимізації прогнозованості та управління ризиками.

На рисунку 1.1 наведено схему звичайного Спринту.

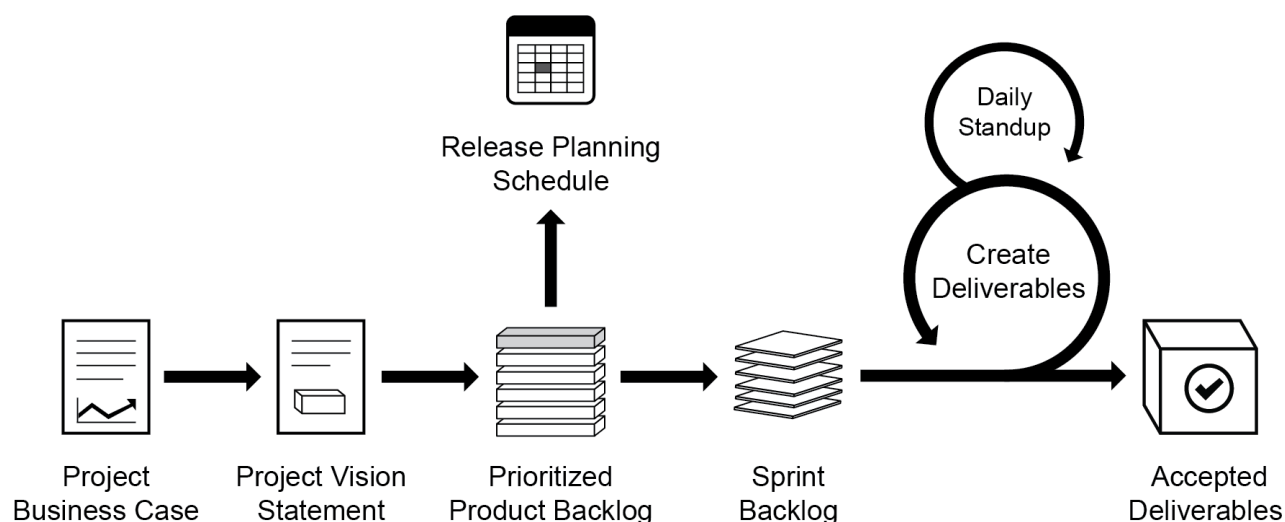


Рисунок 1.1 – Схема одного класичного Спринту

Робота, що її виконують під час Спринту, планується під час наради із Планування Спринту. План дій розробляється при спільній роботі цілої Скрам Команди. Для Спринту ще перед початком роботи над проектом обирається деяка довжина. Як було вказано раніше, це зазвичай складає від двох до чотирьох тижнів.

До початку планування має бути підготовлений список завдань, які треба зробити по проекту на поточний момент. В термінології Скрам цей список називається беклогом продукту. Це можуть бути завдання з будь-яких частин проекту. Деякі з цих завдань можуть бути обов'язковими до виконання якнайшвидше, вони будуть позначені відповідним чином. Також роботу над деякими завданнями можна буде розпочати лише після виконання інших завдань. Для кожного завдання має бути вказано критерій, за яким можна визначити, чи це завдання завершене, та всю необхідну для його виконання інформацію, яка відома на поточний момент [4].

Кожен з учасників команди має до початку планування оцінити час, який він витратить на виконання кожного з завдань. Цю інформацію також буде враховано при плануванні. Також на початок планування має бути підготовленою інформація про продуктивність команди. Для складання цієї характеристики використовується досвід попередніх Спринтів. Це допоможе реалістично оцінити обсяг роботи, який команда зможе виконати у Спринті, що планується.

Поширеною практикою є впорядкування завдань за тим, наскільки вони важливі та скільки користі та прибутку принесуть. Назвемо сукупність цих факторів важливістю завдання.

Звичайно ж існують деякі обмеження на завдання та можливість їх виконати. Ці обмеження можуть виходити як з бізнес-сторони, так і з технічної сторони. Інформація про наявні обмеження необхідна для коректного планування. В залежності від обмежень деякі завдання можуть бути вилучені з планування.

Останній фактор – можливості команди. Він включає перелік членів команди, навички, які вони мають, та час, який вони зможуть працювати на протязі Спринту [4].

Задача команди на плануванні – вибрати такий набір завдань на Спринт, що цей набір завдань принесе найбільшу користь та прибуток проекту, який розробляється.

У звичному розумінні планування Спринту – це зустріч, на якій переглядаються усі завдання та вибирається деякий їх набір до виконання. Існує два найпопулярніших підходи до планування Спринтів, планування з двох частин та планування з однією загальною частиною. Розглянемо їх докладніше.

При плануванні з двох частин зустріч розбивається на два етапи. Під час першого етапу команда переглядає всі завдання, які містяться у списку завдань проекту, та вибирає деякий їх набір, який команда вважає можливим зробити. Цей перший етап ще називають етапом “Що”. Другий етап (“Як”) полягає в тому, що команда переглядає обраний набір завдань на Спринт та розбиває його на менші точніші завдання за необхідності. Також команда створює план виконання всіх обраних завдань. Після цього переглядається отриманий план та порівнюється потенційний час його виконання із довжиною Спринту та за необхідності обраний список завдань коригується.

Другий спосіб – планування з однієї частини – є комбінацією всіх дій, які виконуються за два етапи, в один. Він використовується найчастіше та водночас є складнішим для учасників команди, бо необхідно постійно переключатися з оцінки

на додавання завдання до Спринту, а з того на розбиття на менші завдання. Цей підхід гірше структурований, але легший для використання на практиці [4].

Також необхідно вказати, що у формальному визначенні планування Спринту не вказано, що завдання мають бути приписані до конкретних учасників команди під час планування. Проте на практиці при плануванні Спринту команда має приблизно уявляти обсяг роботи, який необхідно виконати. Для найкращого досягнення даної мети необхідно оцінити рівень навичок та підготовки кожного з учасників та відштовхуватися від найкращого випадку. Щоб мати уявлення про цей найкращий випадок, важливо спробувати розподілити завдання між учасниками та виявити, чи можливо взагалі їх розподілити та дотриматися всіх обмежень та вкластися у довжину Спринту. Якщо це неможливо, то і відібраний набір завдань треба змінювати (в основному ці зміни полягають у вилученні деяких найменш важливих завдань).

Для Спринту тривалістю в місяць часові рамки зустрічі становлять вісім годин. Для більш коротких Спринтів на планування виділяють менше часу, пропорційно загальній довжині Спринту. Приміром, для двотижневого Спринту планування займе не більше чотирьох годин [3].

Це дуже велика кількість часу, а якщо враховувати, що кількість зустрічей у даній методології є досить великою, то отримуємо витрати часу та сил команди невиправдано високими [5]. У зв'язку з цим у деяких джерелах пропонують навіть виділяти час перед самим плануванням на те, щоб якнайкраще до нього підготуватися [6]. Але проблема в тому, що зустріч для підготовки до планування також вимагає витрат часу.

Саме тому автоматизація процесу планування роботи для одного Спринту принесе значну економію часу команди, яка працює над проектом. Таким чином, формалізація задачі планування роботи на ітерацію є актуальною з практичної точки зору.

1.2 Існуючі алгоритми розв'язання задачі планування Спринту

Задача планування Спринту є досить мало дослідженою на сьогоднішній день. Це пов'язано в основному із її трудомісткістю та залежністю вхідних даних від складу команди та особливостей її учасників. Але навіть за наявності цих складнощів було зроблено спроби автоматизувати процес планування Спринту.

Один із способів розв'язування задачі планування Спринту наведено у [7]. Автори обґрунтовують необхідність автоматизації планування Спринту тим, що при плануванні роботи на ітерацію командою можливі ситуації, в яких учасники команди не всі можливі обмеження візьмуть до уваги. Через це виникатимуть затримки по виконанню проекту та втрати ресурсів, яких можна було б уникнути. Також специфічним є те, що у [7] задача планування Спринту розглядається у контексті конкретно проектів для сховищ даних [8].

Формально задачу планування Спринту у [7] зведено до задачі складання декількох рюкзаків з додатковими обмеженнями [9] та розв'язано за допомогою інструменту IBM ILOG CPLEX Optimizer [10].

У [7] запропоновано постановку задачі, яка враховувала наступні фактори:

- ризики;
- взаємозв'язок між завданнями;
- залежність між завданнями;
- продуктивність команди за день.

З цілей оптимального розкладу виділено задоволення користувачів, керування взаємозв'язками між завданнями та керування ризиками.

Автори [7] пропонують використати IBM ILOG CPLEX Optimizer, який розв'язував сформульовану ними задачу методом гілок та меж [11], модифікованим за допомогою методу відсікаючих площин [12].

Недоліком даного підходу є те, що знаходження точного розв'язку поставленої задачі зі зростанням її розмірності займає дедалі більше часу та вимагає більше ресурсів (пам'ять комп'ютера). Також у постановці задачі вказано, що є можливим розробити план на декілька Спринтів вперед. Проте при використанні

ітеративних підходів ми завжди маємо враховувати те, що умови постійно змінюються та обсяг роботи не є фіксованим. Тому все одно довелося б перепланувувати те, що було заплановано раніше.

Трохи інший підхід до задачі планування Спринту запропоновано у [13]. Автори [13] також вважають, що автоматизація процесу планування Спринту надає великий вигаш у часі порівняно із проведенням планування вручну командою. У [13] стверджується, що при великій розмірності задачі пошук точного розв'язку не є обов'язковим та можна скористатися деяким евристичним підходом замість методу гілок та меж.

У [13] враховується лише дві цілі оптимального розкладу:

- задоволення користувачів;
- керування ризиками.

На відміну від [7], у [13] задачу планування Спринту зведено до задачі про призначення [14]. Набір Спринтів виступає у ролі виконавців, а завдання на Спринти – завданнями для виконавців. Таким чином, за допомогою жадібного підходу (оптимізації для кожного зі Спринтів) для врахування критичних для проекту завдань автори [13] отримують розв'язання задачі. Спочатку розглядаються критичні завдання (в цьому і полягає жадібність алгоритму), і для кожного зі Спринтів по черзі розв'язується задача про складання рюкзака з обмеженнями. В даному випадку задача про рюкзак виступає як підзадача. Також необхідно відмітити, що у [13] також використано IBM ILOG CPLEX Optimizer.

Недоліком такого підходу також є те, що планується робота на декілька Спринтів одразу. Також необхідно відмітити, що необов'язково буде отримано точний або дуже близький до точного розв'язок. Якщо буде багато Спринтів, то розв'язування може затягнутися.

Ще одна робота на дану тему – [15]. У ній обрано дещо інший погляд на планування Спринту. Постановка задачі відрізняється наступними характеристиками:

- є декілька команд, між якими необхідно розділити завдання;
- команди працюють над завданнями різних типів.

Важливість завдань, які можна включити до Спринту, визначається за декількома критеріями:

- фінансовий;
- організація роботи;
- негативний (втрата важливості);
- якість програмного забезпечення;
- важливість для користувачів;
- важливість для розробників [15].

Як видно з обраних у [15] критеріїв, постановка задачі в даній роботі максимально близька до задачі розробки програмного забезпечення.

Після встановлення даних критеріїв обраховано вагові коефіцієнти, які дозволяють врахувати всі ці критерії.

Ще однією особливістю даної публікації є розділення завдань на технічні та нетехнічні та встановлення взаємозв'язку між ними.

Як і у [7], задачу планування Спринту зведено до задачі про пакування рюкзака. Щоправда, у [15] розглядається саме задача пакування одного рюкзака, не багатьох, як у [7]. Також наведено доведення того, що задача про планування Спринту є NP-складною, шляхом зведення її до задачі пакування рюкзака.

Сам алгоритм розв'язування сформульованої у [15] задачі є жадібним. Переглядаються всі нетехнічні завдання по черзі, причому вони були попередньо впорядковані за незростанням важливості, яку було визначено за згаданими вище ваговими коефіцієнтами. Для кожного завдання встановлюється зв'язок із відповідними технічним завданнями, після чого вони додаються до обсягу роботи конкретної команди розробників [15].

Недоліком даного підходу є те, що жадібний алгоритм сам по собі в більшості випадків не може дати достатньо близького до точного розв'язку. В подальшому можливо використати даний алгоритм для знаходження припустимого розв'язку та покращення його іншим евристичним алгоритмом. Також сформульована в [15] задача є досить специфічною, оскільки включає в себе всі етапи не тільки планування Спринту, але й розділення задач на технічні задачі. Це не буде

застосовним у всіх випадках, адже зазвичай учасники команди на момент планування Спринту вже мають всі технічні завдання, які їм необхідні.

У [16] також розглянуто задачу планування Спринту. Для неї запропоновано евристику Лагранжа, яка базується на послабленні початкової моделі цілочислового програмування та використанні деяких жадібних алгоритмів. Необхідність такої евристики обумовлено тим, що розв'язування за допомогою таких інструментів, як IBM ILOG CPLEX, займає багато часу та є сенс розробляти швидші евристичні алгоритми. У цій роботі також розглядається задача з багатьма обмеженнями, як-то спорідненість завдань чи оцінена складність завдань. Постановка задачі в цій роботі також відрізняється від тієї, яка розглядається у [16] наявністю обов'язкових та необов'язкових завдань та часткової впорядкованості завдань.

Загалом можна сказати, що задача планування Спринту є достатньо мало дослідженою на сьогоднішній день. Існують різні моделі та представлення цієї задачі, які враховують відмінні практичні аспекти цього процесу планування, та є й інші аспекти, які ще не відображені в існуючих роботах. Тому розробка нових формальних моделей для задачі планування Спринту є актуальною.

1.3 Задачі комбінаторної оптимізації, елементи яких складають задачу про планування Спринту

З публікацій, розглянутих вище, можна зробити висновок, що зазначена проблема являє собою комбінацію задачі пакування декількох рюкзаків та задачі про призначення. Справді, результатом розв'язування сформульованої задачі є розклад, який включає в себе вибрані за важливістю завдання та призначення їх учасникам команди. Розглянемо існуючі способи розв'язування кожної з задач, зазначених вище.

1.3.1 Задача пакування декількох рюкзаків

Задачу пакування декількох рюкзаків можна сформулювати наступним чином: маємо набір речей та набір рюкзаків. Кожна річ має важливість та вагу, кожен

рюкзак – місткість. Необхідно спакувати наявні рюкзаки таким чином, щоб максимізувати сумарну важливість речей, які до них потрапили [9]. Спостерігаємо явну аналогію між рюкзаками та робочим часом кожного учасника команди, також кожна річ має вагу (аналогія з часом, який витрачається на виконання завдання) та важливість.

Для знаходження точних розв'язків задачі про пакування декількох рюкзаків зазвичай використовують метод гілок та меж. Існує багато варіацій цього методу для задачі, що розглядається. У [17] запропоновано варіант методу гілок та меж, який базується на пошуку вглиб. Для цього додається ще один додатковий рюкзак, додавання речі в який визначає виключення цієї речі з кінцевого розв'язку. Для кожної речі переглядається кожен рюкзак (включаючи додатковий) та будується дерево з цих варіантів, поступово відкидаючи неприпустимі варіанти.

Наступний спосіб розв'язування задачі про пакування багатьох рюкзаків розглянуто у [18], він відомий під назвою евристики Martello та Toth. Даний спосіб складається з трьох етапів:

1. Знаходження припустимих розв'язків для кожного з рюкзаків окремо. Це досягається застосуванням жадібного алгоритму для кожного з рюкзаків по черзі.

2. Спроби покращення знайдених на першому етапі розв'язків шляхом спроби обміну кожної пари речей, які були додані до різних рюкзаків, та після цього спробою додати ще якісь речі, які принесуть збільшення сумарної цінності.

3. Спроби видалити якісь речі з поточних розв'язків та замінити видалені речі деякими іншими, які принесуть збільшення сумарної цінності речей, що вже у рюкзаках.

Основним недоліком такого підходу є те, що розглядаються обміни лише пар речей, а не їх комбінацій [18].

У [18] запропоновано інший підхід до задачі про пакування кількох рюкзаків. Він, як і евристика Martello та Toth, складається з кількох етапів. Для кожного крім останнього рюкзака спочатку відбувається його заповнення речами за жадібним алгоритмом. Після цього перевіряється, чи є ще місце у цьому рюкзаку. Якщо є, то

за допомогою динамічного програмування розв'язується задача суми підмножини, де сумою є вільне місце у рюкзаку. Якщо буде знайдено кращий розв'язок, то він використовується для рюкзака, що розглядається. Останній рюкзак заповнюється за допомогою алгоритму динамічного програмування [18].

У [19] запропоновано алгоритм розв'язування задачі про пакування кількох рюкзаків за наявності великої розмірності задачі. Запропонований у [19] алгоритм є точним, він є рекурсивним різновидом методу гілок та меж. Основною відмінністю цього алгоритму є рекурсія. Алгоритм полягає в тому, що для початку ми для кожного з рюкзаків розв'язуємо задачу суми підмножини. Після цього ми розв'язуємо послаблену задачу для сумарної місткості всіх рюкзаків. Якщо розв'язок не відкидається, то ми розбиваємо його на всі рюкзаки шляхом розв'язування задачі суми підмножини для рюкзаків. Після цього ми складаємо найменший рюкзак та переходимо до наступної речі [19].

У [19] наведено результати експериментів для великої розмірності задачі показано, що даний алгоритм дозволяє зменшити витрати часу на розв'язування порівняно з іншими алгоритмами. Проте через точність він таки займає багато часу й менш точні евристики можуть працювати швидше.

Наступною роботою, в якій запропоновано алгоритм розв'язування задачі про пакування декількох рюкзаків, є [20]. Однією з особливостей задачі пакування кількох рюкзаків у цій публікації є те, що в ній також наявні обмеження на призначення. Іншими словами, для кожної речі вказаний конкретний набір рюкзаків, в які цю річ можна покласти. Така задача є спеціальним випадком узагальненої задачі про призначення.

Виходячи з потреб авторів [20], у цій публікації запропоновано швидкі евристики. Загальна для всіх наведених у [20] евристик полягає у наступному:

1. Почати з заданої задачі як початкової.
2. Знайти розв'язок задачі лінійного програмування, яка виключає з початкової задачі обмеження на призначення.

3. Записати частину розподілу речей за рюкзаками з отриманого на кроці 2 розв'язку. Яку саме частину, визначає конкретна з евристик, запропонованих у публікації.

4. Вилучити з розгляданих речей всі, які вже призначено до рюкзаків. Врахувати зменшений обсяг вільного місця у кожному рюкзаку, в який додалася хоча б одна річ.

5. Переглянути можливості: в деяких рюкзаках може не вистачати місця для деяких речей після кроку 4 [20].

Перша з евристик (DET) полягає в тому, щоб додати до розподілу всі речі, яку було на кроці 2 додано до рюкзаків, та які не порушують обмежень на призначення.

Друга евристика (RAN) полягає у використанні випадкової ймовірності для додавання речі до розв'язку. Для того, щоб забезпечити припустимість розв'язку, який отримується з випадковими змінними, використовується жадібний алгоритм – іншими словами, додаються лише речі, які не перевершують місткість рюкзака.

Третя запропонована у [20] евристика (COMBI) полягає у поєднанні попередніх двох. Вона не дає значного виграшу порівняно з ними. За результатами експериментів з [20] видно, що найкращі результати дає евристика RAN.

Евристики з [20] можуть бути успішно використані для розв'язування задачі про пакування кількох рюкзаків за наявності обмежень на призначення. Але задача планування Спринту не має таких обмежень, а її особливості полягають в інших характеристиках (наприклад, різна продуктивність учасників для кожного завдання). Тому в розробці евристик для задачі планування Спринту запропоновані в [20] евристики в чистому вигляді використовувати не можна.

У [21] розглянуто спеціальну версію задачі пакування одного рюкзака, яка має багато спільного з проблемою, яка розглядається, оскільки в цій задачі речі є частково впорядкованими. У [21] наведено поліноміальний алгоритм для одного спеціального випадку цієї задачі, а також розглянуто потенційний зв'язок такої задачі з задачами складання розкладів.

Задачу про складання декількох рюкзаків розглянуто також у роботах [22-24]. Всі ці роботи пропонують наближені алгоритми для розв'язування цієї задачі.

Як видно з [20], задача про складання кількох рюкзаків також може мати обмеження на призначення речей до рюкзаків. В задачі про планування Спринту також присутні елементи задачі про призначення, оскільки основною метою є складання плану Спринту з призначенням виконання завдань учасникам команди. Тому корисним буде розглянути деякі роботи, присвячені задачі про призначення та узагальненій задачі про призначення.

1.3.2 Узагальнена задача про призначення

Задача про призначення формується наступним чином. Нехай потрібно розподілити n робіт (операцій) між n робітниками таким чином, щоб мінімізувати сумарні витрати на виконання комплексу робіт. При цьому кожна робота може бути закріплена лише за одним виконавцем, а кожен робітник може виконувати тільки одну роботу.

Якщо кількість виконавців та робіт однакова, то задача є класичною задачею про призначення. Для такої задачі існують поліноміальні алгоритми її розв'язування. Прикладом такого алгоритму є угорський метод [25].

Проте для задачі планування Спринту задача про призначення не є актуальною, оскільки на практиці кількість учасників команди та завдань майже ніколи не є однаковою. Значно більше схожу ситуацію представляє узагальнена задача про призначення (ще відома, як задача про призначення у відкритій формі) [26].

Задачі про призначення у відкритій формі виникають тоді, коли кількість робітників не дорівнює кількості робіт. У цих випадках задача може бути перетворена в задачу, сформульовану в стандартній формі. Слід особливо зазначити, що задача про призначення є приватним випадком транспортної задачі, у якій кількість пунктів виробництва збігається з кількістю пунктів споживання, а всі величини попиту й величини пропозиції рівні [26].

Узагальнену задачу про призначення можна розглянути у термінах задач про пакування рюкзаків, що і зроблено у [9]. З точних алгоритмів найпоширенішим є реалізація методу гілок та меж з пошуком вглиб [9].

Було розроблено досить багато наближених алгоритмів для даної задачі. Одним із них є поліноміальний алгоритм Martello та Toth. В ньому для кожної з робіт задано, наскільки бажаним є призначення цієї роботи кожному з виконавців. На першому етапі алгоритму переглядаються всі непризначені роботи та для тої роботи, в якій найбільша різниця між найбільшою та другою за розміром бажаністю, визначається виконавець, бажаність призначення якому є максимальною. Після першого етапу отриманий розв'язок покращується локальними обмінами. В результаті роботи даного алгоритму можна отримати деякий припустимий розв'язок [9].

Поширеними є генетичні алгоритми для розв'язування узагальненої задачі про призначення. У [27] запропоновано один із генетичних алгоритмів, де оператором кросоверу є кросовер із спільним елементом (Common Element Crossover), а мутації – In-Pool Mutation. Common Element Crossover полягає в тому, що при кросовері батьків всі елементи, які стоять на однакових місцях у батьків, лишаються на тих самих місцях і в нащадків, а всі інші елементи обмінюються. In-Pool Mutation полягає у заміні одного з елементів на інший, який не порушує припустимості поточного розв'язку [27].

Дані евристичні підходи примушують розроблений генетичний алгоритм сходитися швидше, оскільки простір розв'язків звужується як з Common Element Crossover, так і з In-Pool Mutation. Має сенс розробка інших способів проводити дані операції в генетичних алгоритмах.

У [28] наведено гібридний алгоритм для розв'язування узагальненої задачі про призначення, який поєднує в собі табу-пошук та реалізацію методу гілок та меж. Цей працює дозволяє отримувати досить точні результати (були порівняні з результатами методу гілок та меж у [28]) та працювати припустимий час. В процесі розв'язування за допомогою табу-пошуку генеруються підзадачі, які після цього

розв'язуються. Автори порівняли результати цього алгоритму з результатами стандартного алгоритму гілок та меж та деяких інших евристичних алгоритмів [28].

Ще одним поширеним типом алгоритмів, які застосовуються для розв'язування узагальненої задачі про призначення, є табу-пошук (або пошук із заборонами). Деякі реалізації табу-пошуку для цієї задачі описано у [29-30].

У [31] узагальнену задачу про призначення пропонується розв'язувати за допомогою жадібної рандомізованої процедури адаптивного пошуку GRASP [32] та системи мурах MAX-MIN [33]. В цій роботі показано, що комбінація наведених евристик для узагальненої задачі про призначення дає кращі результати, ніж застосування їх окремо. Але це наближені методи, тому є сенс розробки інших алгоритмів для цієї задачі.

Ще однією роботою, присвяченою узагальненій задачі про призначення, є [34]. В ній пропонується алгоритм табу-пошуку, для якого в якості околу сусідніх розв'язків запропоновано ланцюг викидань (ejection chain). Основна ідея побудови цього околу полягає в тому, що деяке з завдань може бути вилучене з існуючого розв'язку та вважатися вільним. При цьому сам розв'язок стає неповним. Після цього одне з завдань в цьому розв'язку переноситься до виконавця, в якого вилучили завдання перед цим, і, нарешті, вільне завдання призначається одному з виконавців. Якщо вийде так, що два виконавці обмінялися завданнями, то це вважається зсувом. Автори показали, що цей алгоритм табу-пошуку працює дуже добре на задачах маленьких та середніх розмірностей та дозволяє отримати ефективні розв'язки для задач дуже великої розмірності [34]. Але це також наближений алгоритм, тому має сенс розробка інших алгоритмів для цієї задачі.

У роботах [35-38] запропоновано інші наближені алгоритми розв'язування узагальненої задачі про призначення.

Як видно з усіх наведених вище відомостей про узагальнену задачу про призначення, її розв'язують здебільшого за допомогою деяких наближених алгоритмів. Тому має сенс розробка нових алгоритмів для її розв'язування і в подальшому.

1.3.3 Задача складання розкладів для паралельних пристроїв з різною непропорційною продуктивністю та наявністю директивного терміну

Існує одна особливість, якої немає в жодній із розглянутих вище публікацій по плануванню роботи на Спринт або ж задач, елементи яких виявлено у цій задачі. Всі вони не розглядають можливості планування не тільки того, які саме завдання будуть виконані, а й порядку, в якому вони будуть приблизно виконані учасниками команди. Якщо між завданнями існує відношення часткового впорядкування (що на практиці буває часто), то виникає необхідність врахування порядку виконання завдань. Таким чином, задачу оптимального планування роботи на фіксований проміжок часу можна також розглядати як задачу складання розкладів.

В термінах теорії розкладів аналогами учасників команди є паралельні пристрої, які виконують завдання, що є роботами. З визначення задачі маємо різну непропорційну продуктивність для всіх пристроїв та наявність директивного терміну, однакового для всіх учасників.

Задачі планування на паралельних машинах з різною непропорційною продуктивністю розглянуто у [39]. У [39] показано, що лише один вид цих задач є розв'язуваним за поліноміальний час шляхом зведення до лінійної задачі про призначення.

Задачі про оптимальне планування на пристроях з різною продуктивністю та відношенням часткового впорядкування розглянуто у [40]. У публікації розглянуто два види часткового впорядкування, які названо І-впорядкування та АБО-впорядкування. Вони визначають відповідно завдання, які не можна розпочати до закінчення всіх їх попередників, та завдання, які не можна розпочати до закінчення хоча б одного з попередників. У випадку нашої задачі це не грає ролі, оскільки у кожного з завдань може бути тільки один попередник. У [40] показано, що задача планування робіт з відношенням часткового передування на кількох пристроях рівної продуктивності є NP-повною. Очевидно, що при розгляданні пристроїв різної продуктивності також отримаємо NP-повну задачу.

У [41] розглядається задача планування незалежних робіт з призначенням однакового директивного терміну на паралельних пристроях. В якості цільової функції у [41] виступає сумарні втрати, які залежать від обраного директивного терміну та від запізнення робіт, які не вкладаються в даний термін. В даній роботі доведено, що сформульована задача є NP-складною при наявності лише двох пристроїв. Якщо ж кількість пристроїв є довільною, то задача стає NP-складною у сильному сенсі [41].

У [42] розглядається задача, що має дуже багато спільного із задачею оптимального планування роботи за умови різної продуктивності пристроїв. Основні особливості задачі з [42] наступні:

- між завданнями, які необхідно виконати, визначено відношення часткового впорядкування;
- всі пристрої можуть виконувати будь-які завдання;
- пристрої мають різні продуктивності, проте ці продуктивності не залежать від конкретного завдання;
- не всі завдання доступні на початок планування;
- для кожного завдання визначено директивний термін;
- можливий час початку виконання завдань залежить від пристрою та існуючої послідовності завдань на цій машині.

У [42] запропоновано гібридний метаевристичний алгоритм для сформульованої задачі. Він є поєднанням генетичного алгоритму та алгоритму імітаційного відпалу. Спочатку працює генетичний алгоритм, який знаходить деякий глобально непоганий розв'язок. Після цього відбувається локальне покращення отриманого розв'язку за допомогою алгоритму імітаційного відпалу [42].

Для забезпечення припустимості розв'язків задача у [42] розглядається як послідовність двох етапів розв'язання, спочатку складання розкладу завдань, а після цього вже призначення завдань конкретним пристроям. Початкова популяція генерується за допомогою випадкового призначення завдань пристроям.

Проте, як і зазначено у [42], отримані в результаті запропонованого алгоритму розв'язки не є оптимальними та їх точність варіюється в залежності від розмірності та вигляду вхідних даних. Також у розглянутій задачі багато специфічних обмежень, і деякі з них не застосовні до задачі, що розглядається у даній дисертації. Ще однією відмінністю від нашої задачі є те, що продуктивність пристроїв не залежить від того, яке саме завдання додається до розкладу конкретного пристрою.

У [43] також розглядається генетичний алгоритм для розв'язування схожої задачі, проте у [43] пристрої є пропорційними. Особливостями розглянутої задачі є наявність різних директивних термінів для завдань, незалежність завдань одне від одного та різні моменти початку виконання для кожного з завдань. У [43] запропоновано генетичний алгоритм, який знаходить деякий близький до оптимального розв'язок.

З результатів експериментів у [43] очевидно, що дана евристика не дає оптимального розв'язку та зі збільшенням розмірності задачі займає дедалі більше часу та відхиляється від найкращого розв'язку.

У [44] запропоновано узагальнений підхід до розв'язування задач планування на паралельних пристроях з різною продуктивністю. Представлений у [44] алгоритм протестовано з найпоширенішими видами даної задачі, які включають мінімізацію сумарного зваженого часу закінчення всіх робіт, мінімізацію середньої завантаженості пристроїв та багатокритеріальних варіантів задачі.

Особливістю задачі, що розглядається у [44], є те, що завдання можуть розподілятися між кількома пристроями. На цьому і базується запропонований у [44] алгоритм. Він починає свою роботу з деякого вже існуючого розподілу завдань між пристроями, а після цього намагається випадковим чином перерозподілити частини завдань між різними іншими пристроями, по максимуму зберігаючи поточні значення завантаженості кожного з пристроїв [44].

Таким чином, запропонований у [44] алгоритм може бути використаний для задачі про планування роботи на ітерацію за умов, що завдання можуть розподілятися між учасниками команди. Така ситуація трапляється на практиці, але

на поточний момент приділимо увагу дискретній версії сформульованої задачі, де кожне завдання буде виконуватися лише одним учасником команди.

У [45] розглядається задача планування робіт на паралельних пристроях з різною продуктивністю та наявністю передування між роботами. Цікавим у [45] є те, що розглядається таке передування, яке представляється у вигляді лісу, тобто набору дерев передування, а не тільки ланцюгів. Це загальний випадок, а у випадку задачі про планування роботи на ітерацію між завданнями визначено відношення часткового передування, яке не є деревом, а скоріш набором ланцюгів [45]. В подальшому можливе розширення цієї задачі і тоді матеріали [45] можуть бути використані.

У [46] розглядається задача складання розкладу для лише одного пристрою з умовою, що завдання є частково впорядкованими. Метою задачі є мінімізація сумарного зваженого часу завершення виконання завдань. В цій роботі запропонований алгоритм знаходження наближеного розв'язку поставленої задачі [46].

Ще одна робота, яка розглядає подібну задачу, - це [47]. В ній досліджено алгоритм, запропонований у [48], та запропоновано його модифікацію, яка працює краще, ніж оригінальний алгоритм. Також задачу у цій роботі розглянуто в термінології теорії графів.

У [49] також розглядається задача складання розкладу для одного пристрою за наявності обмежень часткової впорядкованості завдань. Робота [49] заснована на результатах, описаних у [50]. Також у [49] виявлено зв'язок між задачею вершинного покриття та задачею складання розкладу для одного пристрою.

Ще один алгоритм розв'язування задачі про планування на паралельних машинах з різною продуктивністю запропоновано у [51]. Він складається з двох етапів. Перший з них полягає у знаходженні за допомогою бінарного пошуку найбільшої кількості завдань, які можуть бути призначені повністю одному пристроєві. Другий полягає у знаходженні розподілу завдань, які не було розподілено, за допомогою пошуку найкращого паросполучення у графі, побудованого на пристроях та завданнях, що лишилися [51].

Для задачі про планування роботи на ітерацію цей підхід не є застосовним у повній мірі, оскільки не враховує важливості робіт ніяким чином. Звичайно, можна модифікувати підхід з [51] та використати для розв'язування сформульованої задачі.

У роботі [52] розглянуто спосіб розв'язування задачі складання розкладу виконання завдань пристроями з різною продуктивністю за умови наявності спільного директивного терміну, застосовуючи нейронну мережу. В якості цільової функції виступає сумарна вартість, яка є сумою випередження та запізнення для виконання завдань. Мережа навчається на деяких наборах вхідних даних, де вчиться визначати вагові коефіцієнти для випередження та запізнення, а після цього її використовують для розв'язування інших екземплярів задачі [52].

Робота [53] описує алгоритм локального пошуку для задачі складання розкладів для пристроїв з різною продуктивністю, де цільова функція є максимальним часом закінчення роботи пристроїв, а метою є її мінімізація. При генерації наступного розв'язку з околу враховується ефективність кожного пристрою для завдання, яке розглядається [53].

У [54] також розглядається задача складання розкладів для пристроїв з різною продуктивністю з метою мінімізації часу завершення, проте у [54] пропонується застосування променевого пошуку з відновленням (recovering beam search). Цей алгоритм вперше запропоновано у [55]. У [54] показано, що запропонований алгоритм є достатньо ефективним на задачах великої розмірності.

Більше варіацій задачі про планування робіт на пристроях з різною продуктивністю можна знайти у роботах [56-65]. Необхідно звернути увагу, що більшість з них не мають такої складової, як важливість завдань, або ж часткова впорядкованість завдань, тому в повній мірі алгоритми, запропоновані у [56-65] не можуть бути використані для розв'язування задачі про планування роботи на ітерацію.

Висновок до розділу

Задача оптимального планування роботи на ітерацію (як-то Спринт у Скрам) є дуже важливою з практичної точки зору, оскільки наразі ітеративні підходи до виконання проектів набувають все більшої популярності. До того ж витрати часу на планування роботи можуть бути досить значущими через те, що кожен учасник команди витрачає рівну кількість свого часу на планування.

Для задачі планування роботи на Спринт за умови різної продуктивності учасників команди запропоновано низку алгоритмів та використано різні інструменти. Проте точний метод гілок та меж займає надто багато часу при великій розмірності задачі, а наближені алгоритми, які було запропоновано, все ще можна покращити.

Задача планування роботи на ітерацію містить в собі елементи задачі про пакування кількох рюкзаків, узагальненої задачі про призначення та задач складання розкладів. Всі ці типи задач були докладно розглянуті та для них було запропоновано велику кількість різноманітних алгоритмів. Задачу планування роботи на ітерацію за умови різної продуктивності пристроїв також ще не було докладно розглянуто в контексті задач складання розкладів. Схожі за умовами задачі складання розкладів є NP-повними, тобто розробка нових наближених алгоритмів розв'язування задачі, що розглядається, є актуальною.

2 ПОСТАНОВКА ТА ФОРМАЛІЗАЦІЯ ЗАДАЧІ

2.1 Системний аналіз проблематики

Задача планування Спринту сама по собі є задачею планування проекту. На сьогоднішній день планування виконання проекту є задачею, яка постає в кожній індустрії. Цю задачу можна природнім чином декомпонувати на декілька етапів. Це розбиття наведено на рисунку 2.1.



Рисунок 2.1 – Складові планування проекту

До початку планування виконання завдань необхідно виконати попередні етапи, розбиття на завдання та підбір команди виконавців. Ми розглядатимемо саме етап планування виконання існуючих завдань вже відібраною командою.

Те, як саме буде виконане розподілення завдань між виконавцями, залежить від специфіки проекту. Наведемо критерії, за якими процес розподілу та планування завдань різняться:

- наявність директивного терміну для всіх завдань, для кожного з завдань або кожного з виконавців;
- наявність часу початку роботи для завдань чи виконавців;
- наявність впорядкованості між завданнями;
- можливість розбиття задач на підзадачі;
- можливість виконання підзадач різними виконавцями;
- можливість виконання завдань виконавцями: всіма чи лише деякими та ін.

Всі ці критерії впливають як на розподіл, так і на часові рамки, в яких будуть виконані завдання. В цій роботі ми розглядатимемо випадок, при якому завдання не можуть бути розбиті на підзадачі, всі завдання можуть бути виконаними будь-яким виконавцем, завдання є частково впорядкованими та всі завдання мають однаковий директивний термін.

2.2 Змістовна постановка задачі

Для автоматизації процесу планування роботи команди протягом обмеженого проміжку часу необхідно формалізувати цю задачу. Наведемо її змістовну постановку.

Нехай $N = \{1, \dots, n\}$. Маємо команду, яка складається з m виконавців, і вона працює над проектом, для якого задано n завдань. Кожне з цих завдань може виконуватися будь-яким виконавцем, але тільки одним. Також завдання є частково упорядкованими, тобто є такі завдання $j, j \in N$, для яких задане деяке завдання $k, k \in N \setminus \{j\}$, яке має бути виконане до них. Кожне завдання може передувати або одному завданню, або жодному. Для кожного виконавця заданий час виконання кожного з завдань даним учасником $t_{ij}, i = \overline{1, m}, j = \overline{1, n}$. Для кожного завдання визначена його важливість $v_j, j = \overline{1, n}$.

Серед цих завдань є q обов'язкових до виконання завдань $Q = \{r, r \in N\}, |Q| = q$. Інші завдання не обов'язкові до виконання. Для визначеності без втрати загальності вважатимемо, що в списку завдань ці обов'язкові завдання є першими q завданнями. В загальному випадку обов'язкові завдання можуть бути впорядковані, але заради простоти викладу розглянемо випадок, в якому вони не є такими, тобто для них не визначено такі завдання k , що їм передують або мають бути виконані після їх закінчення. Отримані для такої ситуації результати можна буде без ускладнень розвинути на загальний випадок.

Заданий також директивний термін виконання підмножини завдань d .

Необхідно вибрати для виконання виконавцями такий набір завдань, що обов'язково включає всі завдання з Q та щоб сумарна важливість цього набору завдань була максимальною, та скласти розклад виконання цих задач виконавцями.

2.3 Математична модель задачі

Задача полягає у максимізації спеціальної цільової функції, а саме

$$\sum_{i=1}^m \sum_{j=1}^n x_{ij} v_j \rightarrow \max, \quad (2.1)$$

яка задає сумарну важливість усіх завдань, що будуть виконані, де

$$x_{ij} = \begin{cases} 1, & \text{якщо учасник } i \text{ виконує завдання } j, \\ 0, & \text{інакше,} \end{cases} \quad (2.2)$$

де $i = \overline{1, m}, j = \overline{1, n}$.

Також позначимо s_j час початку виконання завдання $j, j \in N$, причому:

$$s_j = \begin{cases} = 0, & \text{якщо завдання } j \text{ не буде виконане,} \\ > 0, & \text{якщо завдання } j \text{ буде виконане.} \end{cases} \quad (2.3)$$

Обмеження задачі:

$$0 \leq \sum_{i=1}^m x_{ij} \leq 1, \quad (2.4)$$

$$\sum_{i=1}^m x_{ij} = 1, j \in Q. \quad (2.5)$$

Для всіх $j \in N, k \in N$ таких, що $x_{ij} = x_{ik}$ для всіх $i = \overline{1, m}$ та $\sum_{i=1}^m x_{ij} = 1, \sum_{i=1}^m x_{ik} = 1$:

$$s_j < s_k \rightarrow s_j + t_{ij} < s_k. \quad (2.6)$$

Для всіх завдань $j, j \in N$ таких, що $s_j > 0$:

$$s_j + \sum_{i=1}^m x_{ij} t_{ij} \leq d. \quad (2.7)$$

Для всіх $j, j \in N \setminus Q$, для яких $\sum_{i=1}^m x_{ij} = 1$ та є завдання k , що їм передують:

$$s_k < s_j, s_k + \sum_{i=1}^m x_{ik} t_{ij} < s_j. \quad (2.8)$$

Обмеження (2.4) задає можливість виконання кожного з завдань лише одним виконавцем. (2.5) задає обов'язкове виконання всіх завдань $j, j \in Q$. (2.6) – неперетинання запланованих робіт для одного виконавця. Обмеження (2.7) задає виконання всіх завдань в розкладі до настання директивного терміну, а (2.8) – виконання завдань відповідно до відношення часткового строгого впорядкування.

2.4 Декомпозиційний підхід

Розглянемо поставлену задачу докладніше.

Зрозуміло, що нижня межа задачі дорівнює сумарній важливості всіх обов'язкових завдань

$$\sum_{j \in Q} v_j. \quad (2.9)$$

За умовами задачі всі обов'язкові завдання будуть включені в будь-який припустимий розв'язок. Тобто в отриманому розкладі матимемо всі обов'язкові завдання та деяку кількість необов'язкових. З такого розділення можна зробити припущення, що проблема природнім чином декомпонується на дві підзадачі, які розглянемо далі.

Перша підзадача – розподілити виконання обов'язкових завдань між виконавцями, друга – доповнити складений в результаті розв'язання першої підзадачі розподіл деякою кількістю необов'язкових завдань та скласти розклад. Якщо у другій підзадачі вибір цільової функції очевидний (максимізація сумарної важливості всіх завдань в розкладі), то з першою підзадачею ситуація трохи інша.

Сумарна важливість обов'язкових завдань не змінюється через різний розклад, тому необхідно розглянути зв'язок між підзадачами.

Очевидно, що необхідно так розподілити обов'язкові завдання між виконавцями, щоб отримати якомога більше часу на виконання необов'язкових завдань. Існує два можливих способи для досягнення даної мети. Для їх викладення наведемо означення випередження:

Означення 2.1. Випередженням для виконавця $i, i \in \{1, 2, \dots, m\}$ є час, який залишається до настання директивного терміну після закінчення виконання останнього завдання в розкладі даного виконавця:

$$E_i = d - \max \{s_j + t_{ij}\}. \quad (2.10)$$

Згадані вище способи отримання часу на виконання необов'язкових завдань є такими:

1) Мінімізація сумарного часу, що буде витрачений на виконання обов'язкових завдань:

$$\sum_{i=1}^m \sum_{j=1}^q x_{ij} t_{ij} \rightarrow \min. \quad (2.11)$$

2) Максимізація мінімального випередження серед усіх виконавців:

$$\max \{E_i\} \rightarrow \min, i \in \{1, 2, \dots, m\}. \quad (2.12)$$

На перший погляд другий спосіб (2.3) здається вигіднішим, оскільки завдання більшого розміру можна буде розмістити в отриманому проміжку до настання директивного терміну. Проте розглянемо цю задачу з практичної точки зору.

У Скрамі завдання, які беруться до розгляду, мають відповідати певним критеріям. Одним із них є те, що вони мають бути приблизно одного розміру [66, 67]. На практиці завдання намагаються розбивати таким чином, щоб вони вимагали найменшого можливого часу виконання, оскільки тоді їх простіше виконати, перевірити та вважати закінченими. Також необхідно зазначити, що ще однією вимогою до завдання є його цінність для проекту в цілому, тобто його виконання

має просувати проект до успішного завершення. Тому завдання мають деякий середній час, який необхідний для їх виконання [66]. У [66] розглядаються основні вимоги до завдань у Скрамі та те, як формулювання завдань та необхідний на їх виконання час залежать від команди.

Здебільшого розкид часу на виконання завдань є невеликим. У [67] проведено опитування людей, які працюють за Скрамом, в тому числі і на рахунок розмірності завдань. В цьому опитуванні брали участь люди з різних команд, які працювали над різними проектами. В рамках дослідження розглянемо результати опитування про оптимальну кількість завдань на Спринт та найбільше завдання на Спринт. З результатів опитування видно, що в середньому на Спринт береться від 4 до 13 завдань, що показано на рисунку 2.2.

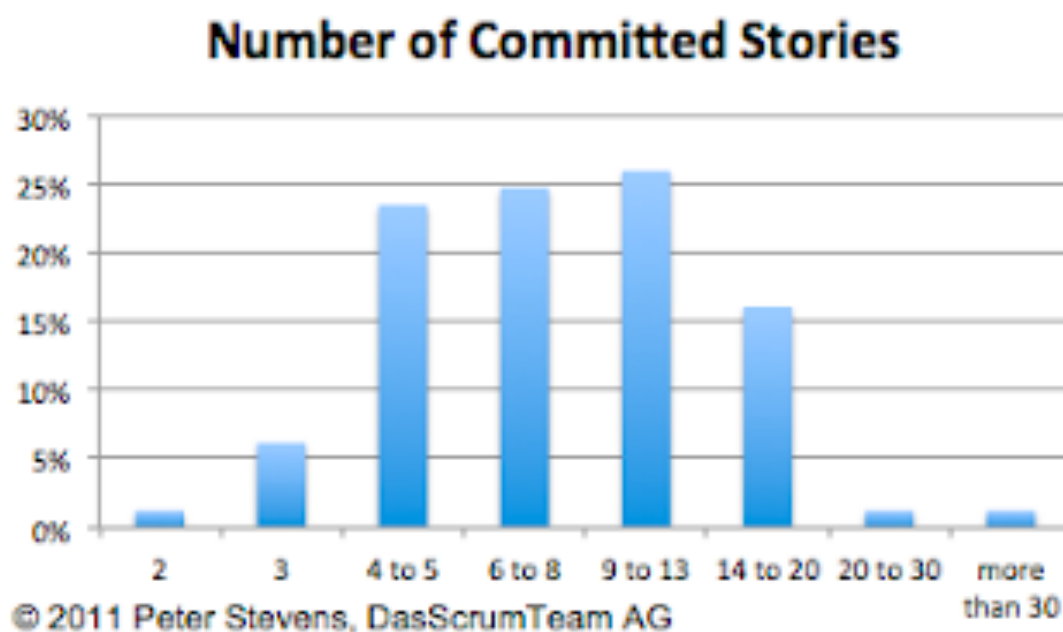


Рисунок 2.2 – Загальна кількість завдань на Спринт

Вісь X на рисунку 2.1 показує кількість завдань, а вісь Y – відсоток опитуваних, що схилиються до даної кількості завдань. Що стосується найбільшого завдання, то необхідно звернути увагу, що зазвичай такі завдання займають 10-20% часу від всього Спринту. Результати опитування наведено на рисунку 2.3.

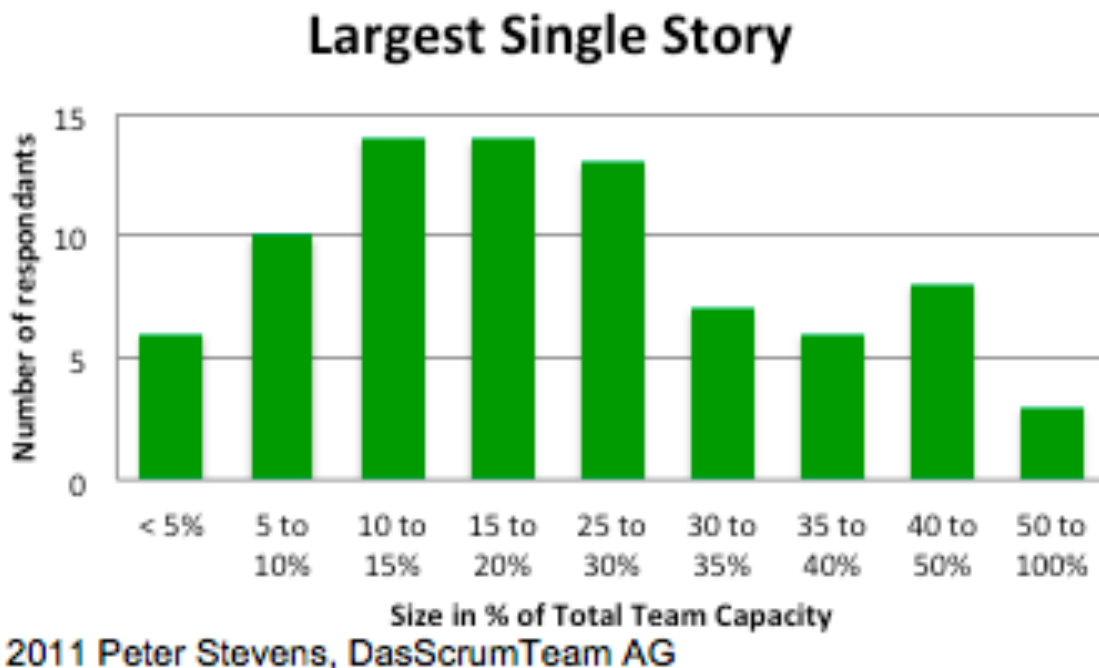


Рисунок 2.3 – Середній час на виконання найбільшого завдання на Спринт за результатами опитування

Вісь X показує обсяг загального часу, який виділяється на найбільше завдання, а вісь Y – кількість опитуваних, що погоджуються з таким обсягом. Як видно, найбільша кількість опитуваних навели обсяг часу на виконання завдання в середньому 5-30% від загального часу, який має команда виконавців. З цього можна зробити висновок, що, так як розкид часу, необхідного на виконання завдань, не є великим, то всі завдання приблизно однакові. Розглянемо цю ситуацію докладніше.

Нехай існує деякий розклад всіх обов'язкових завдань $T = \{(i, s_j), i \in \{1, 2, \dots, m\}, j \in Q\}$. В цьому розкладі є деяке завдання $j', j' \in Q$ у розкладі виконавця $i', i' \in \{1, 2, \dots, m\}$, яке можна помістити в кінець розкладу іншого виконавця. Тобто є такий виконавець $i'', i'' \in \{1, 2, \dots, m\} \setminus \{i'\}$, що $E_{i''} \geq t_{i'j'}$.

Але з практичної інформації зрозуміло, що через невеликий розкид часу на завдання, які необхідно виконати, на дане місце у розкладі виконавця i'' з великою імовірністю можна буде додати і деяке необов'язкове завдання $j'', j'' \in N \setminus Q$.

Тому з практичної точки зору обидва способи визначення витрат часу на виконання обов'язкових завдань рівноцінні.

Якщо обрати цільовою функцією (2.11), отримуємо наступну математичну модель для першої підзадачі.

Математична модель підзадачі 1 (планування обов'язкових завдань)

Задача полягає у мінімізації спеціальної цільової функції, а саме

$$\sum_{i=1}^m \sum_{j=1}^q x_{ij} t_{ij} \rightarrow \min, \quad (2.13)$$

де цільова функція задає сумарний час, витрачений на виконання всіх обов'язкових завдань, де:

$$x_{ij} = \begin{cases} 1, & \text{якщо учасник } i \text{ виконує завдання } j, \\ 0, & \text{інакше,} \end{cases} \quad (2.14)$$

де $i = \overline{1, m}, j = \overline{1, n}$.

Обмеження задачі:

$$\sum_{i=1}^m x_{ij} = 1, j \in Q. \quad (2.15)$$

Для всіх виконавців $ij \in \{1, 2, \dots, m\}$:

$$\sum_{j=1}^q x_{ij} t_{ij} \leq d. \quad (2.16)$$

Обмеження (2.15) задає виконання кожного з обов'язкових завдань одним виконавцем. (2.16) задає виконання всіх обов'язкових завдань в розкладі кожного з виконавців до настання директивного терміну.

Сформульована математична модель описує узагальнену задачу про призначення. Ця класична задача комбінаторної оптимізації є NP-складною [68], для неї вже було розроблено велику кількість різноманітних алгоритмів, як точних, так і наближених.

Сформулюємо математичну модель підзадачі 2. Для неї ми матимемо додаткові вхідні дані після розв'язування підзадачі 1 – розподілені між виконавцями обов'язкові завдання з Q . Позначимо їх як $b_j, j \in Q$. Очевидно, що $b_j \in \{1, 2, \dots, m\}$.

Математична модель підзадачі 2 (планування необов'язкових завдань)

Задача полягає у максимізації спеціалізованої цільової функції, а саме

$$\sum_{i=1}^m \sum_{j=1}^n x_{ij} v_j \rightarrow \max, \quad (2.17)$$

яка задає сумарну важливість усіх завдань, що будуть виконані, де

$$x_{ij} = \begin{cases} 1, & \text{якщо учасник } i \text{ виконує завдання } j, \\ 0, & \text{інакше,} \end{cases} \quad (2.18)$$

де $i = \overline{1, m}, j = \overline{1, n}$.

Також позначимо s_j час початку виконання завдання $j, j \in N$, причому:

$$s_j = \begin{cases} = 0, & \text{якщо завдання } j \text{ не буде виконане,} \\ > 0, & \text{якщо завдання } j \text{ буде виконане.} \end{cases} \quad (2.19)$$

Обмеження задачі:

$$0 \leq \sum_{i=1}^m x_{ij} \leq 1, \quad (2.20)$$

$$x_{b_j j} = 1, j \in Q. \quad (2.21)$$

Для всіх $j \in N, k \in N$ таких, що $x_{ij} = x_{ik}$ для всіх $i = \overline{1, m}$ та $\sum_{i=1}^m x_{ij} = 1, \sum_{i=1}^m x_{ik} = 1$:

$$s_j < s_k \rightarrow s_j + t_{ij} < s_k. \quad (2.22)$$

Для всіх завдань $j, j \in N$ таких, що $s_j > 0$:

$$s_j + \sum_{i=1}^m x_{ij} t_{ij} \leq d. \quad (2.23)$$

Для всіх $j, j \in N$, для яких $\sum_{i=1}^m x_{ij} = 1$ та є завдання k , що їм передують:

$$s_k < s_j, s_k + \sum_{i=1}^m x_{ik} t_{ij} < s_j. \quad (2.24)$$

Обмеження (2.20) задає можливість виконання кожного з завдань лише одним виконавцем. (2.21) задає виконання обов'язкових завдань саме тими виконавцями, яких було визначено в результаті розв'язання підзадачі 1. (2.22) – неперетинання запланованих робіт для одного виконавця. Обмеження (2.23) задає виконання всіх завдань в розкладі до настання директивного терміну, а (2.24) – виконання завдань відповідно до відношення часткового строгого впорядкування.

У [40] було показано, що задача планування робіт з відношенням часткового передуювання на кількох пристроях рівної продуктивності є NP-повною. Очевидно, що при розгляданні пристроїв різної продуктивності та з існуючим частковим фіксованим розподілом завдань також отримаємо NP-повну задачу.

Таким чином, після виокремлення двох підзадач, як наведено вище, отримуємо такий алгоритм розв'язування задачі (2.1) – (2.8):

Крок 1. Розв'язати задачу (2.13) – (2.16).

Крок 2. Використовуючи отриманий на кроці 1 розподіл обов'язкових завдань між виконавцями, розв'язати задачу (2.17) – (2.24).

Висновок до розділу

Показано місце досліджень та проблематики у процесі планування проектів в загальному. Сформульовано змістовну постановку конкретної проблеми, яка розглядається, та для неї наведено математичну модель. Запропоновано алгоритм розв'язування сформульованої задачі, що розглядається, шляхом розбиття її на дві

підзадачі та розв'язування їх послідовно. Для обох підзадач сформульовано математичні моделі.

Одна з підзадач є класичною узагальненою задачею про призначення, що дозволяє використати існуючі алгоритми для її розв'язування. Ця задача є NP-складною, тому для неї існують як наближені, так і точні алгоритми розв'язування.

Друга підзадача використовує результати розв'язування попередньої та є різновидом задачі складання розкладів із спеціальними обмеженнями, до яких входять часткова впорядкованість завдань, різний час виконання завдань для різних виконавців, наявність обов'язкових та необов'язкових завдань тощо. Ця підзадача також є NP-складною, тому для неї має сенс розробка наближених алгоритмів, які дозволять знайти деякий розв'язок за прийнятний час.

3 АЛГОРИТМИ РОЗВ'ЯЗУВАННЯ ЗАДАЧІ

3.1 Алгоритм розв'язування підзадачі про розподіл обов'язкових завдань

Підзадача розподілу обов'язкових завдань між виконавцями є класичною узагальненою задачею про призначення (GAP). Однією з робіт, присвячених цій задачі та її різновидам, є [69].

Для розв'язування першої підзадачі використано алгоритм знаходження наближеного розв'язку узагальненої задачі про призначення з [70]. Він полягає у тому, що для кожного з виконавців по черзі призначаються завдання за допомогою деякого алгоритму складання рюкзака. Алгоритм, запропонований авторами [70], також застосовний у випадку, коли не тільки час виконання, а й важливість завдань залежить від того, який саме виконавець виконуватиме це завдання. Це обумовлене перерахунком важливості кожного завдання при переході до кожного наступного виконавця за формулою [70]

$$P_j[i] = \begin{cases} v_j, & \text{якщо завдання } j \text{ ще не призначене виконавцю,} \\ 0 & \text{інакше.} \end{cases} \quad (3.1)$$

Нехай є деякий алгоритм A для розв'язування задачі про пакування рюкзака [9]. Наведемо опис алгоритму обчислення наближеного розв'язку узагальненої задачі про призначення з [70].

Крок 1. Вважати всі завдання нерозподіленими.

Крок 2. Для кожного виконавця $i = \overline{1, m}$,

2.1. Обчислити важливість завдань $j \in Q$ за формулою (3.1).

2.2. Запустити алгоритм A для розкладу завдань виконавця i , використовуючи P_j з кроку 2.1 в якості вектору важливості завдань.

2.3. Отримавши набір завдань S_j' з 2.2, всі завдання $j', j' \in S_j'$ призначити виконавцю i [70].

В якості алгоритму складання рюкзака A використано алгоритм динамічного програмування [71].

3.2 Алгоритми розв'язування підзадачі планування необов'язкових завдань

3.2.1 Жадібний алгоритм знаходження початкового припустимого розв'язку

Пропонується жадібний алгоритм для знаходження припустимого розв'язку сформульованої підзадачі 2. Перевагами жадібних алгоритмів є невеликий час роботи та простота реалізації, отже даний клас алгоритмів є зручним для початку дослідження нових задач та їх розв'язування перед розробкою складніших алгоритмів.

Але для нашої задачі послідовність локально оптимальних (жадібних) виборів не дає в результаті глобального оптимального розв'язку [72]. Тому наведений далі алгоритм може бути використаний для знаходження деякого припустимого розв'язку, який не обов'язково буде оптимальним.

Для початку наведемо алгоритм сортування списку завдань таким чином, що у відсортованому списку завдань номер кожного завдання $k_j \in N$, $j \in N$ є більшим, ніж номер завдання, яке йому передує, тобто $r_j > r_k$. Такий алгоритм є модифікацією алгоритму сортування включенням [72], в якому замість порівняння значень елементів, що сортуються, перевіряється відношення передування.

Позначимо $P = N \setminus Q$, $|P| = p$, $p = n - q$.

Алгоритм сортування завдань у порядку відношення передування

Крок 1. Виділити усі підмножини завдань, які є у відношенні передування.

Крок 2. Відсортувати кожну з цих підмножин алгоритмом сортування включенням.

Крок 3. Поєднати отримані відсортовані ланцюги завдань в один.

Запропонований алгоритм буде використаний у наведеному далі жадібному алгоритмі для того, щоб задовольнити обмеження на передування завдань.

Жадібний алгоритм для задачі про планування роботи команди на ітерації

Крок 1. Помістити всі обов'язкові завдання в початок розкладу відповідного виконавця, впорядкувавши їх за порядковим номером.

Крок 2. Впорядкувати всі завдання з $j, j = \overline{1, p}$ за алгоритмом сортування завдань у порядку відношення передування.

Крок 3. Для кожного завдання $j, j = \overline{1, p}$ із впорядкованої на кроці 2 послідовності:

3.1 ЯКЩО для поточного завдання j не визначено завдання k , яке йому передує,

ТО для всіх виконавців $i = \overline{1, m}$:

ЯКЩО в розкладі даного виконавця ще немає завдань та $t_{ij} < d$ АБО час завершення поточного завдання j , якщо його розмістити після останнього завдання в розкладі даного виконавця $k \in N$ виконується $s_k + t_{ik} + t_{ij} < d$,

ТО додати поточне завдання до розкладу поточного виконавця i та перейти до наступного завдання $j + 1$.

ІНАКШЕ перейти до наступного виконавця

3.2 ІНАКШЕ ЯКЩО завдання k вже у розкладі одного з виконавців,

ТО для всіх виконавців $i = \overline{1, m}$:

ЯКЩО в розкладі даного виконавця ще немає завдань та

$$s_k + \sum_{i=1}^m x_{ik} t_{ik} + t_{ij} < d$$

ТО додати поточне завдання до розкладу поточного виконавця на час

$$s_j = s_k + \sum_{i=1}^m x_{ik} t_{ik};$$

АБО ЯКЩО час завершення поточного завдання j , якщо його розмістити після останнього завдання в розкладі даного виконавця $r \in N$

$$s_r + t_r + t_{ij} < d \text{ та } s_k + \sum_{i=1}^m x_{ik} t_{ik} < s_r + t_{ir},$$

ТО додати поточне завдання до розкладу поточного виконавця на час

$$s_j = s_r + t_{ir}$$

перейти до наступного завдання $j + 1$

ІНАКШЕ перейти до наступного виконавця

3.3 ІНАКШЕ перейти до наступного завдання $j + 1$

В результаті роботи даного алгоритму буде отримано деякий припустимий розв'язок сформульованої підзадачі 2. Оскільки алгоритм є дуже примітивним, то отриманий розв'язок буде з високою імовірністю далеким від оптимального, але він надає можливість знайти деяку відправну точку для подальшого покращення отриманого розв'язку за допомогою складніших алгоритмів. В наступних пунктах наведено алгоритми детермінованого локального пошуку (ДЛП), імітаційного відпалу (АІВ) та прискореного імовірнісного моделювання (G-алгоритм).

3.2.2 Алгоритм детермінованого локального пошуку

Алгоритми детермінованого локального пошуку - це сім'я ітераційних методів, заснована на частковому перебиранні варіантів на кожній ітерації серед точок околу поточної точки, тобто серед сусідніх до неї [73].

В алгоритмах цього типу замість повного перебору застосовується спрямований локальний перебір у підмножинах варіантів, які називаються околами. Цим пояснюється їх назва - *алгоритми локального пошуку* (Local search). У сфері комбінаторної оптимізації алгоритми локального пошуку мають давню історію через свою наочність і високу ефективність. Наприклад, перший алгоритм локального пошуку для задачі комівояжера був запропонований ще в 1956 р., а локальний пошук для задачі розміщення обладнання був розроблений у 1962 р. Загальна схема локального пошуку у задачах мінімізації така: починаючи з деякого припустимого розв'язку задачі, новий розв'язок із кращим значенням цільової функції шукають у його околі. Якщо такий розв'язок знайдено, то він приймається і пошук поліпшення розв'язку далі здійснюється вже в його околі і т. д. Алгоритм закінчується, коли досягнуто локального оптимуму, тобто коли в околі поточного

розв'язку немає ніякого іншого варіанта з меншим значенням цільової функції.

Загальна схема алгоритмів детермінованого локального пошуку може бути подана таким чином:

1. Генерація початкового припустимого розв'язку x , який обираємо як поточний варіант.

2. Чергова ітерація. Формуємо оточення $O(x)$ поточного варіанта й точно чи наближено знаходимо елемент $y \in O(x)$, який є субоптимальним розв'язком у цьому оточенні. Якщо $y \neq x$, то знайдений елемент оголошується черговим поточним варіантом (здійснюється переприсвоєння $x \leftarrow y$) і починається чергова ітерація, інакше - повернення на п. 3.

3. Завершення роботи алгоритму: x - локальний розв'язок, якщо на останній ітерації здійснюється вичерпний пошук в оточенні.

Таким чином, локальний пошук подібний простому алгоритму спуску з тією відмінністю, що (а) оточення поточного розв'язку досліджуються систематично замість безладного блукання, і (б) пошук в оточенні повторюється до тих пір, поки не буде знайдений локально оптимальний розв'язок. У додатку 1 наведено загальну блок-схему алгоритмів локального пошуку.

Ключовими аспектами реалізації алгоритмів детермінованого пошуку є:

- визначення оточень $O(x)$;
- генерація чергової точки $y \in O(x)$;
- критерій завершення перегляду точок у поточному оточенні та переходу до наступного;
- спосіб обчислення величини зміни цільової функції при переході до нового поточного варіанта;
- критерій завершення;
- формування початкового наближення.

Розглянемо ключові аспекти реалізації для нашого алгоритму.

Формування початкового наближення

В якості початкового наближення виступає розв'язок, отриманий жадібним

алгоритмом, описаним у пункті 3.2.1.

Визначення околів та генерація чергової точки околу

Використаємо кільцевий генератор точок околу, тобто такий, що продовжує свою роботу з попереднього місця замість перегляду всіх точок, які вже були переглянуті. Це дозволяє не переглядати та покращувати постійно одну й ту саму частину розв'язку, а змінювати його цілком.

Вважатимемо два розв'язки сусідніми, якщо вони розрізняються складом завдань на одне завдання, тобто розв'язок 1, який має набір завдань

$$K_1 = \{i \mid i \in N\},$$

та розв'язок 2, який має набір завдань

$$K_2 = K_1 \setminus \{r\} \cup \{j\}, j \in N \setminus K_1$$

є сусідніми.

Пропонується процедура генерації розв'язків з околу, заснована на ідеї зсувів. Основна мотивація використання зсувів полягає у можливості виділити більше часу на виконання необов'язкових завдань, які будуть додані до розв'язку під час його зміни алгоритмами локального пошуку. Цей підхід дозволить додавати завдання, для виконання яких потрібно більше часу.

Для формального опису процедури генерації сусіднього розв'язку введемо означення відстані зсуву вниз та відстані зсуву вверх.

Означення 3.1. Відстанню зсуву вниз d_{down} для деякого завдання $j, j \in N$, яке в поточному розв'язку буде виконане виконавцем $i, i \in [1, \dots, m]$, починаючи у момент часу $s_j > 0$ та за час t_{ij} , є проміжок часу, на який можна посунути ближче до закінчення директивного терміну всі завдання $k \in N \setminus \{j\} = K$, які у розкладі виконавця i будуть виконані після завдання j , тобто $x_{ik} = 1, s_k > s_j + t_{ij}$. Зрозуміло, що цей проміжок часу залежить від того, чи мають завдання $k \in K$ такі завдання, які заплановані після них, та чи мають вони завдання, які можуть бути виконані тільки після них. Позначимо:

- завдання, яке можна виконати тільки після деякого завдання $k \in K$, як $succ(k), succ(k) \in N \setminus \{k\}$;
- завдання, яке заплановано на виконання після завдання $k \in K$, як $after_k, after_k \in N \setminus \{k\}$;
- завдання, яке заплановано на виконання до завдання $k \in K$, як $before_k, before_k \in N \setminus \{k\}$;
- завдання, яке має бути виконано до завдання $k \in K$, як $prev(k), prev(k) \in N \setminus \{k\}$.

Проміжок часу для кожного завдання $k \in K$, окрім останнього у розкладі виконавця i , яке має $succ(k), succ(k) \in N \setminus \{k\}$, обчислюється таким чином:

$$d_{down}(k) = s_{succ(k)} - (s_k + t_{ik}).$$

Для останнього ж завдання $k_{last} \in K$, якщо воно має $succ(k), succ(k) \in N \setminus \{k\}$, цей проміжок часу обчислюється так:

$$d_{down}(k_{last}) = \min\{d - (s_{k_{last}} + t_{ik_{last}}), s_{succ(k_{last})} - (s_{k_{last}} + t_{ik_{last}})\},$$

де d – директивний термін.

Якщо ж завдання $k \in K$ не має $succ(k), succ(k) \in N \setminus \{k\}$, то проміжок часу обчислюється таким чином:

$$d_{down}(k) = d - (s_k + t_{ik}).$$

Тоді загальна відстань зсуву вниз обчислюється наступним чином:

$$d_{down} = \min_{k \in K} \{d_{down}(k)\}. \quad (3.2)$$

Звільнений для нової роботи проміжок часу буде обчислюватись таким чином: якщо є завдання $after_j, after_j \in N \setminus \{j\}$ та завдання $before_j, before_j \in N \setminus \{j\}$, то:

$$gap_{down} = (s_{after_j} + t_{iafter_j}) + d_{down} - s_{before_j}. \quad (3.3)$$

інакше, якщо є тільки завдання $after_j$, $after_j \in N \setminus \{j\}$, то

$$gap_{down} = (s_{after_j} + t_{iafter_j}) + d_{down}. \quad (3.4)$$

інакше, якщо є тільки завдання $before_j$, $before_j \in N \setminus \{j\}$, то

$$gap_{down} = d - s_{before_j}. \quad (3.5)$$

інакше:

$$gap_{down} = d. \quad (3.6)$$

Тоді час початку нового доданого завдання буде обчислюватись так: якщо є завдання $before_j$, $before_j \in N \setminus \{j\}$ та $prev(j)$, $prev(j) \in N \setminus \{j\}$, то:

$$s_k = \max \{s_{before_j} + t_{ibefore_j}, s_{prev(j)} + t_{iprev(j)}\}. \quad (3.7)$$

інакше, якщо є лише $before_j$, $before_j \in N \setminus \{j\}$, то

$$s_k = s_{before_j} + t_{ibefore_j}. \quad (3.8)$$

інакше, якщо є лише $prev(j)$, $prev(j) \in N \setminus \{j\}$, то

$$s_k = s_{prev(j)} + t_{iprev(j)}. \quad (3.9)$$

інакше:

$$s_k = 0. \quad (3.10)$$

На рисунку 3.1 наведено графічний приклад зсуву вгору. Вилучається завдання №2, а замість нього після зсуву додається завдання №6.

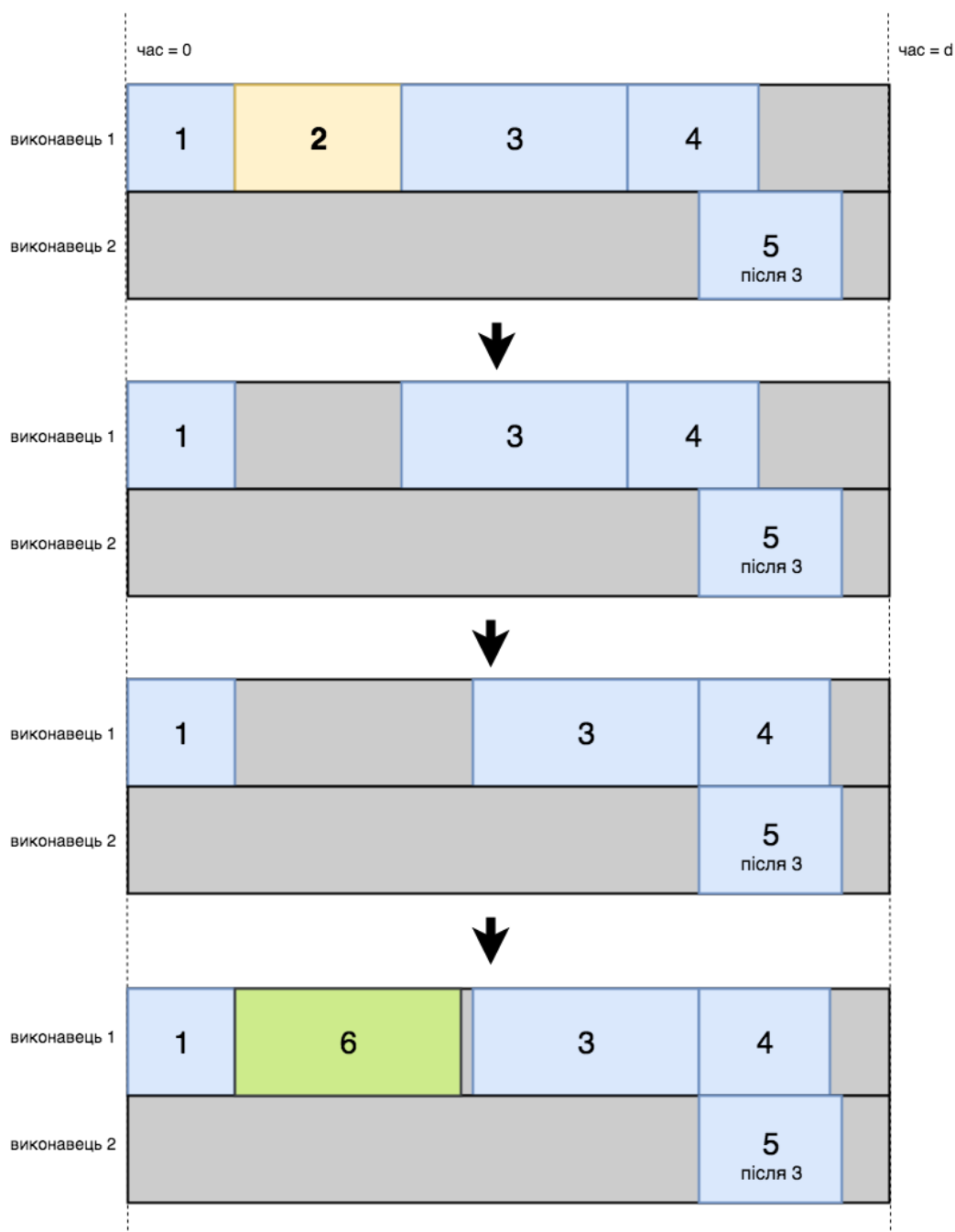


Рисунок 3.1 – Приклад зсуву вниз

Означення 3.2. Відстанню зсуву вгору d_{up} для деякого завдання $j, j \in N$, яке в поточному розв'язку буде виконано виконавцем $i, i \in [1, \dots, m]$, починаючи у момент часу $s_j > 0$ та за час t_{ij} , є проміжок часу, на який можна посунути ближче до моменту початку цього завдання всі завдання $k \in N \setminus \{j\} = K$, які у розкладі виконавця i будуть виконані після завдання j , тобто $x_{ik} = 1, s_k > s_j + t_{ij}$, якщо це завдання $j, j \in N$ вилучити з поточного розкладу для виконавця i . Зрозуміло, що цей проміжок часу залежить від того, чи мають завдання $k \in K$ такі завдання, які

заплановані до них, та чи мають вони завдання, які мають їм передувати. Проміжок часу для кожного завдання $k \in K$, окрім першого у розкладі виконавця i , яке має $prev(k), prev(k) \in N \setminus \{k\}$, обчислюється таким чином:

$$d_{up}(k) = \min\{s_k - (s_{before_k} + t_{ibefore_k}), s_k - (s_{prev(k)} + t_{iprev(k)})\}.$$

Для першого ж завдання $k_{first} \in K$, якщо воно має $prev(k), prev(k) \in N \setminus \{k\}$, цей проміжок часу обчислюється так:

$$d_{up}(k_{first}) = s_{k_{first}} - (s_{prev(k_{first})} + t_{iprev(k_{first})}).$$

Якщо ж завдання $k \in K$ не має $prev(k), prev(k) \in N \setminus \{k\}$, то проміжок часу для всіх завдань, окрім першого, обчислюється таким чином:

$$d_{up}(k) = s_k - (s_{before_k} + t_{ibefore_k}),$$

для першого ж завдання $k_{first} \in K$ формула має такий вигляд:

$$d_{up}(k_{first}) = s_{k_{first}}.$$

Тоді загальна відстань зсуву ввверх обчислюється наступним чином:

$$d_{up} = \min_{k \in K} \{d_{up_k}\}. \quad (3.11)$$

Час початку нового доданого завдання буде обчислюватись так: якщо є завдання $prev(j), prev(j) \in N \setminus \{j\}$ та $k_{last} \in K$, то:

$$s_k = \max\{s_{k_{last}} + t_{ik_{last}}, s_{prev(j)} + t_{iprev(j)}\}. \quad (3.12)$$

інакше, якщо є лише $k_{last} \in K$, то

$$s_k = s_{k_{last}} + t_{ik_{last}}. \quad (3.13)$$

інакше, якщо є лише $prev(j), prev(j) \in N \setminus \{j\}$, то

$$s_k = s_{prev(j)} + t_{iprev(j)}. \quad (3.14)$$

інакше:

$$s_k = 0. \quad (3.15)$$

На рисунку 3.2 зображено схему зсуву ввєрх.

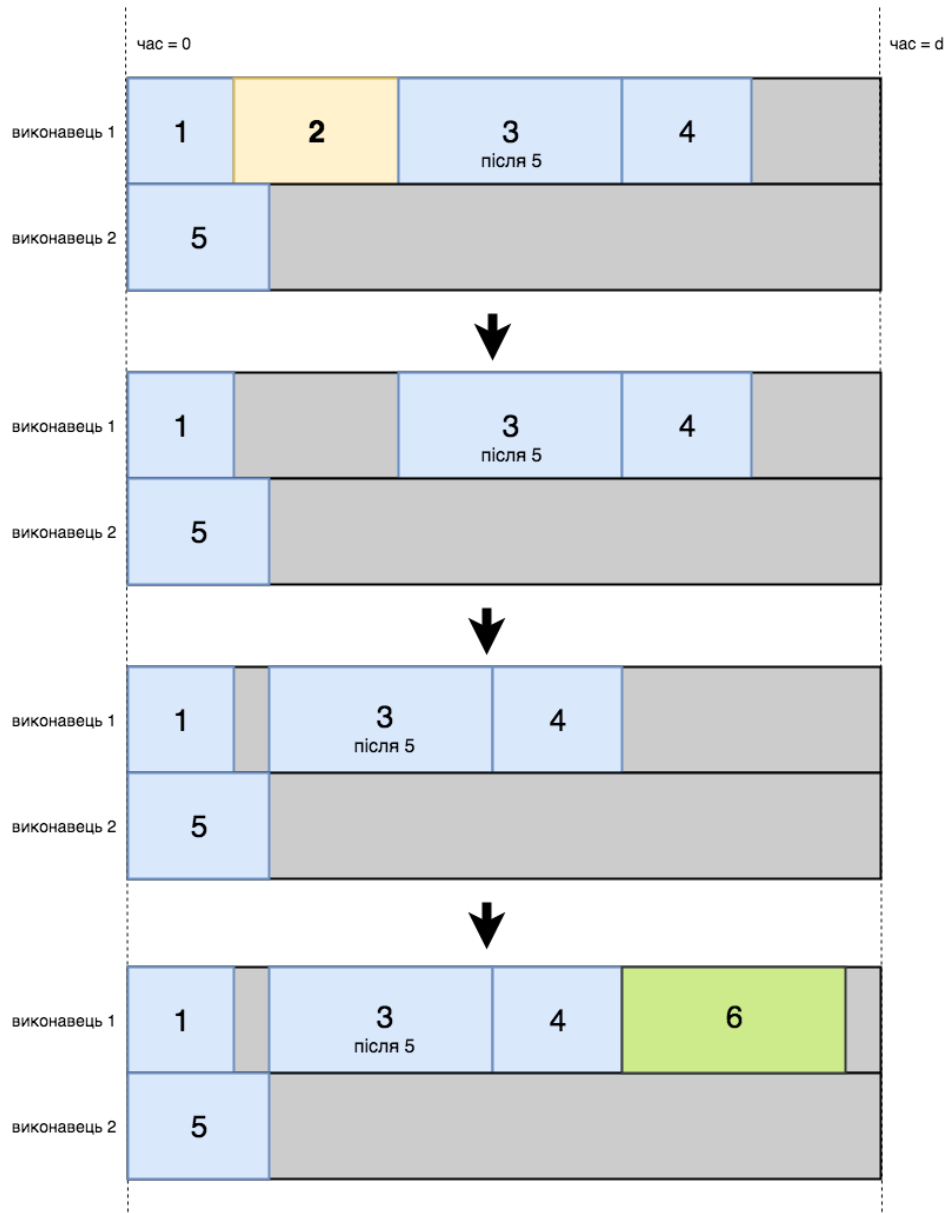


Рисунок 3.2 – Приклад процедури зсуву ввєрх

Наведемо схему процедури генерації сусіднього розв'язку.

Процедура 1 (генерація сусіднього розв'язку).

Крок 1. Доки не знайдено припустимий новий розв'язок або не переглянуто

всі завдання у поточному розв'язку:

1.1. ЯКЩО множина переглянутих завдань у поточному розкладі є порожньою,

ТО вважати перше завдання першого виконавця поточним

ІНАКШЕ вважати наступне після останнього переглянутого завдання поточним

1.2. Вважати, що ще не виконувались зсув ввєрх та зсув вниз;

1.3. ЯКЩО для поточного завдання j визначено завдання $\text{succ}(j), \text{succ}(j) \in N \setminus \{j\}$ та воно є у поточному розкладі, ТО перейти до 1.1.

ІНАКШЕ перейти до 1.4.

1.4. Вилучити поточне завдання j з поточного розв'язку.

1.5. ЯКЩО у поточного завдання є $\text{succ}(j), \text{succ}(j) \in N \setminus \{j\}$ та ще не виконувався зсув ввєрх,

ТО перейти до 1.6.

ІНАКШЕ, ЯКЩО ще не виконувався зсув вниз, ТО перейти до 1.7.

ІНАКШЕ, ЯКЩО ще не виконувався зсув ввєрх, ТО перейти до 1.6.

ІНАКШЕ перейти до 1.1.

1.6. Спробувати виконати зсув ввєрх:

1.6.1. Обчислити відстань зсуву ввєрх d_{up} за (3.11).

1.6.2. Знайти таке завдання $k \in N$ з тих, які не у поточному розв'язку, що його важливість максимальна та $t_{ik} \leq d - (s_{k_{last}} + t_{ik_{last}}) + d_{up}$.

1.6.3. ЯКЩО є таке завдання $k \in N$, ТО додати його до поточного розв'язку до розкладу поточного виконавця з часом початку s_k , обчисленим за формулами (3.12) – (3.15);

Вважати зсув ввєрх виконаним;

Вважати отриманий розв'язок новим сусіднім розв'язком;

Перейти до кроку 1.

ІНАКШЕ

Вважати зсув ввєрх виконаним;

Перейти до 1.5.

1.7. Спробувати виконати зсув вниз:

1.7.1. Обчислити відстань зсуву вниз d_{down} за (3.2).

1.7.2. Знайти таке завдання $k \in N$ з тих, які не у поточному розв'язку, що його важливість максимальна та $t_{ik} \leq gap_{down}$, де gap_{down} обчислюється за формулами (3.2) – (3.6).

1.7.3. ЯКЩО є таке завдання $k \in N$, ТО додати його до поточного розв'язку до розкладу поточного виконавця з часом початку s_k , обчисленим за формулами (3.7) – (3.10);

Вважати зсув вниз виконаним;

Вважати отриманий розв'язок новим сусіднім розв'язком;

Перейти до кроку 1.

ІНАКШЕ

Вважати зсув вниз виконаним;

Перейти до 1.5.

Графічне зображення процедури 1 наведено у додатку 2.

Також необхідно враховувати специфіку алгоритму, який використовує процедуру 1. Для алгоритмів стохастичного локального пошуку впорядкований перебір завдань може виявитися не кращим варіантом, оскільки для них існує імовірність переходу до гіршого розв'язку. Тому має сенс введення двох різних способів переходу до наступного завдання для утворення сусіднього розв'язку. Перший полягає у послідовному перегляді всіх завдань поточного розв'язку, впорядкованих за виконавцем, який їх виконує, та за часом початку. Тобто розглядатиметься така послідовність завдань:

$$j_1^1, j_1^2, \dots, j_m^1, j_m^2, \dots, j_m^{r_m}. \quad (3.16)$$

де r_i - кількість завдань у поточному розкладі для виконавця $i, i \in \overline{1, m}$.

Другий спосіб – вибір випадкового наступного завдання з поточного розв'язку, яке ще не було переглянуте. Для алгоритму детермінованого локального

пошуку використаємо перший спосіб, для алгоритмів імітаційного відпалу та прискореного імовірнісного моделювання – другий.

Критерій завершення перегляду точок у поточному околі та переходу до наступного

Перехід до наступного розв'язку здійснюється при знаходженні першого кращого за поточний розв'язок сусіднього розв'язку.

Спосіб обчислення зміни цільової функції при переході до нового поточного варіанта

Нехай з поточного розв'язку було вилучено деяке завдання $i, i \in N$ та його було замінено завданням $j, j \in N \setminus \{i\}$. Тоді зміна цільової функції обчислюється таким чином:

$$\Delta = v_j - v_i. \quad (3.17)$$

Критерій завершення

В якості критерію завершення обрано відсутність кращих варіантів в околі поточного розв'язку або перевищення заданої максимальної кількості ітерацій.

З урахуванням наведеного вище сформулюємо схему алгоритму ДЛП для задачі планування робіт, що розглядається.

Крок 1. Вважати початковий розв'язок S' поточним.

Крок 2. Доки окіл поточного розв'язку не переглянутий повністю та не перевищено кількість проведених ітерацій,

2.1. Сформулювати наступний розв'язок з околу поточного розв'язку за процедурою 1.

2.2. ЯКЩО сформований розв'язок має більше значення цільової функції за значення поточного розв'язку,

ТО вважати сформований розв'язок поточним;

2.3. Перейти на крок 2.

3.2.3 Алгоритм імітаційного відпалу

Алгоритм детермінованого локального пошуку дозволяє перехід тільки до кращих розв'язків, що обмежує його результати околom початкового розв'язку. Для того, щоб уникнути цього обмеження, можна використати алгоритми стохастичного локального пошуку, в яких за деяких умов дозволяється перехід до гіршого розв'язку.

Саме така ідея лягла в основу алгоритмів *імітаційного*, або *модельованого*, *відпалу* (AIB) (Simulated annealing), які базуються на аналогії з процесом оптимізації та фізичним процесом відпалу. В алгоритмах даного класу пошук глобального розв'язку імітується процесом релаксації термодинамічної системи з багатьма степенями вільності, де значенням узагальненої енергії (гамільтоніана) є цільова функція задачі. Зі статистичної механіки відомо, що такі системи при деякій кінцевій температурі прямують до стану рівноваги. Флуктуації, що виникають при високих температурах, мають імовірнісний характер і можуть зумовлювати тимчасове збільшення енергій системи, однак при поступовому зниженні температури термодинамічна система досягає стану рівноваги, що відповідає мінімальному значенню узагальненої енергії системи. У фізичному аспекті *відпал* - це тепловий процес для отримання стану з низькою енергією тіла в тепловій ванні. Загалом процес можна описати так. Спочатку температура теплової ванни збільшується до максимального значення, при якому тверде тіло тане. Таким чином, усі частинки тіла отримують значну свободу. Згодом температура поступово зменшується спеціальним чином до такого стану, у якому тіло твердне, а його частинки організуються у високоструктуровану ґратку з мінімальною енергією[73].

Якщо система перебуває в певному стані, то вона з однаковою ймовірністю може перейти як у вищий, так і в нижчий стани. Однак не виключаються її флуктуації (коливання), але врешті-решт зі зменшенням температури T система "заморожується" [73].

Початковий етап пошуку (при високій температурі) близький до випадкового блукання, але при зменшенні значень параметра T усе більше наближається до

стратегії детермінованого ЛП. Основною задачею при розробці алгоритмів, які відносяться до схеми алгоритму імітаційного відпалу, є вибір правильного температурного режиму. При його виборі необхідно враховувати особливість задачі, для розв'язування якої розробляється алгоритм.

У додатку 1 наведено загальну блок-схему алгоритму імітаційного відпалу.

Наведемо ключові аспекти реалізації алгоритмів імітаційного відпалу.

- поняття околу та перебір точок в околі, тобто спосіб генерації точок.
- імовірність переходу від поточного варіанта до нової точки.
- принцип рівноваги. Задаються параметри $\varepsilon > 0$ та ν – натуральне, а ν переходів у алгоритмі називається *прогоном*. Тоді, якщо здійснюється k прогонів і маємо значення f_1, \dots, f_k (характеристики цільових функцій на цих прогонах), то на $(k + 1)$ -му прогоні вважається досягнутою рівновага, якщо

$$\exists i = \overline{1, k} : |f_i - f_{k+1}| / f_i \leq \varepsilon. \quad (3.18)$$

- температурний розклад (правило зміни значення параметра T).

Опишемо ці аспекти для алгоритму, що пропонується.

Поняття околу та перебір точок в околі, тобто спосіб генерації точок

Для перебору точок в околі використовується процедура 1 з випадковим перебором завдань.

Імовірність переходу від поточного варіанта до нової точки

Нехай для розв'язку S знайдено деякий припустимий сусідній розв'язок S_{new} .

Тоді спочатку обчислюється різниця цільових функцій Δ за формулою (3.17). Після цього генерується випадкове число $r = \text{random}[0,1]$. Тоді імовірність переходу до нового розв'язку обчислюється так:

$$p = e^{\frac{-\Delta}{T}}. \quad (3.19)$$

Якщо $r < \min\{p, 1\}$, то перехід відбувається.

Принцип рівноваги

Нехай є початковий розв'язок S_{start} , і в ньому $n_{S_{start}}$ робіт. Тоді довжина прогону визначається таким чином:

$$length = \frac{n_{S_{start}}}{m}. \quad (3.20)$$

А f_i обчислюється для прогону як довжиною $length$ таким чином:

$$f_i = \frac{\sum_{w=1}^{length} f_w}{length}. \quad (3.21)$$

Температурний розклад (правило зміни значення параметра T)

Наступне значення параметру T_r на ітерації під номером r обчислюється за формулою

$$T_r = T_{r-1} \cdot \alpha. \quad (3.22)$$

де $\alpha \in [0,1]$.

Наведемо схему АІВ, що пропонується.

Крок 1. Вважати початковий розв'язок S' поточним.

Крок 2. Обчислити необхідну кількість ітерацій v для одного прогону за формулою (3.20).

Крок 3. Доки температура не досягла мінімально припустимого значення T_{min} :

3.1. Доки не досягнуто балансу за умовами (3.21) та (3.18):

3.1.1. Доки не проведено визначену кількість ітерацій v для прогону:

3.1.1.1. Згенерувати новий розв'язок з околу поточного розв'язку за процедурою 1;

3.1.1.2. ЯКЩО новий розв'язок має більше значення цільової функції за поточний розв'язок

ТО вважати новий розв'язок поточним

Перейти до 3.1.1;

3.1.1.3. Розрахувати імовірність переходу до гіршого розв'язку $p \in [0,1]$ за формулою (3.19);

3.1.1.4. Згенерувати випадкове число $t \in [0,1]$.

3.1.1.5. ЯКЩО $t < p$

ТО вважати новий розв'язок поточним

Перейти до 3.1.1

ІНАКШЕ перейти до 3.1.1.1

3.1.2. Перейти до 3.1.

3.2. Обчислити нове значення температури за формулою (3.22).

3.2.4 *G*-алгоритм

Досвід застосування алгоритмів імітаційного відпалу показав, що отримувані результати мають досить високу точність, ефективно реалізуються на БОК. Проте обчислювальні експерименти виявили суттєві затрати машинного часу на пошук розв'язків, а також значні коливання точності отриманих результатів залежно від вибору значень параметрів алгоритмів, що варіюються.

З метою зменшення часових затрат і підвищення стабільності результатів була запропонована сім'я алгоритмів прискореного ймовірнісного моделювання, у яких використовується інша ймовірнісна модель; вони отримали назву *G*-алгоритмів [73]. В цих алгоритмах також здійснюється побудова точок в околі поточного варіанта і поліпшуючі варіанти завжди приймаються як чергове наближення, а варіанти, що відповідають погіршенню (зростанню) цільової функції, теж можуть бути вибрані з деякою ймовірністю. Проте, на відміну від АІВ, значення цієї ймовірності розраховуються однаково впродовж усього обчислювального процесу, але змінюється порогове значення, яке й визначає умови відсіювання погіршуючих варіантів [73].

Для побудови обчислювального процесу формується строго монотонно

зростаюча послідовність дійсних чисел $0 \leq \mu_0 < \mu_1 < \dots \leq 1$, які відіграють роль, у певному розумінні подібну до ролі параметра температури T в АІВ. Якщо x^h – це поточний варіант на кроці h алгоритму, то досліджуваний варіант $y \in O(x^h)$, для якого $f(y) < f(x^h)$ (оскільки ми розглядаємо задачу максимізації), може бути прийнятий як x^{h+1} . якщо розрахована ймовірність $p(x^h, y)$ перевищить поточне значення μ_i , збільшене на певну випадкову величину.

Нехай $\gamma, \gamma > 0$, – дійсне число. Задамо для всіх $x, y \in X$ функцію

$$\phi(x, y) = 1 - \frac{f(x) - f(y)}{f(x) \cdot \gamma}. \quad (3.23)$$

а на її підставі – ймовірність переходу від точки x до y :

$$p(x, y) = \begin{cases} \min\{1, \phi(x, y)\}, & \phi(x, y) \geq 0, \\ 0 & \text{в іншому разі.} \end{cases} \quad (3.24)$$

У додатку 1 наведено загальну блок-схему G-алгоритму.

Слід зазначити, що в обчислювальній схемі G-алгоритму досить просто врахувати багато типів обмежувальних умов, які часто зустрічаються на практиці: відповідні умови перевірки на припустимість можуть бути враховані при породженні точок з околу поточного варіанта. Це дає змогу розв'язувати не одну, а відразу цілий підклас задач, причому обмеження й деякі дані задачі, що розв'язується, можуть змінюватися в процесі її розв'язання.

Отже, *ідеологія* методів прискореного ймовірнісного моделювання полягає в тому, що на кожному кроці, маючи x^0, x^1, \dots як центри, будуємо точки їх околу із заданим радіусом і дозволяємо перехід не лише до поліпшуючих (за значенням цільової функції), але й до погіршуючих точок з певною ймовірністю. З часом ця ймовірність переходу до гірших точок зменшується (якщо в алгоритмах імітаційного відпалу T прямує до нуля, а в G-алгоритмі штучно підводимо поріг до 1). Таким чином, на завершальних ітераціях цей метод у наведеній вище інтерпретації фактично перетворюється на стандартний локальний пошук.

До переваг G-алгоритмів варто віднести:

- зменшення залежності від початкового наближення;
- застосовність до розв'язання задач із різними обмежувальними умовами.

Ключові аспекти реалізації G-алгоритмів включають в себе:

- механізм побудови послідовності μ_i ;
- способи генерації точок з околу поточного варіанта;
- умови рівноваги при даному значенні величини μ_i ;
- правила зупинки.

Наведемо ці аспекти реалізації для алгоритму, що пропонується.

Механізм побудови послідовності μ_i

Застосуємо класичний підхід до побудови цієї послідовності: шляхом визначення й використання деякої строго монотонної функції, яка зазвичай позначається через G (звідси скорочена назва – (G-алгоритми). В алгоритмі, що пропонується, вона має такий вигляд:

$$G(x) = x + b.$$

Тоді наступне значення послідовності μ_{i+1} обчислюється таким чином:

$$\mu_{i+1} = G(\mu_i). \quad (3.25)$$

Способи генерації точок з околу поточного варіанта

Для генерації сусідніх розв'язків використано процедуру 1 з випадковим перебором завдань.

Умови рівноваги при даному значенні величини μ_i

Умови рівноваги співпадають з умовами рівноваги алгоритму АІВ.

Правила зупинки

Алгоритм зупиняє свою роботу, якщо окіл поточного кращого розв'язку порожній або ж якщо досягнуто мінімального значення $\mu_{\min} = 0$.

Наведемо схему G-алгоритму, що пропонується.

Крок 1. Вважати початковий розв'язок S' поточним.

Крок 2. Обчислити необхідну кількість ітерацій ν для одного прогону за формулою (3.20).

Крок 3. Доки значення μ не досягло мінімально припустимого значення $\mu_{\min} = 0$:

3.1. Доки не досягнуто балансу за умовами (3.21) та (3.18):

3.1.1. Доки не проведено визначену кількість ітерацій ν для прогону:

3.1.1.1. Згенерувати новий розв'язок з околу поточного розв'язку за процедурою 1;

3.1.1.2. ЯКЩО новий розв'язок має більше значення цільової функції за поточний розв'язок

ТО вважати новий розв'язок поточним

Перейти до 3.1.1;

3.1.1.3. Розрахувати імовірність переходу до гіршого розв'язку $p \in [0,1]$ за формулою (3.24);

3.1.1.4. ЯКЩО $m_{\text{current}} < p$

ТО вважати новий розв'язок поточним

Перейти до 3.1.1

ІНАКШЕ перейти до 3.1.1.1

3.1.2. Перейти до 3.1.

3.2. Обчислити нове значення μ за формулою (3.25).

Висновок до розділу

Описано розроблені алгоритми розв'язування отриманих для поставленої задачі підзадач.

Перша підзадача – розподіл обов'язкових завдань між виконавцями – є узагальненою задачею про призначення, для її розв'язування використано існуючий алгоритм, який полягає у використанні деякого алгоритму пакування рюкзака та

використання його послідовно для всіх виконавців [70]. Наведено схему використаного алгоритму.

Друга підзадача є задачею планування з додатковими обмеженнями. Для отримання її початкового припустимого розв'язку використовується жадібний алгоритм, який враховує також розподіл обов'язкових завдань, отриманий в результаті розв'язування першої підзадачі. Отриманий початковий розв'язок використовується як відправна точка для трьох різних алгоритмів локального пошуку: алгоритму детермінованого локального пошуку, алгоритму імітаційного відпалу та алгоритму прискореного імовірнісного моделювання. Для запропонованих алгоритмів розроблено спосіб генерації точок околу, які визначають сусідство розв'язків, який базується на поняттях зсувів, наведено ключові аспекти реалізації всіх алгоритмів та їх схеми.

4 ОПИС РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Призначення програмного забезпечення

Розроблене програмне забезпечення призначене для автоматизації процесу оптимального планування виконання завдань шляхом розв'язування екземплярів задачі розробленими алгоритмами.

Програмне забезпечення має такий функціонал:

- зчитування вхідних даних задачі з обраного користувачем файлу;
- розв'язування задачі розробленими алгоритмами;
- виведення результатів, отриманих кожним із алгоритмів;
- збереження отриманого конкретний алгоритмом розв'язку у файл;
- перегляд інформації про вхідні та вихідні дані.

4.2 Засоби розробки

Для розробки програмного забезпечення обрано мови програмування Java та Kotlin.

Мова програмування Java на сьогоднішній день є однією з найвідоміших та найбільш використовуваних у світі.

Java – це паралельна, заснована на понятті класів мова програмування загальної направленості. Її було спроектовано так, щоб якомога більше розробників програмного забезпечення змогли з легкістю її опанувати. Вона пов'язана з C та C++, але організована по-іншому, вона має декілька аспектів, яких немає у C та C++, та деякі особливості інших мов програмування. [74]

Це мова програмування високого рівня, основними характеристиками якої є :

- простота;
- об'єктно-орієнтованість;
- розподіленість;
- наявність механізму багатопоточності;

- динамічність;
- незалежність від архітектури;
- портативність;
- висока продуктивність;
- надійність;
- безпечність [75].

Kotlin - це статично типізована мова програмування для JVM, яку розроблює компанія JetBrains, починаючи з 2010 року [76]. Основними перевагами цієї мови, як стверджують її автори, є лаконічність, безпечність коду, сумісність з іншими мовами як Java та можливість використовувати будь-які інтегровані середовища розробки, які підтримують Java [77]. Мова Kotlin є:

- крос-платформною, оскільки код запускається на JVM;
- статично типізованою;
- функціональною (функції можна зберігати як змінні та передавати в якості параметрів в інші функції, об'єкти розділені на змінні та постійні (mutable та immutable));
- об'єктно-орієнтованою;
- безкоштовною та з відкритим вихідним кодом [78].

Основною ціллю мови Kotlin є створення лаконічнішої, продуктивнішої та безпечнішої альтернативи Java, яку можна буде використати в усіх сферах, те Java на сьогоднішній день набула розповсюдження. Java – дуже популярна мова програмування, і сфери її використання включають широкий спектр середовищ, від карток з чіпами до найбільших центрів збору інформації Google, Twitter, LinkedIn та інших великих компаній. В більшості цих місць використання мови Kotlin може допомогти розробникам досягати своїх цілей з меншим обсягом коду та меншою кількістю проблем на шляху. [79, 80]

Незважаючи на те, що ця мова є відносно новою, її вже використовують такі великі компанії, як Pinterest, Evernote, Uber, Coursera та Atlassian [77]. Також

Googlena даний момент працюють над повною підтримкою Kotlin для Android-розробки [79]. Все це говорить про серйозне майбутнє цієї мови програмування.

В якості інтегрованого середовища розробки обрано IntelliJ IDEA від компанії-автора Kotlin – JetBrains. Це є дуже зручним, оскільки вона повністю підтримує роботу з Kotlin та надає багато можливостей, які полегшують розробку. Потенційним аналогом цього середовища розробки були такі середовища, як Eclipse та NetBeans, але підтримка Kotlin у них є значно менш розвиненою, тому було обрано саме IntelliJ IDEA.

Для розробки графічного інтерфейсу програми обрано бібліотеку Java Swing з пакету `javax.swing`.

Вона має наступні переваги:

- включає в себе більший та зручніший набір елементів користувацького інтерфейсу;
- значно менше, ніж AWT, залежить від платформи, на якій виконуватиметься програма – з цього випливає, що вона менш схильна до помилок конкретних платформ;
- забезпечує однакове сприйняття кінцевими користувачами додатків з графічним користувацьким інтерфейсом на різних платформах [81].

Також Swing надає можливість зробити інтерфейс програми як за спеціальною темою Metal, так і за стилем платформи, на якій програму запущено [81]. Для кращої портативності бібліотека Swing повністю написана на Java[82]. Бібліотека Swing має всі компоненти, необхідні для створення сучасного стильного інтерфейсу, - починаючи кнопками з ілюстраціями і закінчуючи деревами та таблицями. Вона складається з великої колекції компонентів JavaBean, зрозумілих та простих у використанні; вони гарно працюють і для самостійної побудови графічного інтерфейсу, і в середовищах візуальної розробки [82].

4.3 Опис програмної реалізації

На рисунку 4.1 наведено схему структурну варіантів використання розробленого програмного забезпечення.

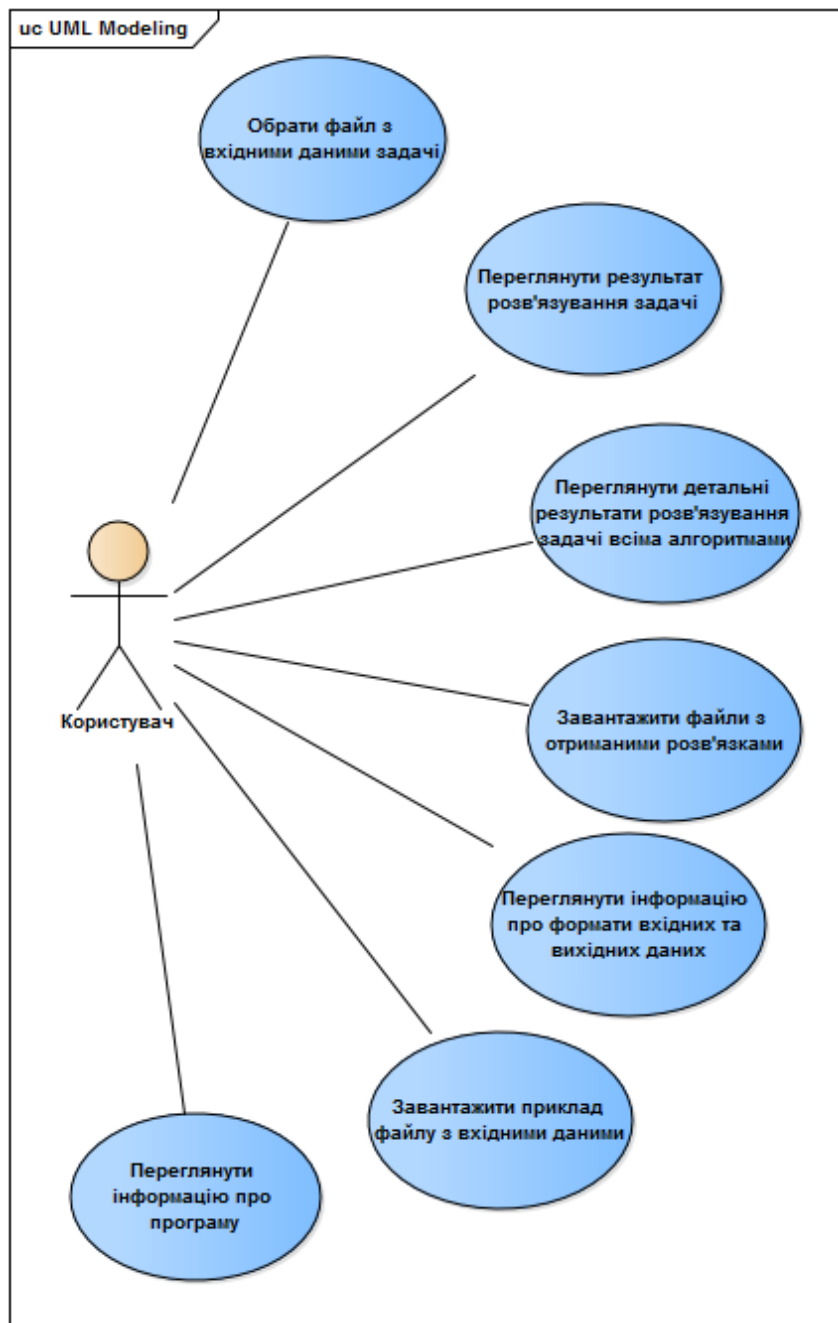


Рисунок 4.1 – Схема структурна варіантів використання

У таблиці 4.1 наведено опис класів модулю розв'язування підзадачі 1.

Таблиця 4.1 – Класи модулю розв’язування підзадачі розподілу обов’язкових завдань

Назва класу	Функціональність
Knapsack	Клас, який містить в собі реалізацію алгоритму розв’язування задачі про рюкзак
GAPAlgorithm	Клас, який містить в собі реалізацію алгоритму розв’язування узагальненої задачі про призначення
TeamMember	Клас, який представляє собою учасника команди, має номер
Task	Представляє собою завдання, яке має номер

На рисунку 4.2 наведено схеми структурну класів модулю, в якому розміщено програмну реалізацію алгоритмів для розв’язування першої підзадачі з підрозділу 2.4.

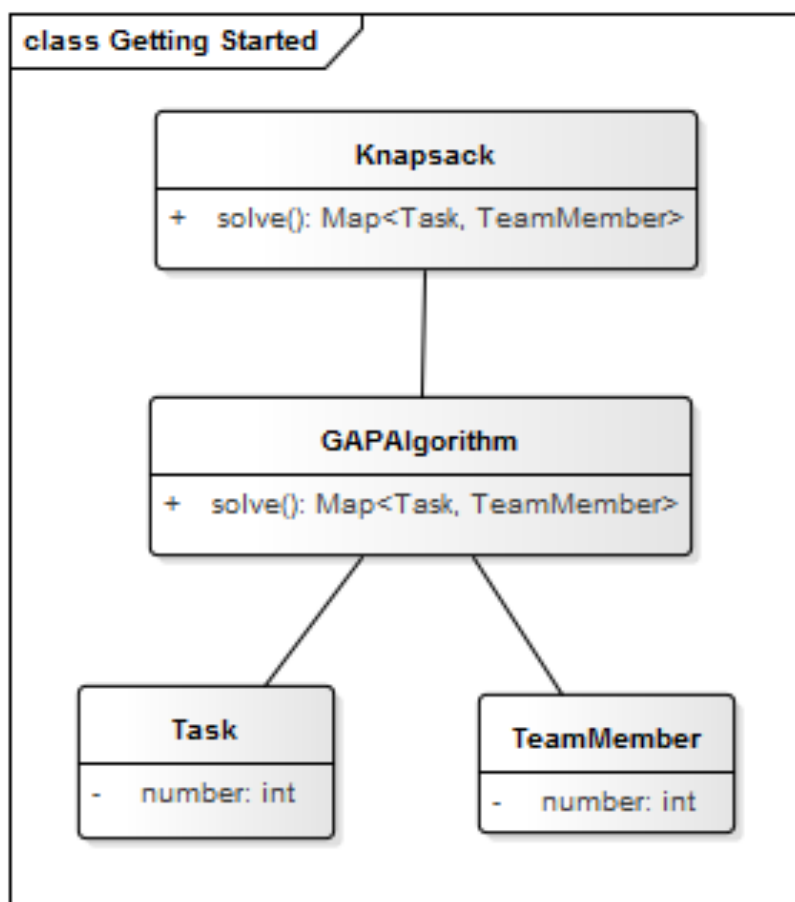


Рисунок 4.2 – Схема структурна класів модулю розв’язування підзадачі розподілу обов’язкових завдань

У таблиці 4.2 наведено опис класів модулю розв'язування підзадачі 2.

Таблиця 4.2 – Класи модулю розв'язування підзадачі планування необов'язкових завдань

Назва класу	Функціональність
GreedyAlgorithm	Клас, який містить в собі реалізацію жадібного алгоритму для знаходження початкового припустимого розв'язку
LocalSearchAlgorithm	Клас, який містить в собі реалізацію алгоритму детермінованого локального пошуку
SimulatedAnnealingAlgorithm	Клас, який містить в собі реалізацію алгоритму імітаційного відпау
GAlgorithm	Клас, який містить в собі реалізацію алгоритму прискореного моделювання
OrderedNeighborGenerator	Клас, який генерує наступний розв'язок в околі, зберігаючи впорядкованість розв'язків
RandomizedNeighborGenerator	Клас, який генерує наступний розв'язок в околі, обираючи його випадковим чином, але такий, що не повторюється
Schedule	Клас, який представляє собою розклад, ставлячи у відповідність учасників команди та завдання
TeamMember	Клас, який представляє собою учасника команди, має номер
Task	Представляє собою завдання, яке має номер

На рисунку 4.3 наведено схему структурну класів модулю, в якому розміщено програмну реалізацію алгоритмів для розв'язування другої підзадачі з підрозділу 2.4.

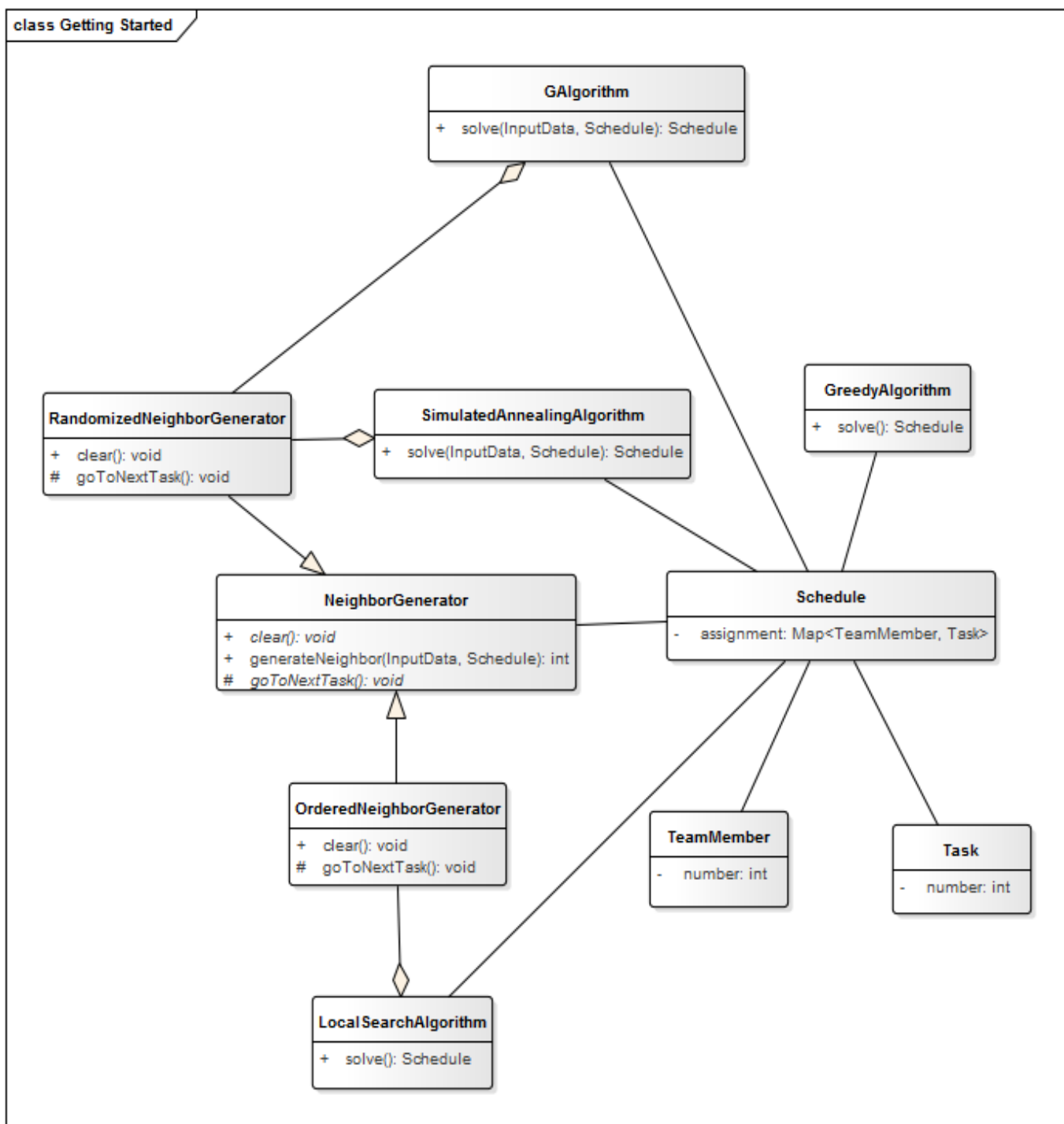


Рисунок 4.3 – Схема структурна класів модулю розв’язування підзадачі планування необов’язкових завдань

У таблиці 4.3 описано основні функції класів модулів розв’язування обох підзадач.

Таблиця 4.3 – Опис основних функцій модулів розв'язування першої та другої підзадач

Клас	Функція	Призначення
GAPAlgorithm	solve()	Розв'язує узагальнену задачу про призначення та повертає розподіл завдань між виконавцями
GreedyAlgorithm	solve()	Розв'язує задачу жадібним алгоритмом та повертає результат розв'язування
OrderedNeighborGenerator	getNextNeighbor()	Генерує наступний послідовний розв'язок в околі та повертає його
RandomizedNeighborGenerator	getNextNeighbor()	Генерує наступний випадковий розв'язок в околі та повертає його
LocalSearchAlgorithm	solve()	Розв'язує задачу алгоритмом детермінованого локального пошуку та повертає результат розв'язування
SimulatedAnnealingAlgorithm	solve()	Розв'язує задачу алгоритмом імітаційного відпалу та повертає результат розв'язування
GAlgorithm	solve()	Розв'язує задачу алгоритмом прискореного моделювання та повертає результат розв'язування

У таблиці 4.4 наведено опис основних класів модулю користувацького інтерфейсу та їх функцій.

Таблиця 4.4 – Опис основних функцій модулю користувацького інтерфейсу

Клас	Функція	Призначення
MainForm	chooseInputFile()	Показує діалогове вікно вибору файлу з вхідними даними задачі
	solve()	Запускає розв'язування задачі з даними з вхідного файлу всіма алгоритмами

Клас	Функція	Призначення
ResultsForm	showResults()	Показує результати розв'язування обраної задачі всіма алгоритмами
	openDetailsForResult()	Відкриває вікно з детальною інформацією про окремий розв'язок
	downloadResult()	Зберігає файл з розв'язком на комп'ютері користувача за вказаною адресою
ResultsDetailsForm	showDetails()	Показує докладну інформацію про отриманий розв'язок для конкретного алгоритму, обраного користувачем
InputFileForm	downloadSample()	Відкриває діалогове вікно вибору директорії, в яку буде збережено приклад, та записує туди файл з форматом оформлення вхідних даних

У програмі також є інші форми, але їх функції полягають лише у відображенні деякої статичної інформації, тому їх не було внесено в таблицю 4.4. Ці форми включають в себе OutputFileForm, LoadingForm та HowItWorksForm.

На рисунку 4.4 наведено схему структурну класів модулю користувацького інтерфейсу.

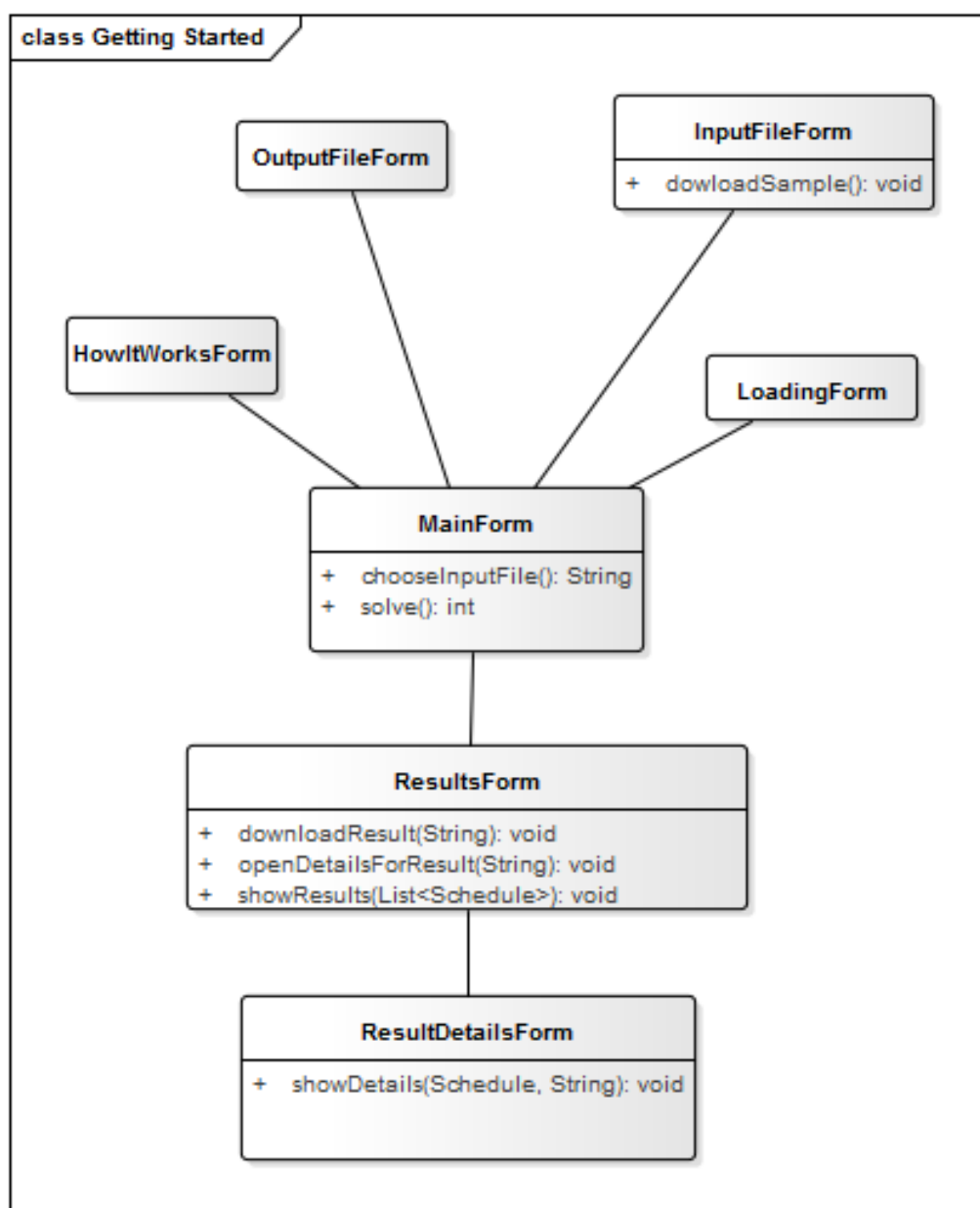


Рисунок 4.4 – Схема структурна класів модулю користувацького інтерфейсу

Для коректної роботи програми файл із вхідними даними задачі має відповідати визначеному форматові. У цьому файлі необхідно вказати наступну інформацію:

- кількість виконавців m ;
- кількість завдань n ;
- важливості завдань v_1, v_n ;
- час на виконання усіх завдань t_{11}, t_{mn} ;
- попередні завдання для завдань необов'язкових завдань k_1, k_n ; якщо завдання не має попередника, необхідно вказати 0;

- множину обов'язкових завдань Q ;
- директивний термін d .

Цей формат наведено на рисунку 4.5.

```

m
n
v_1
...
v_n
t_1_1
t_1_2
...
t_1_n
t_2_1
t_2_2
...
t_2_n
...
t_m_n
k1
k2
k3
...
kn
[1,2] (Q)
d

```

Рисунок 4.5 – Формат вхідних даних

На рисунку 4.6 наведено приклад вихідного файлу, який створює програма при завантаженні отриманого одним з алгоритмів розв'язку.

Результати розв'язування задачі 100_5x200.txt алгоритмом AIB.

Сумарна важливість: 2839

Розподіл завдань:

Учасник №1: [завдання №47, час початку: 0, час виконання: 27],
 [завдання №63, час початку: 27, час виконання: 19], [завдання №80, час
 початку: 46, час виконання: 5], [завдання №100, час початку: 51, час
 виконання: 4], [завдання №109, час початку: 55, час виконання: 5],
 [завдання №132, час початку: 60, час виконання: 33], [завдання №161,
 час початку: 93, час виконання: 3], [завдання №31, час початку: 96, час
 виконання: 1], [завдання №46, час початку: 97, час виконання: 1],
 [завдання №95, час початку: 98, час виконання: 1], [завдання №192, час
 початку: 99, час виконання: 1].

Учасник №2: [завдання №18, час початку: 0, час виконання: 19],
 [завдання №50, час початку: 19, час виконання: 27], [завдання №55, час
 початку: 46, час виконання: 14], [завдання №106, час початку: 60, час
 виконання: 8], [завдання №120, час початку: 68, час виконання: 20],
 [завдання №150, час початку: 88, час виконання: 12].

Учасник №3: [завдання №17, час початку: 0, час виконання: 27],
 [завдання №22, час початку: 27, час виконання: 23], [завдання №36, час
 початку: 50, час виконання: 2], [завдання №41, час початку: 52, час
 виконання: 27], [завдання №91, час початку: 79, час виконання: 11],
 [завдання №144, час початку: 90, час виконання: 6], [завдання №39, час
 початку: 96, час виконання: 1], [завдання №105, час початку: 97, час
 виконання: 1], [завдання №35, час початку: 98, час виконання: 1].

Учасник №4: [завдання №152, час початку: 0, час виконання: 46],
 [завдання №128, час початку: 46, час виконання: 1], [завдання №89, час
 початку: 47, час виконання: 1], [завдання №6, час початку: 48, час
 виконання: 19], [завдання №122, час початку: 67, час виконання: 16],
 [завдання №103, час початку: 83, час виконання: 14].

Учасник №5: [завдання №42, час початку: 0, час виконання: 3],
 [завдання №12, час початку: 3, час виконання: 10], [завдання №163, час
 початку: 13, час виконання: 3], [завдання №72, час початку: 16, час
 виконання: 22], [завдання №69, час початку: 38, час виконання: 1],
 [завдання №189, час початку: 39, час виконання: 16], [завдання №195,
 час початку: 55, час виконання: 32].

Рисунок 4.6 – Приклад вихідного файлу з розв'язком, створеного програмою

Як видно з рисунку 4.6, програма виводить сумарну важливість обраних завдань та розклад для кожного учасника.

У додатку 3 наведено схему структурну послідовності роботи програми, у додатку УУ – загальну схему структурну класів розробленого програмного забезпечення, у додатку 5 – схему структурну пакетів.

Висновок до розділу

Програмно реалізовано запропоновані алгоритми детермінованого локального пошуку, імітаційного відпалу та прискореного імовірнісного моделювання. Також

реалізовано жадібний алгоритм для знаходження початкового припустимого розв'язку та алгоритм розподілу обов'язкових завдань між виконавцями.

Для програмної реалізації обрано мови програмування Java та Kotlin. Java – класична об'єктно-орієнтована мова програмування, вона є однією з найпоширеніших мов програмування на сьогоднішній день. Kotlin – нова мова програмування, яка компілюється у байт-код для JVM та стає дедалі поширенішою з моменту свого створення у 2010 році. Для них було обрано інтегроване середовище розробки IntelliJ IDEA через його сумісність з Kotlin та деяку функціональність, якої не мають аналогічні середовища. Для розробки графічного інтерфейсу було обрано класичну бібліотеку мови Java Swing.

Програмне забезпечення дозволяє обрати файл із вхідними даними, переглянути інформацію про програму та формати файлі вхідних та вихідних файлів та отримати результати розв'язування задачі всіма реалізованими алгоритмами. Також є можливість переглянути детальну інформацію про процес розв'язування задачі кожним з трьох алгоритмів локального пошуку.

Розроблене програмне забезпечення може бути використане для розв'язування екземплярів задачі, яка розглядається у дисертації.

5 ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ АЛГОРИТМІВ

5.1 Процес проведення експериментів

Для перевірки ефективності запропонованих алгоритмів проведено серію експериментів. Ці експерименти включають в себе розв'язування наборів задач, які згенеровані випадковим чином та відрізняються за деякими змінними характеристиками, а також аналіз отриманих результатів.

Порівнюються результати роботи алгоритмів локального пошуку, отримані при використанні кожного з алгоритмів окремо. Тобто спочатку знайдено розв'язок першої підзадачі (розподіл обов'язкових завдань), після цього знайдено розв'язок другої підзадачі жадібним алгоритмом з 3.2.1 і його використано як початкове наближення для алгоритмів ДЛП, АІВ та G-алгоритму.

Розглядаючи GAP (див. 3.1), позначимо

$$f_{GAP} = \sum_{j \in Q} v_j. \quad (5.1)$$

де Q – множина обов'язкових завдань,

v_j – важливість завдання j , $j \in N$.

Оцінку точності $\Delta ЖАД$ жадібного алгоритму будемо розраховувати за формулою

$$\Delta ЖАД = \frac{f_{ЖАД} - f_{GAP}}{f_{GAP}} \cdot 100\%, \quad (5.2)$$

де $f_{ЖАД}$ – значення цільової функції, отримане жадібним алгоритмом для конкретного екземпляру задачі. Нехай w – кількість задач в наборі, на якому проведено експерименти.

Усереднення попередньої оцінки на наборі задач розраховується так:

$$\Delta_{ЖАД}^{сер} = \frac{\sum_{i=1}^w \Delta_{ЖАД}}{w} \cdot 100\%. \quad (5.3)$$

Оцінка точності $\Delta(*)$ ДЛП, АІВ та G-алгоритму обчислюється за формулою:

$$\Delta(*) = \frac{f_* - f_{ЖАД}}{f_{ЖАД}} \cdot 100\%, \quad (5.4)$$

де f_* – краще значення цільової функції, отримане відповідним алгоритмом за всі тестові запуски для конкретного екземпляру задачі. Для АІВ та G-алгоритму проводилося по 100 запусків для кожної задачі з набору.

Позначимо

$$\Delta(*)_{сер} = \frac{\sum_{i=1}^w \Delta(*)}{w} \cdot 100\%. \quad (5.5)$$

5.1.1. Постійні параметри вхідних даних

Для проведення експериментів використовувалися вхідні дані для задач з такими параметрами:

- директивний термін $d = 100$ одиниць часу;
- кількість виконавців $m = 10$;
- кількість завдань $n = 500$;
- важливість кожного завдання $v_j \in [0,1,\dots,100]$ одиниць важливості.

5.1.2 Змінні параметри вхідних даних

Задача, що розглядається, має велику кількість характеристик, від яких залежить ефективність роботи запропонованих алгоритмів. Виокремимо основні:

- співвідношення між кількостями обов’язкових q та необов’язкових p завдань;
- кількість впорядкованих необов’язкових завдань;
- розкид тривалості завдань t_{ij} .

Необхідно дослідити вплив зміни цих параметрів на кінцеві результати, отримані кожним із алгоритмів.

5.1.3 Параметри алгоритмів

Алгоритм детермінованого локального пошуку

Єдиним фіксованим параметром алгоритму детермінованого локального пошуку є максимальна кількість ітерацій, при проведенні експериментів використано значення

$$I_{\max} = 100.$$

Алгоритм імітаційного відпалу

Для алгоритму імітаційного відпалу при проведенні експериментів використано наступні значення параметрів:

$$\varepsilon = 0.1 \cdot f_{r-1},$$

де r – номер поточної ітерації для деякого значення температури T_r .

$$T_{\max} = 1,$$

$$T_{\min} = 0.00001,$$

$$\alpha = 0.95.$$

G-алгоритм

Для G-алгоритму при проведенні експериментів використано такі значення параметрів:

$$b = 0.001.$$

5.1.4 Характеристики апаратного забезпечення

Експерименти проводилися на комп'ютері з такими характеристиками:

- обсяг ОЗУ: 8 Гб;
- тактова частота процесора: 2.9 ГГц.

5.2 Співвідношення між кількостями обов'язкових та необов'язкових завдань

Для оцінки впливу кількості обов'язкових та необов'язкових завдань на ефективність роботи кожного з запропонованих алгоритмів проведено серію експериментів зі змінною часткою обов'язкових завдань, інші параметри було зафіксовано.

Час виконання кожного завдання деяким виконавцем визначався в інтервалі:

$$t_{\min} = \frac{d}{100} \text{ одиниць часу,}$$

$$t_{\max} = \frac{d}{2} \text{ одиниць часу.}$$

Тоді час виконання деякого завдання $j \in N$ виконавцем $i \in \{1, 2, \dots, m\}$ визначається так:

$$t_{ij} = \text{random}[t_{\min}, t_{\max}] \text{ одиниць часу.}$$

Нехай кількість обов'язкових завдань визначається так:

$$q = k_q \cdot n,$$

де $k_q \in [0, 1]$.

Змінюватимемо параметр k_q в інтервалі $k_q \in [0.01, 0.3]$ з кроком 0.01.

У таблиці 5.1 наведено результати випробувань для жадібного алгоритму зі змінною часткою обов'язкових завдань.

Таблиця 5.1 – Результати випробувань жадібного алгоритму для змінної частки обов'язкових завдань

k_q	$\Delta ЖАД, \%$	$t_{сер}, мс.$
0.01	650	5.6
0.02	573.84	9.4
0.03	376.56	4.2
0.04	286.25	6.7
0.05	204.26	7.2
0.06	150	5.1
0.07	107.33	8.1
0.08	100.45	15
0.09	82.42	4.7
0.1	73	6.8
0.11	62	7.1
0.12	43.81	5.7
0.13	42.82	5.4
0.14	41.49	5.6
0.14	41.49	5.6
0.15	30.62	4.8
0.16	28.7	4.9
0.17	22.38	4.5
0.18	18.92	5.2
0.19	15.03	4.8
0.2	15.25	4.6
0.21	14.74	4.2
0.22	12.71	3.6
0.23	11.36	3.1
0.24	5.95	3.3
0.25	3.18	2.8
0.26	3.76	2.5
0.27	3.94	1.4
0.28	2.11	1.5
0.29	1.96	1
0.3	2.3	1.2

У додатку 6 наведено графічне зображення результатів роботи жадібного алгоритму зі зміною частки обов'язкових завдань.

З таблиці 5.1 видно, що при збільшенні k_q відсоток покращення жадібним алгоритмом початкового розв'язку, який складається лише з обов'язкових завдань, стрімко спадає та на відмітці $k_q = 0.26$ стає майже нульовим.

Результати випробувань алгоритмів локального пошуку наведено у таблиці 5.2.

Таблиця 5.2 – Результати випробувань алгоритмів локального пошуку для змінної кількості обов'язкових завдань.

k_q	ДЛП		АІВ		G-алгоритм	
	$\Delta ДЛП, \%$	$t_{сер}, сек.$	$\Delta АІВ, \%$	$t_{сер}, сек.$	$\Delta G, \%$	$t_{сер}, сек.$
0.01	87.23%	4.03	89.46%	15.6	89.36%	15.5
0.02	63.05%	4.4	64.60%	16.6	63.30%	16.4
0.03	61.94%	3.8	64.30%	14.6	63.95%	15.5
0.04	65.40%	3.4	67.64%	15.3	66.44%	16
0.05	55.96%	3.3	57.80%	15	57.53%	15.3
0.06	56.86%	3.4	59.01%	17.6	57.30%	18.1
0.07	48.37%	3	50.65%	17.4	50.42%	18.1
0.08	42.29%	2.9	45.22%	18	43.57%	18.5
0.09	36.67%	3.2	38.04%	18.1	37.90%	19.3
0.1	28.98%	2.6	31.74%	16.3	30.09%	16.7
0.11	30.60%	2.9	32.50%	17.2	32.37%	17.7
0.12	27.82%	2.6	30.70%	19.2	28.36%	21.26
0.13	28.69%	2.5	30.84%	13.4	30.54%	13.4
0.14	21.01%	2.8	22.22%	13.6	21.73%	15.2
0.15	18.95%	2.4	20.30%	10.2	20.08%	10.3
0.16	17.12%	2.6	18.91%	9.6	17.69%	9.8
0.17	13.26%	2.5	14.17%	6.5	13.55%	7.2
0.18	10.37%	2.1	12.15%	4.3	10.76%	5.3
0.19	10.51%	1.9	11.01%	4.5	10.83%	5.2
0.2	9.10%	2.3	9.61%	4	9.27%	4.6
0.21	6.33%	1.5	6.68%	3.5	6.33%	3.6
0.22	6.12%	1.7	6.51%	3.1	6.20%	3.4
0.23	5.02%	1.7	6.09%	2.8	5.88%	3
0.24	6.54%	1.5	6.80%	3.1	6.56%	3.1
0.25	4.88%	1.2	4.96%	2.5	4.93%	2.7
0.26	2.56%	1	2.55%	2.1	2.55%	2.2
0.27	2.61%	1.5	2.68%	2.3	2.64%	2.5
0.28	3.10%	1.3	3.13%	2.8	3.13%	2.8
0.29	2.81%	1	2.82%	2.1	2.79%	2.4
0.3	2.78%	1.4	2.77%	2.5	2.74%	2.6

У додатку 7 наведено графічне зображення результатів роботи всіх розроблених для другої підзадачі алгоритмів локального пошуку зі змінною часткою обов'язкових завдань.

З таблиці 5.2 видно, що при збільшенні k_q відсоток покращення алгоритмами локального пошуку розв'язку, отриманого жадібним алгоритмом, стрімко спадає та на відмітці $k_q = 0.28$ стає майже нульовим. Найбільша швидкість спаду спостерігається на проміжку $k_q \in [0.02, 0.1]$, після цього швидкість спаду покращення значення цільової функції зменшується. Це пояснюється тим, що обов'язкові завдання займають все більшу частку часу, який мають виконавці, і через це кількість доданих необов'язкових завдань зменшується.

Загалом можна зробити висновок, що співвідношення між кількістю обов'язкових та необов'язкових завдань критично впливає на результати роботи як алгоритмів локального пошуку, так і жадібного алгоритму.

5.3 Кількість впорядкованих завдань

Для оцінки впливу кількості впорядкованих завдань на ефективність роботи кожного з запропонованих алгоритмів проведено серію експериментів зі змінною кількістю впорядкованих завдань, інші змінні параметри зафіксовано.

Кількість обов'язкових завдань:

$$q = 0.1 \cdot n.$$

Нехай кількість впорядкованих необов'язкових завдань обчислюється таким чином:

$$p = k_p \cdot (n - q),$$

де $k_p \in [0, 1]$.

Для проведення експериментів k_p змінювалося в такому інтервалі:

$$k_p \in [0.02, 0.04, \dots, 0.6].$$

У таблиці 5.3 наведено результати випробувань для жадібного алгоритму зі змінною часткою впорядкованих необов'язкових завдань.

Таблиця 5.3 – Результати випробувань жадібного алгоритму для змінної частки впорядкованих необов'язкових завдань

k_q	$\Delta ЖАД$, %	$t_{сер}$, мс.
0.02	69.65	6.2
0.04	70.99	8.3
0.06	66.38	4.4
0.08	70.3	5.7
0.1	67.43	7.3
0.12	69.8	5.1
0.14	64.89	8.2
0.16	67.4	9.6
0.18	65.84	6.7
0.2	69.18	6.8
0.22	65.28	7.2
0.24	66.68	5.4
0.26	68.23	5.4
0.28	63.9	5.5
0.3	69.06	6.7
0.32	63.7	7.2
0.34	62.57	7.1
0.36	63.47	5.8
0.38	65.18	7.4
0.4	62.69	6.7
0.42	61.3	7.2
0.44	64.21	5.9
0.46	62.74	6.6
0.48	61	7.4
0.5	63.07	6.5
0.52	63.8	6.5
0.54	61.36	6.2
0.56	61.79	5.8
0.58	62.44	6.3
0.6	63.17	6.8

З таблиці 5.3 видно, що при збільшенні k_p відсоток покращення жадібним алгоритмом початкового розв'язку, який складається лише з обов'язкових завдань, повільно спадає, з 70-72% збільшення значення цільової функції на відмітці $k_p = 0.4$ та більше досягає 60-63%.

У додатку 6 наведено графічне зображення результатів роботи жадібного алгоритму зі зміною частки впорядкованих необов'язкових завдань.

Результати випробувань для алгоритмів локального пошуку наведено у таблиці 5.4.

Таблиця 5.4 – Результати випробувань алгоритмів локального пошуку для змінної кількості впорядкованих необов'язкових завдань

k_q	ДЛП		АІВ		G-алгоритм	
	$\Delta ДЛП$, %	$t_{сер}$, сек.	$\Delta АІВ$, %	$t_{сер}$, сек.	ΔG , %	$t_{сер}$, сек.
0.02	33	2.9	34.42	18.1	33.23	19.8
0.04	32.86	3.4	34.49	18.6	34.01	20.8
0.06	33.45	3.1	35.21	18.2	34.42	19.7
0.08	33.43	3.3	34.5	19.1	33.85	20.9
0.1	33.55	2.7	35.18	18.4	34.37	19.3
0.12	32.56	3.3	34.36	19.6	33.33	20.1
0.14	32.45	3.1	34	18.9	33.4	20
0.16	32.65	3.8	33.82	19.7	32.81	23
0.18	31.96	2.8	33.91	19.8	32.68	20.1
0.2	31.92	5	33.31	21.4	33.2	22.6
0.22	32.36	2.7	34.29	21.5	33.54	21.6
0.24	32.5	2.8	34.32	20.9	33.95	22.8
0.26	32.88	4.9	34.3	22.7	32.98	24
0.28	32.02	2.8	34.24	22	34.06	22.3
0.3	32.54	5.3	34.29	24	33.09	27.1
0.32	32.29	2.9	34.19	21.6	33.95	22.5
0.34	32.82	3.1	34.18	19.5	33.88	20
0.36	31.38	3.3	33.93	23.1	32.96	23.5
0.38	30.4	3.4	32.18	21.5	30.96	21.2
0.4	30.05	2.5	32.4	16.9	31.31	18.2
0.42	30.37	2.8	32.13	18.8	31.29	19.3
0.44	30.22	2.8	31.47	19.3	31.32	19.5
0.46	30.15	2.6	31.41	14.6	30.3	14.7
0.48	28.68	2.6	30.5	13.6	29.66	13.7
0.5	28.22	2.6	29.97	16.5	28.61	18.7

k_q	ДЛП		АІВ		G-алгоритм	
	$\Delta ДЛП, \%$	$t_{сер}, сек.$	$\Delta АІВ, \%$	$t_{сер}, сек.$	$\Delta G, \%$	$t_{сер}, сек.$
0.52	28.79	2.6	30.1	20.5	28.83	21.8
0.54	28.49	2.2	29.64	18.5	28.43	18.2
0.56	27.79	2.7	29.81	14.5	28.64	14.6
0.58	28.28	2.7	29.24	18.3	28.91	19.7
0.6	27.67	2.6	29.15	17.9	28.72	21

З таблиці 5.4 видно, що зі збільшенням кількості впорядкованих завдань значення цільової функції кращого розв'язку для всіх трьох алгоритмів повільно знижується. Для всіх алгоритмів відсоток збільшення значення цільової функції початкового розв'язку з 33-35% зменшується до 28-29%.

Графічне зображення результатів експериментів для алгоритмів локального пошуку зі змінною часткою впорядкованих необов'язкових завдань наведено у додатку 7.

Загалом можна зробити висновок, що кількість впорядкованих необов'язкових завдань помірно впливає на результати роботи алгоритмів локального пошуку.

5.4 Розкид тривалості завдань

Перевіримо, чи впливає інтервал, в якому знаходиться час виконання деякого завдання, на отриманні розв'язки задачі. Для простоти експерименту змінюватимемо тільки розкид часу виконання необов'язкових завдань, тоді як для обов'язкових завдань цей інтервал буде фіксованим. Час виконання кожного обов'язкового завдання деяким виконавцем визначався в інтервалі:

$$t_{\min} = \frac{d}{100} \text{ одиниць часу,}$$

$$t_{\max} = \frac{d}{2} \text{ одиниць часу.}$$

Тоді час виконання деякого обов'язкового завдання $j, j \in Q$ виконавцем $i \in \{1, 2, \dots, m\}$ визначається так:

$$t_{ij} = \text{random} [t_{\min}, t_{\max}] \text{ одиниць часу.}$$

Зафіксуємо кількість обов'язкових завдань:

$$q = 0.1 \cdot n.$$

Нехай максимальний час на виконання необов'язкового завдання обчислюється таким чином:

$$t_{\max}^p = k_i \cdot d,$$

де

$$k_i \in [0, 1].$$

Для проведення експериментів k_i змінювалося в такому інтервалі:

$$k_i \in [0.05, 0.1, \dots, 1].$$

Змінюватимемо час на виконання необов'язкових завдань в інтервалі.
 $[t_{\min}^p, t_{\max}^p]$

$$t_{\min}^p = \frac{d}{100} \text{ одиниць часу.}$$

У таблиці 5.5 наведено результати випробувань для жадібного алгоритму зі змінним максимальним часом на виконання необов'язкового завдання.

Таблиця 5.5 – Результати випробувань жадібного алгоритму для змінного максимального часу на виконання необов’язкового завдання

k_q	$\Delta ЖАД, \%$	$t_{сер}, мс.$
0.1	229.87	8.2
0.15	135.39	7.3
0.2	137.02	6.8
0.25	114.65	7.1
0.3	114.66	5.6
0.35	86.23	5.5
0.4	82.36	8.1
0.45	66.2	8.7
0.5	68.73	6.2
0.55	72.23	6.9
0.6	79.43	7.2
0.65	51	5.3
0.7	59	5.7
0.75	67.53	5.8
0.8	42.39	6.1
0.85	50.91	7.5
0.9	65	6.4
0.95	58.93	6.1
1	43.37	5.9

Як видно з таблиці 5.5, найбільше покращення розв’язку жадібним алгоритмом досягається при діапазоні значень для часу виконання завдання в межах 40% від директивного терміну. Починаючи зі значення 20% від директивного терміну спостерігаємо спад різниці в значеннях цільової функції до 50% та менше. Найстрімкіший спад спостерігається при значеннях $k_i \in [0.1, 0.4]$, після цього спад стає повільнішим та менш помітним.

У додатку 6 наведено графічне зображення результатів роботи жадібного алгоритму зі зміною частки впорядкованих необов’язкових завдань.

Результати випробувань алгоритмів локального пошуку наведено у таблиці 5.6.

Таблиця 5.6 – Результати випробувань алгоритмів локального пошуку для змінного максимального часу на виконання необов’язкового завдання

k_q	ДЛП		АІВ		G-алгоритм	
	$\Delta ДЛП, \%$	$t_{сер}, \text{сек.}$	$\Delta АІВ, \%$	$t_{сер}, \text{сек.}$	$\Delta G, \%$	$t_{сер}, \text{сек.}$
0.1	48.64	7.65	48.78	60.2	48.69	64.1
0.15	48.1	7.7	50.04	36.1	49.4	36.9
0.2	43.23	7.7	46.7	30.78	44.7	32.1
0.25	39.52	6.3	43.1	30.3	40.85	30
0.3	39.61	5.1	42.73	24.7	41.64	25.2
0.35	41.65	6.2	42.92	27.6	41.95	27
0.4	38.26	4.3	40.6	24.5	40.16	25.3
0.45	37.4	4.01	38.61	20.8	37.2	22.1
0.5	34.52	4.6	35.55	1.35	34.96	20.6
0.55	29.85	4.8	32.61	1.1	30.96	20.5
0.6	26.78	4	27.56	1.09	27.35	20.4
0.65	26.01	3.1	27.02	0.6	26.05	16.4
0.7	27.31	4	28.11	1.2	27.97	21
0.75	25.79	3.8	27.43	1.1	26.95	20.7
0.8	23.01	3.5	25.2	0.68	24.18	17
0.85	22.13	4	23.41	0.67	22.98	16.8
0.9	23.3	4	24.96	1	24.32	20.6
0.95	23.36	4.5	23.53	0.67	23.53	16
1	22.43	3.13	23.31	0.43	22.16	9.5

Як видно з таблиці 5.6, зі збільшенням верхньої межі часу виконання завдання отримані результати поступово погіршуються – від майже 50% на початку покращення початкового розв’язку алгоритмами локального пошуку знижується до 23-25%. Це пояснюється тим, що чим меншою є кількість часу, яку необхідно витратити на одне завдання, тим більше завдань потрапить до кінцевого розв’язку та, як наслідок, тим більшим буде значення цільової функції. Найстрімкіший спад спостерігається при значеннях $k_i \in [0.1, 0.6]$, після цього спад стає повільнішим та менш помітним.

Графічне зображення результатів експериментів для алгоритмів локального пошуку зі змінною часткою впорядкованих необов’язкових завдань наведено у додатку 7.

Висновок до розділу

Для перевірки ефективності алгоритмів проведено серію експериментів зі змінними параметрами екземплярів задач, що розв'язуються. Параметри, які розглянуто – відсоток обов'язкових завдань, відсоток впорядкованих необов'язкових завдань та розкид часу на виконання необов'язкових завдань.

На підставі проведених експериментів можна зробити наступні висновки:

- відсоток обов'язкових завдань критично впливає на ефективність розроблених алгоритмів, як жадібного, так і алгоритмів локального пошуку. Зі збільшенням кількості обов'язкових завдань їх ефективність швидко знижується. Це обумовлене тим, що обов'язкові завдання займають все більше часу виконавців та поступово для необов'язкових завдань лишається все менше і менше часу;

- відсоток впорядкованих необов'язкових завдань не має серйозного впливу на ефективність розроблених алгоритмів. Зменшення покращення початкового розв'язку є незначним та повільним порівняно з кількістю обов'язкових завдань як для жадібного алгоритму, так і для алгоритмів локального пошуку;

- розкид часу на виконання необов'язкових завдань є важливим параметром задачі, який впливає на ефективність розроблених алгоритмів. Зі збільшенням максимального часу на виконання необов'язкових завдань ефективність жадібного алгоритму та алгоритмів локального пошуку поступово зменшується, досягаючи відмітки у 20% на максимальному часі зі значенням директивного терміну.

ВИСНОВКИ

В роботі досліджено проблему оптимального планування виконання завдань виконавцями з різною продуктивністю. Особливості проблеми, яку розглянуто, включають в себе різну непропорційну продуктивність виконавців, наявність загального директивного терміну для виконання завдань, наявність обов'язкових та необов'язкових для виконання завдань та наявність часткової впорядкованості завдань.

Проведено аналіз наукової літератури, присвяченої цій проблемі, та виявлено, що розробка нових алгоритмів її розв'язування є актуальною через велику розмірність задач та наявність особливостей проблеми, вказаних вище.

Сформульовано змістовну постановку проблеми та розроблено її математичну модель. Запропоновано декомпонувати розроблену математичну модель на дві підзадачі. Запропоновано алгоритм розв'язування поставленої задачі шляхом послідовного розв'язування отриманих підзадач.

Перша підзадача є класичною узагальненою задачею про призначення, для неї вже розроблено велику кількість алгоритмів розв'язування. Для другої підзадачі, яка містить в собі особливості задач складання розкладів та задачі про пакування рюкзака, вперше розроблено чотири алгоритми, які належать до схем локального пошуку та жадібних алгоритмів.

Здійснено програмну реалізацію розроблених алгоритмів у вигляді спеціалізованого програмного забезпечення яке може бути використане для розв'язування сформульованої задачі.

Дослідження ефективності алгоритмів здійснено на основі обчислювального експерименту, результати наступні: екземпляри поставленої задачі мають параметри, які впливають на можливість знаходження задовільних розв'язків. Такими параметрами є кількість обов'язкових завдань та розкид часу на виконання необов'язкових завдань. При збільшенні цих параметрів ефективність всіх алгоритмів розв'язування другої підзадачі знижується. Це обумовлене обмеженням за часом, оскільки в першому випадку обов'язкові завдання займають все більше

часу, дозволяючи додати все меншу кількість необов'язкових завдань для збільшення значення цільової функції, а в другому – самі необов'язкові завдання займають все більше часу.

Розроблені алгоритми та програмні засоби можуть бути застосовані для задач оптимального планування в інших сферах. Напрямком подальших досліджень може бути розробка та дослідження ефективності алгоритмів для розв'язування сформульованої задачі та отриманих з неї нових підзадач.

ПЕРЕЛІК ПОСИЛАНЬ

1. Галкіна Г. А. Задача оптимального планування робіт за наявності різної продуктивності виконавців / Г. А. Галкіна, Л. Ф. Гуляницький // Науковий огляд. – 2018. - №3. – с. 66-80.
2. Галкіна Г. А. Алгоритми локального пошуку для однієї задачі планування робіт / Г. А. Галкіна // Науковий огляд. – 2018. - №3. – с. 47-65.
3. Посібник зі Скраму [Електронний ресурс] / К. Швабер, Д. Сазерленд – Режим доступу до ресурсу: <https://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-UA.pdf>
4. Rubin K. Essential Scrum / Kenneth Rubin. – Boston : Addison-Wesley, 2013. – 504 p.
5. Top 4 Challenges for Agile Planning [Електронний ресурс] – Режим доступу до ресурсу: <https://www.telerik.com/blogs/top-4-challenges-for-agile-planning>
6. Inside Atlassian: four steps to better Sprint planning [Електронний ресурс] – Режим доступу до ресурсу: <https://www.atlassian.com/blog/agile/sprint-planning-atlassian>
7. Golfarelli M. Sprint Planning Optimization in Agile Data Warehouse Design / M. Golfarelli, S. Rizzi, E. Turricchia // 14th International Conference, DaWaK 2012, Vienna, Austria, September 3-6, 2012. Proceedings. – P.30-41.
8. Kimball R. The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling / R. Kimball, M. Ross. – New Jersey : Wiley, 2002. – 464 p.
9. Martello S. Knapsack Problems. Algorithms and Computer Implementations / S. Martello, P. Toth. – New Jersey : John Wiley & Sons, 1990. – 296 p.
10. IBM CPLEX Optimization Studio [Електронний ресурс] – Режим доступу до ресурсу: <https://www.ibm.com/products/ilog-cplex-optimization-studio>
11. Комбінаторні методи. Метод гілок та меж [Електронний ресурс] – Режим доступу до ресурсу: <http://ecolib.com.ua/article.php?book=33&article=4095>

12. Метод Гоморі (метод відсікаючих площин) [Електронний ресурс] – Режим доступу до ресурсу: <http://www.mathros.net.ua/metod-gomori-metod-vidsikajuchyh-ploshhyn.html>
13. Jansi S. A Greedy Heuristic Approach for Sprint Planning in Agile Software Planning / S. Jansi, K. C. Rajeswari // International Journal for Trends in Engineering & Technology. – 2015. – №1. – P.18-21.
14. Карагодова О.О. Дослідження операцій: Навч. посібник / О. О. Карагодова, В. Р. Кігель, В. Д. Рожок. — К.: Центр учбової літератури, 2007 — 256 с.
15. Sobiech F. A Heuristic Approach to Solve the Elementary Sprint Optimization Problem for Non-Cross-Functional Teams in Scrum / F. Sobiech, B. Eilermann, A. Rausch // Applied Computing Review. – 2014. – №4. – P.19-25.
16. Boschetti M. A. A Lagrangian heuristic for sprint planning in agile software development / M. A. Boschetti, M. Golfarelli, S. Rizzi, E. Turrichia // Computers and Operations Research. – 2014. – №43. – P.116-128.
17. Hung S. An algorithm for 0-1 multiple-knapsack problems / M. S. Hung, J. C. Fisk // Naval Research Logistics. – 1978. – №3. – P.571-579.
18. Lalami M. A procedure-based heuristic for 0-1 Multiple Knapsack Problems / M. E. Lalami, M. Elkihel, D. E. Baz, V. Boyer // Int. J. Mathematics in Operational Research. – 2012. – №4. – P.214-224.
19. Pisinger D. An Exact Algorithm for Large Multiple Knapsack Problems / D. Pisinger // European Journal of Operational Research. – 1999. – №3. – P.528-541.
20. Dahl G. LP based heuristics for the multiple knapsack problem with assignment restrictions / G. Dahl, N. Foldnes // Annals of Operations Research. – 2006. – №1. - P.91-104.
21. Kolliopoulos S. G. Partially ordered knapsack and applications to scheduling / S. G. Kolliopoulos, G. Steiner // Discrete Applied Mathematics. – 2007, vol. 155. – №8. – P. 889-897.

22. Dawande M. Approximation Algorithms for the Multiple Knapsack Problem with Assignment Restrictions / M. Dawande, J. Kalagnanam, P. Keskinocak, F. S. Salman, R. Ravi // *Journal of Combinatorial Optimization*. – 2000, vol. 4. – №2. – P. 171-186.
23. Kellerer H. Knapsack Problems / H. Kellerer, U. Pferschy, D. Pisinger. – Berlin: Springer, 2004. – 548 p.
24. Shah-Hosseini H. Intelligent water drops algorithm: A new optimization method for solving the multiple knapsack problem / H. Shah-Hosseini // *International Journal of Intelligent Computing and Cybernetics* – 2008, vol. 1. – №2. – P. 193-212.
25. Kuhn H. The Hungarian Method for the Assignment Problem / H. Kuhn // *Naval Research Logistics Quarterly*. – 1955. – №1-2. – P.83–97.
26. Задача про призначення [Електронний ресурс] – Режим доступу до ресурсу: <https://studfiles.net/preview/5163126/page:5/>
27. Akbulut M. A Modified Genetic Algorithm for the Generalized Assignment Problems / M. B. Akbulut, A. E. Yilmaz // *Journal of Electrical & Electronics Engineering*. – 2009. – №2. – P.951-958.
28. Woodcock A. A hybrid tabu search/branch & bound approach to solving the generalized assignment problem / A. J. Woodcock, J. M. Wilson // *European Journal of Operational Research*. – 2010. – №2. – P.566-578.
29. Laguna M. Tabu search for multilevel generalized assignment problem / M. Laguna, J. P. Kelly, J. L. Gonzalez-Velarde, F. Glover // *European Journal of Operational Research*. – 1995. – №82. – P.176-189.
30. Wu T.-H. A tabu search approach to the generalized assignment problem / T.-H. Wu, J.-Y. Yeh, Y.-R. Syau // *Journal of the Chinese Institute of Industrial Engineers*. – 2003. – №3. – P.301-311.
31. Lourenco H. R. Adaptive Search Heuristics for the Generalized Assignment Problem // H. R. Lourenco, D. Serra – *Mathware & Soft Computing*. – 2000. – №7. – P. 1-15.
32. Feo T. A. Greedy randomized adaptive search heuristic // T. A. Feo, M. G. C. Resende // *Journal of Global Optimization*. – 1995, vol. 6. – P. 109-133.

33. Stutzle T. Max-Min Ant System and Local Search for Combinatorial Optimization // T. Stutzle, H. Hoos – Meta-Heuristics: Trends in Local Search paradigms for Optimization. – Kluwer Academic Publishers. – P. 313-329.
34. Yagiura M. An ejection chain approach for the generalized assignment problem / M. Yagiura, T. Ibaraki, F. Glover // INFORMS Journal on Computing. – 2004, vol. 16. - №2. – P. 133-151.
35. Terry Ross G. A branch and bound algorithm for the generalized assignment problem / G. Terry Ross, R. M. Soland // Mathematical Programming. – 1975, vol. 8. - №1. – P. 91-103.
36. Ozbakir L. Bees algorithm for generalized assignment problem / L. Ozbakir, A. Baykasoglu, P. Tapkan // Applied Mathematics and Computation. – 2010, vol. 215. – №11. – P. 3782-3795.
37. Randall M. Heuristics for ant colony optimisation using the generalized assignment problem / M. Randall // Evolutionary Computation, 2004, Portland, OR, USA, June 19-23, 2004. Proceedings. – P.110-116.
38. Guignard M. Technical Note – An Improved Dual Based Algorithm for the Generalized Assignment Problem / M. Guignard, M. B. Rosenwein // Operations Research. – 1989, vol. 37. - №4. – P. 658-663.
39. Brucker P. Scheduling Algorithms / P. Brucker. – New York : Springer Publishing, 2007. – 371 p.
40. Gillies D.W. Scheduling Tasks with AND/OR Precedence Constraints / D.W.Gillies, J.W.-S. Liu. // SIAM Journal on Computing. – 1995. – №4. – P.797-810.
41. De P. Due-date assignment and early/tardy scheduling on identical parallel machines / P. De, J. B. Ghosh, C. E. Wells // Naval Research Logistics. – 1994. – №1. – P.17-32.
42. Nikabadi M. A hybrid algorithm for unrelated parallel machines scheduling / M. S. Nikabadi, R. Naderi // International Journal of Industrial Engineering Computations. – 2016. – №7. – P.681-702.

43. Raja K. A Genetic Approach for Scheduling Independent Jobs on Uniform Parallel Machines / K. Raja, V. Selladurai, R. Saravanan // Manufacturing and Industrial Engineering. – 2007. – №4. – P.12-16.

44. A Unified Approach to Scheduling on Unrelated Parallel Machines [Электронный ресурс] / V. S. Anil Kumar, M. V. Marathe, S. Parthasarathy, A. Srinivasan – Режим доступа до ресурсу: <http://www.cs.umd.edu/~srin/PDF/2009/upm-jou.pdf>

45. Scheduling on Unrelated Machines under Tree-Like Precedence Constraints [Электронный ресурс] / V. S. Anil Kumar, M. V. Marathe, S. Parthasarathy, A. Srinivasan – Режим доступа до ресурсу: <http://www.cs.umd.edu/~srin/PDF/2007/treesched-jou.pdf>

46. Chekuri C. Precedence constrained scheduling to minimize sum of weighted completion times on a single machine / C. Chekuri, R. Motwani // Discrete Applied Mathematics. – 1998. – №98. – P. 29-38.

47. Pizaruk N. N. A fully combinatorial 2-approximation algorithm for precedence-constrained scheduling a single machine to minimize average weighted completion time / N. N. Pizaruk // Discrete Applied Mathematics. – 2003. – №131. – P. 655-663.

48. Pizaruk N. N. The boundaries of submodular functions / N. N. Pizaruk // Computational Mathematics and Mathematical Physics. – 1992. – №32. – pp. 1769-1783.

49. Correa J. R. Single-Machine Scheduling with Precedence Constraints / J. R. Correa, A. S. Schulz // Mathematics of Operations Research. – 2005, vol.30. – №4. – P. 1005-1021.

50. Sidney J. B. Decomposition algorithms for single machine scheduling with precedence relations and deferral costs / J. B. Sidney // Operations research. – №23. – P. 283-298.

51. Scheduling on Unrelated Parallel Machines [Электронный ресурс] – Режим доступа до ресурсу: <https://www.corelab.ntua.gr/courses/netalg/presentations/presentations0708/Karakatsanis%20Scheduling%20on%20Unrelated%20Parallel%20Machines.pdf>

52. Hamad A. A neural network or common due date job scheduling problem on parallel unrelated machines / A. Hamad, B. Sanugi, S. Salleh // *Matematika*. – 2001. - №2. – P. 63-70.
53. Piersma N. A local search heuristic for unrelated parallel machine scheduling with efficient neighborhood search / N. Piersma, W. van Dijk // *Mathematical and Computer Modeling*. – 1996, vol. 4. - №9. – P. 11-19.
54. Ghirardi M. Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach / M. Ghirardi, C. N. Potts // *European Journal of Operational Research*. – 2005, vol. 165. – №2. – P.457-467.
55. Della Croce F. Recovering Beam Search: Enhancing the Beam Search Approach for Combinatorial Optimization Problems / F. Della Croce, M. Ghirardi, R. Tadei // *Journal of Heuristics*. – 2004, vol. 10. – №1. – P. 89-104.
56. Better Unrelated Machine Scheduling for Weighted Completion Time via Random Offsets from Non-Uniform Distributions [Электронный ресурс] / I. Sungjin, S. Li – Режим доступа до ресурсу: <https://www.cse.buffalo.edu/~shil/papers/rrjwc-FOCS2016.pdf>
57. Unrelated Machine Scheduling of Jobs with Uniform Smith Ratios [Электронный ресурс] / C. Kalaitzis, O. Svensson, J. Tarnawski – Режим доступа до ресурсу: <https://arxiv.org/pdf/1607.07631.pdf>
58. Lenstra J. K. Approximation Algorithms for Scheduling Unrelated Parallel Machines / J. K. Lenstra, D. B. Shmoys, E. Tardos // *Mathematical programming*. – 1990, vol. 46. - №1-3. – pp. 259-271.
59. Scheduling on Unrelated Parallel Machines [Электронный ресурс] – Режим доступа до ресурсу: https://courses.engr.illinois.edu/cs598csc/sp2011/Lectures/lectures_10-11.pdf
60. Schulz A. S. Scheduling Unrelated Machines by Randomized Rounding / A. S. Schulz, M. Skutella // *SIAM Journal. Discrete Math*. – 2002. – №4. – P.450-469.
61. Kim D.-W. Unrelated parallel machine scheduling with setup times using simulated annealing / D.-W. Kim, K.-H. Kim, F. Chen // *Robotics and Computer-Integrated Manufacturing*. – 2002. – №3-4. – P.223-231.

62. Weng M. X. Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective / M. X. Weng, J. Lu, H. Ren // *International Journal of Production Economics*. – 2001. – №3. – P.215-226.
63. Potts C. N. Analysis of a linear programming heuristic for scheduling unrelated parallel machines / C. N. Potts // *Discrete Applied Mathematics*. – 1985. – №2. – P.155-164.
64. Bank J. Heuristic algorithms for unrelated parallel machine scheduling with a common due date, release dates, and linear earliness and tardiness penalties / J. Bank, F. Werner // *Mathematical and Computer Modeling*. – 2001. – №4-5. – P.363-383.
65. Vallada E. A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times / E. Vallada, R. Ruiz // *European Journal of Operational Research*. – 2011. – №3. – P.612-622.
66. What is the right size for a user story? [Электронный ресурс] / А. Kelly – Режим доступа до ресурсу: <https://dzone.com/articles/what-right-size-user-story>
67. What is the optimal story size? [Электронный ресурс] – Режим доступа до ресурсу: <http://www.scrum-breakfast.com/2011/03/what-is-optimal-story-size.html>
68. Yagiura M. The Generalized Assignment Problem and Its Generalizations [Электронный ресурс] / М. Yagiura, Т. Ibaraki – Режим доступа до ресурсу: <http://leeds-faculty.colorado.edu/glover/TS%20-%20vignettes%20-%20GAP%20Yagiura%20&%20Ibaraki.pdf>
69. Martello S. Generalized assignment problems / S. Martello, P. Toth // *Third International Symposium.*, ISAAC'92 Nagoya, Japan, December 16-18, 1992 Proceedings – P.351-169.
70. Cohen R. An efficient approximation for the Generalized Assignment Problem / R. Cohen, L. Katzir, D. Raz // *Information Processing Letters*. – 2006, vol. 100. – №4. – P.162-166.
71. Korte B. *Combinatorial Optimization* / B. Korte, J. Vygen. – New York : Springer Publishing, 1997. – 627 p.
72. Кормен Т. Алгоритмы. Построение и анализ / Т. Кормен, Ч.И. Лейзерсон, Р.Л. Ривест, К. Штайн. – М.: МНЦМО, 2002. – 960 с.

73. Гуляницький Л.Ф. Прикладні методи комбінаторної оптимізації: навч. посіб. / Л. Ф. Гуляницький, О. Ю. Мулеса. – К. : Видавничо-поліграфічний центр “Київський університет”, 2016. – 142 с.

74. Gosling J. The Java Language Specification: Second Edition / J. Gosling, B. Joy, G. Steele, G. Bracha // Boston: Addison-Wesley Professional, 200. – 505 p.

75. About the Java Technology [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html>

76. Hello World [Електронний ресурс] – Режим доступу до ресурсу: <https://blog.jetbrains.com/kotlin/2011/07/hello-world-2/>

77. Kotlin [Електронний ресурси] – Режим доступу до ресурсу: <https://kotlinlang.org/>

78. Vasić M. Fundamental Kotlin [Електронний ресурс] / M. Vasić – Режим доступу до ресурсу: <http://www.fundamental-kotlin.com/>

79. Kotlin on Android. Now official [Електронний ресурс] – Режим доступу до ресурсу: <https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/>

80. Jemerov D. Kotlin in Action / D. Jemerov, S. Isakova. – Shelter Island : Manning Publications, 2017. – 360 p.

81. Хорстманн К. Java. Библиотека профессионала, том 1. Основы. 9е изд. : Пер. с англ. / К. Хорстманн, Г. Корнелл – М. : ООО “И.Д. Вильямс”, 2015. – 864 с.

82. Эккель Б. Философия Java. 4-е полное изд / Б. Эккель – СПб.: Питер, 2017. – 1168 с.

ПЛАКАТ 1 БЛОК-СХЕМИ АЛГОРИТМІВ ЛОКАЛЬНОГО ПОШУКУ

**ПЛАКАТ 2 БЛОК-СХЕМА ПРОЦЕДУРИ ГЕНЕРАЦІЇ РОЗВ'ЯЗКУ З ОКОЛУ З
ВИКОРИСТАННЯМ ЗСУВІВ ВВЕРХ ТА ВНИЗ**

ПЛАКАТ 3 СХЕМА СТРУКТУРНА ПОСЛІДОВНОСТІ

ΠΛΑΚΑΤ 4 ΣΧΕΜΑ ΣΤΡΥΚΤΥΡΗΑ ΚΛΙΣΙΒ

ПЛАКАТ 5 СХЕМА СТРУКТУРНА ПΑΚΕΤΙΒ

**ПЛАКАТ 6 РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТІВ ДЛЯ ЖАДІБНОГО
АЛГОРИТМУ**

**ПЛАКАТ 7 РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТІВ ДЛЯ АЛГОРИТМІВ
ЛОКАЛЬНОГО ПОШУКУ**