Дніпровський національний університет імені Олеся Гончара
Міністерство освіти і науки України
Дніпровський національний університет імені Олеся Гончара
Міністерство освіти і науки України

Кваліфікаційна наукова
праця на правах рукопису

Караваєв Костянтин Дмитрович

УДК 519.8:519.176

ДИСЕРТАЦІЯ

МЕТОДИ І АЛГОРИТМИ РОЗВ'ЯЗАННЯ КЛАСИЧНИХ ТА УЗАГАЛЬНЕНИХ ЗАДАЧ УПОРЯДКУВАННЯ ВЕРШИН ОРГРАФІВ

11 – Математика та статистика
113 – Прикладна математика

Подається на здобуття ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ К.Д. Караваєв

Науковий керівник
Турчина Валентина Андріївна
кандидат фізико-математичних наук,
доцент

Дніпро – 2024

АНОТАЦІЯ

Караваєв К.Д. Методи і алгоритми розв’язання класичних та узагальнених задач упорядкування вершин орграфів. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття ступеня доктора філософії за спеціальністю 113 «Прикладна математика» (11 – Математика та статистика). – Дніпровський національний університет імені Олеся Гончара, Дніпро, 2024.

Дисертаційна робота присвячена розробці та обґрунтуванню точних та наближених методів і алгоритмів розв’язання класичних та узагальнених задач паралельного упорядкування вершин ациклічних орієнтованих графів.

Математична теорія паралельного упорядкування за більш ніж 60 років свого розвитку стала одним з потужних інструментів розв’язання багатьох теоретичних та практичних задач, що можуть бути зведеними до оптимізаційних задач на графах. Вона знайшла своє місце у різноманітних практично важливих галузях: при плануванні схем та розкладів виробництва, побудові мікросхем, багатопотоковій обробці даних, паралелізації та розподіленні обчислень та в інших сферах, де на порядок виконання завдань накладається деяка множина несуперечливих технологічних обмежень. У загальному випадку більшість таких задач відноситься до класу NP-важких, тому важливим є як пошук підкласів задач, для яких існують точні поліноміальні алгоритми, так і побудова наближених алгоритмів з прийнятним рівнем похибки отриманих розв’язків.

Враховуючи експоненційну складність задачі, серед методів пошуку точних розв’язків задач широкого поширення набули схеми напрямленого перебору, зокрема метод гілок та меж. До його переваг можна віднести наочність та простоту реалізації.

Найбільшого розповсюдження набули точні поліноміальні алгоритми та методи, запропоновані М. Гері, Д. Джонсоном, Т.С. Ху, М. Фуджі, Дж.К. Ленстрою, Е.Г. Кофманом, Р.Л. Грехемом. Серед них особливе місце посідає рівневий принцип, на якому базується історично перший поліноміальний алгоритм. Цей принцип є

одним з фундаментальних підходів до схем пошуку наближених розв’язків. Також важливе значення мають алгоритми, засновані на максимальному паросполученні та лексикографічному помічені, які є точними у випадку класичної постановки задачі з двома виконавцями. Модифікація цих алгоритмів дозволяє застосовувати їх не лише для класичних випадків.

Точні поліноміальні алгоритми, отримані в подальших дослідженнях, найчастіше були засновані на переборі та мали алгоритмічну складність, показник степіня якої мультиплікативно залежить від ширини упорядкування або довжини критичного шляху. В останні роки основна увага науковців була приділена розробці (1+ε)-точних квазіполіноміальних алгоритмів. Застосування таких алгоритмів на практиці найчастіше є неможливим, з чого випливає важливість отримання наближених алгоритмів прийнятної точності з малою алгоритмічною складністю.

Класична постановка задачі передбачає, що всі виконавці є універсальними, їх кількість є сталою і час виконання кожної роботи є однаковим. Така постановка дала змогу отримати ряд теоретично важливих результатів, проте вона відповідає вузькому колу реальних умов, тому в подальшому були розроблені деякі узагальнення, що детальніше враховують практичні аспекти процесів, які моделюються. До них належать: наявність затримок, різний час виконання робіт, можливість переривання виконання, зазначення проміжків часу, в які має бути завершена кожна або деякі з робіт (директивні терміни); завдання, виконання яких потребує одночасної роботи декількох, можливо конкретних, виконавців; змінна кількість виконавців, різні критерії оптимальності, тощо. Задачі, що враховують перелічені узагальнення, окрім того, що також найчастіше належать класу NP-важких, можуть й зовсім не мати допустимих розв’язків. Це та той факт, що для моделювання реальних процесів зазвичай необхідно враховувати якусь комбінацію з наведених обмежень, підкреслюють важливість побудови ефективних наближених підходів та алгоритмів як універсальних, так і спеціальних для конкретних постановок.

У роботі отримано ряд тверджень, пов’язаних з можливістю зведення задач упорядкування до певних підкласів цих задач. В основу їх доведення покладена ідея

додання до вхідного графу підграфів спеціального вигляду, які за правильно обраної розмірності не вплинуть на довжину оптимального розв'язку. Виявилося, що шукану розмірність цих підграфів можна визначити за допомогою бінарного пошуку.

Особливу увагу в роботі приділено задачам, в яких шукане упорядкування є щільним. Доведено, що алгоритм, заснований на максимальному паросполученні, в принципі, може знайти оптимальний розв'язок таких задач, на відміну від алгоритмів, заснованих на рівневому принципі. Обчислювальний експеримент показав, що алгоритм у класичному вигляді має незадовільну точність, тому для нього розроблено ряд модифікацій, які дозволили значно покращити його показники ефективності.

Були також досліджені необхідні умови існування щільних упорядкувань, пов'язаних з обмеженістю потужності місць та можливістю їх заповнення. Проведені міркування дозволили не тільки об'єднати розглянуті умови в одну, а й отримати нову уточнену оцінку знизу довжини та узагальнення спеціальних упорядкувань $\underline{S}$ та $\overline{S}$, які враховують ширину упорядкування. Більш того, комбінація цих результатів дозволила отримати послідовність оцінок знизу довжини, кожна з яких є, теоретично, точнішою за попередні. Також було запропоновано наближений алгоритм для таких задач, ідея якого полягає в побудові дерева розгалужень у методі гілок та меж лише для найбільш перспективних гілок, відповідно до деякого обраного критерію.

Аналіз випадків перевищення максимально можливої кількості кроків в обчислювальних експериментах з методом гілок та меж показав, що найчастіше це відбувається, оскільки отримуємо велику кількість гілок, які відповідають ізоморфним підграфам. У такому випадку отримання більш точних оцінок не вплине на кількість розгалужень. В зв'язку з цим проведено дослідження можливості визначати та видаляти такі гілки в дереві варіантів у загальному випадку, а також для паралельно-послідовних графів. Додатково також розглянута можливість застосовувати потужні інваріанти для наближеного визначення ізоморфізму.

При розгляді випадків неоптимальності алгоритму, заснованого на рівневому принципі, для задачі зі змінною шириною для бінарних вхідних дерев виділені бажані вимоги, яким має задовольняти алгоритм. Однією з них є незалежність вибору

вершин від місткостей наступних місць в упорядкуванні. Запропоновано алгоритм, що відповідає всім бажаним вимогам, та проведено обчислювальний експеримент для порівняння його з алгоритмом, заснованим на рівневому принципі. Подальший аналіз показав, що побудувати точний алгоритм, що має зазначену властивість неможливо.

Для врахування неповного завантаження виконавців запропоновано нову постановку задачі, в якій виконавці мають визначені «вихідні», під час яких вони не можуть виконувати завдання. Для розв'язання такої задачі запропоновано використати алгоритм, заснований на максимальному паросполученні, в якому з графу досяжності видалялися ребра між вершинами-завданнями одного виконавця. В загальному випадку такий алгоритм виявився лише наближеним, як і алгоритм, заснований на лексикографічному помічені.

Наукова новизна результатів, описаних у дисертаційній роботі, полягає у наступному:

– *вперше* теоретично обґрунтовано можливість зведення будь-якої класичної задачі оптимального упорядкування до задачі із щільним упорядкуванням та шириною заданої парності;

– *дістав подальшого розвитку* класичний алгоритм розв'язання для двох виконавців для побудови щільних упорядкувань;

– удосконалено оцінки знизу довжини упорядкування для класичної задачі паралельного упорядкування;

– *вперше* запропоновано підхід для скорочення перебору у методі гілок та меж за рахунок виключення гілок, що відповідають ізоморфним підграфам: розглянуті точні та наближені варіанти його реалізації;

– *вперше* отримана необхідна умова існування щільних упорядкувань та запропоновані ефективні алгоритми її перевірки для загальної та спеціальної структур графів;

– *вперше* побудовано наближений алгоритм для задач з щільними упорядкуваннями, який заснований на методі гілок та меж з обмеженою глибиною пошуку;

– *вперше* побудовано послідовність оцінок знизу довжини упорядкування, в якій кожна наступна оцінка, теоретично, є точнішою за попередні;

– *дістав подальшого розвитку* спосіб визначення діапазону допустимих місць вершин шляхом врахування ширини упорядкування;

– *вперше* теоретично обґрунтовано можливість зведення задачі зі змінним значенням ширини упорядкування до класичної задачі;

– *отримані нові наукові дані* про поліноміальні наближені алгоритми для розв'язання задачі зі змінною шириною для вхідних бінарних дерев: запропоновано та досліджено online-алгоритм, продемонстровано, що побудова точного online-алгоритму для такого класу задач в принципі неможлива;

– *вперше* розглянуто клас узагальнених задач оптимального упорядкування з неповним завантаженням: показана можливість зведення до інших відомих класів задач паралельного упорядкування, запропоновано наближений алгоритм до розв'язання задач з цього класу;

– створено програмний продукт, використовуючи засоби об'єктно-орієнтованої мови програмування C#;

– проведено обчислювальні експерименти для визначення ефективності запропонованих методів та алгоритмів.

Отримані в результаті дослідження методи й алгоритми можуть бути застосовані до розв'язання прикладних задач, що зводяться в математичній постановці до задач оптимального упорядкування вершин орієнтованих графів. Отримані результати можуть підвищити ефективність побудови розкладів виконання завдань у різних практично важливих сферах та галузях, зокрема у промисловому виробництві, питаннях логістики, пріоритетності використання та розподілу ресурсів, паралелізації та розподілення обчислень, обробки та моніторингу даних в режимі реального часу тощо. Також вони можуть мати важливе значення для подальшого розвитку сучасних технологій, зокрема автоматичного розпаралелення, та слугувати підґрунтям для нових напрямків наукових пошуків за цією тематикою.

Основні результати дисертаційної роботи опубліковано в 13 наукових працях: 3 статті у наукових фахових виданнях України категорії «Б» з фізико-математичних наук, 1 стаття у виданні, що індексується наукометричною базою Scopus, 1 стаття у інших наукових фахових виданнях України з фізико-математичних наук; 8 тез доповідей у збірниках матеріалів міжнародних та регіональних наукових конференцій і семінарів.

Ключові слова: дискретна оптимізація, комбінаторна оптимізація, теорія розкладів, оптимізаційні задачі на графах, задачі паралельного упорядкування, метод гілок та меж, оцінки характеристик упорядкування, перелік без ізоморфізму, щільні упорядкування, рівневий принцип, розмітка графів, задачі з вихідними, оптимальні розбиття.

ABSTRACT

Karavaiev K. Methods and algorithms for solving classical and generalised problems of digraph vertices sequencing. – Qualifying scientific work on manuscript rights.

Dissertation for obtaining the degree of Doctor of Philosophy in the specialty 113 «Applied Mathematics» (11 – Mathematics and Statistics). – Oles Honchar Dnipro National University, Dnipro, 2024.

The thesis is devoted to the development and justification of exact and approximate methods and algorithms for solving classical and generalised problems of parallel sequencing of acyclic directed graph vertices.

The mathematical theory of parallel sequencing for more than 60 years of its development has become one of the powerful tools for solving many theoretical and practical problems that can be represented as optimization problems on graphs. It has found its application in a variety of practically important fields: in planning production schemes and schedules, chip design, multi-threaded data processing, parallelization and distribution of computations, and other areas where a set of consistent technological constraints is imposed on the order of the task execution. In general, most of these problems belong to the NP-hard class, so it is important to find subclasses of problems for which exact polynomial-time algorithms exist, and to build approximate algorithms with an acceptable level of error of the obtained solutions.

Given the exponential complexity of the problem, space search schemes, in particular the branch-and-bound method, have become widespread among methods for finding exact solutions to such problems. Its advantages include clarity and ease of implementation.

Particularly widespread have become the exact polynomial algorithms and methods proposed by M. Gehry, D. Johnson, T.S. Hu, M. Fuji, J.C. Lenstra, E.G. Coffman and R.L. Graham. Special attention is paid to the level principle, on which the first polynomial algorithm is based. This principle is one of the fundamental approaches to approximate solution search schemes. Also important are algorithms based on maximum matching and lexicographic labeling, which are accurate in the case of a classical problem with two executors. Modification of these algorithms allows them to be used beyond classical cases.

The exact polynomial-time algorithms that were obtained in subsequent studies were mostly based on brute-force search and had algorithmic complexity whose exponent depends multiplicatively on the width of the sequencing or the length of the critical path. In recent years, researchers have focused on the development of (1+ε)-accurate quasi-polynomial algorithms. The use of such algorithms in practice is often infeasible, hence the importance of obtaining approximate algorithms of acceptable accuracy with low algorithmic complexity.

The classical formulation of the problem assumes that all executors are universal, their number is fixed, and the time for each task execution is the same. This formulation allowed to obtain a number of theoretically important results, but it corresponds to a narrow scope of real conditions, so some generalisations were developed later that take into account the practical aspects of the modeled processes in finer detail. These include: the presence of delays, different times of task execution, the possibility of interrupting the execution, specifying time intervals within range of which each or some of the tasks should be completed (“start-times” and “deadlines”); tasks that require simultaneous execution by several, possibly specific, executors; variable number of executors, different optimality criteria, etc. Problems that take into account these generalisations, in addition to also being NP-hard, may not have any feasible solutions at all. This and the fact that to model real-world processes, it is usually necessary to take into account some combination of the above constraints emphasize the importance of building efficient approximate approaches and algorithms, both universal and specific to particular settings.

In this thesis, a number of statements have been derived related to the possibility of reducing sequencing problems to certain subclasses of these problems. Their proof is based on the idea of adding special subgraphs to the input graph, which, if the order is chosen correctly, will not affect the length of the optimal solution. It turned out that the desired order of these subgraphs can be determined using a binary search.

Particular attention is paid to problems in which the desired sequencing is dense. It has been proved that the algorithm based on the maximum matching can, in principle, find an optimal solution to such problems, unlike algorithms based on the level principle.

Computational experiments have shown that the algorithm in its classical form has unsatisfactory accuracy, so a number of modifications have been developed which have significantly improved its performance.

The necessary conditions for the existence of dense sequencing related to the limited capacity of the places and the possibility of filling them were also investigated. These considerations not only made it possible to combine all the derived conditions into one, but also to obtain a new refined lower bound on the length and generalisation of the special sequencings $\underline{S}$ та $\overline{S}$, which take into account the width of the sequencing. Furthermore, the combination of these results produces a series of lower bound estimates, each of which is theoretically more accurate than the previous one. An approximate algorithm for such problems was also proposed, the idea of which is to construct an option tree in the branch-and-bound method only for the most promising branches, based on some chosen criterion.

The analysis of cases of exceeding the maximum allowed number of steps in computational experiments with the branch-and-bound method showed that this typically occurs because a large number of branches corresponding to isomorphic subgraphs are formed. In this case, obtaining more accurate estimates will not affect the number of branches. In this regard, the possibility of identifying and removing such branches in the option tree was investigated, both for the general case and for series-parallel graphs. Additionally, the possibility of using powerful invariants for an approximate identification of isomorphism was also analysed.

When considering the cases of suboptimality of the algorithm based on the level principle, for the variable-width problem for binary in-trees, the desired requirements that the algorithm must satisfy were identified. One such condition was the independence of the choice of vertices from the capacities of subsequent places in the sequencing. An algorithm that meets all the desired requirements was proposed and a computational experiment was conducted to compare it with an algorithm based on the level principle. Further analysis has shown that it is impossible to construct an exact algorithm with the aforementioned property.

To take into account the incomplete workload of executors, a new formulation of the problem was proposed, in which executors have certain "days off" during which they cannot

execute tasks. To solve such a problem, an algorithm based on maximum matching was proposed, in which edges between vertex-tasks of the same executor were removed from the reachability graph. In general, such an algorithm turned out to be only approximate, as did the algorithm based on lexicographic labelling.

The scientific novelty of the obtained results is as follows:

- for the *first time*, the possibility of reducing any classical optimal sequencing problem to a problem with a dense sequencing and a given parity of width was proved;

- the classical algorithm for solving the problem with two executors was *further developed* for the construction of dense sequencings;

- improved lower bound estimates of the sequencing length for the classical parallel sequencing problem were obtained;

- for the *first time*, an approach to reduce the branching in the branch-and-bound method by excluding branches corresponding to isomorphic subgraphs is proposed: exact and approximate variants of its implementation are considered;

- for the *first time*, a necessary condition for the existence of dense sequencings was obtained and efficient algorithms for its testing for general and special graph structures were proposed;

- for the *first time*, an approximate algorithm for problems with dense sequencings based on the branch-and-bound method with a limited search depth was constructed;

- for the *first time*, a sequence of lower-bound estimates of the sequencing length was constructed, in which each subsequent estimate is theoretically more accurate than the previous one;

- a method for determining the range of valid vertex places was *further developed* by taking into account the width of the sequencing;

- for the *first time*, the possibility of reducing a problem with a variable sequencing width to a classical problem was proved;

- *new scientific results* on polynomial approximate algorithms for solving the problem with variable width for binary in-trees were *obtained*: an online-algorithm was proposed

and studied, and it was demonstrated that the construction of an exact online-algorithm for this class of problems is impossible in general;

- for the *first time*, a class of generalized optimal sequencing problems with incomplete workload is considered: the possibility of reducing it to other known classes of parallel sequencing problems is shown, and an approximate algorithm for solving problems from this class is proposed;

- a software project was created using the tools of the object-oriented programming language C#;

- computational experiments were conducted to determine the effectiveness of the proposed methods and algorithms.

The methods and algorithms developed in this work can be applied to solving practical problems in various subject areas, which are reduced in the mathematical formulation to the problems of optimal sequencing of vertices of directed graphs. The obtained results can improve the efficiency of scheduling tasks in various practically important areas and industries, including industrial production, logistics, prioritization of resource use and their allocation, parallelization and distribution of computing, real-time data processing and monitoring, etc. In addition, they can be of great importance for the development of modern technologies, in particular automated parallelization, and serve as a foundation for new areas of research in this field.

The main results of the thesis have been published in 13 scientific works: 3 papers in Ukrainian scientific professional journals of category “B” in physical and mathematical sciences, 1 papers in scientific journal indexed by the Scopus scientific and metric base, 1 paper in other Ukrainian scientific professional journals in physical and mathematical sciences; 8 abstracts of reports in collections of materials of international and regional scientific conferences and workshops.

Keywords: discrete optimization, combinatorial optimization, schedule theory, optimization problems on graphs, parallel sequencing problems, branch-and-bound method, estimates of sequencing characteristics, isomorphism-free enumeration, dense sequencing, level principle, graph labelling, problems with day offs, optimal partitioning.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА

Наукові праці, в яких опубліковані основні наукові результати дисертації:

1. Караваєв К. Д. Про необхідні умови існування щільних упорядкувань в класичній задачі паралельного упорядкування. *Збірник наукових праць «Системні технології»*, м. Дніпро, 2024. Вип. 151. С. 76–91. doi: https://doi.org/10.34185/1562-9945-2-151-2024-07. Режим доступу до ресурсу: https://journals.nmetau.edu.ua/index.php/st/article/view/1711.

2. Караваєв К. Д., Турчина В. А. Узагальнення задач упорядкування з урахуванням неповного завантаження. *Збірник наукових праць «Питання прикладної математики і математичного моделювання»*, м. Дніпро, 2022. Вип. 22. С. 67–79. doi: https://doi.org/10.15421/322207. Режим доступу до ресурсу: https://pm-mm.dp.ua/index.php/pmmm/article/view/341.

3. Караваєв К. Д., Турчина В. А. Аналіз впливу автоморфізму графу на схеми направленого перебору. *Збірник наукових праць «Питання прикладної математики і математичного моделювання»*, м. Дніпро, 2021. Вип. 21. С. 94–104. doi: https://doi.org/10.15421/322110. Режим доступу до ресурсу: https://pm-mm.dp.ua/index.php/pmmm/article/view/313.

4. Turchyna V., Karavaiev K. Analysis of algorithms for constructing dense sequencing of digraphs vertices. *Proceedings of The Third International Workshop on Computer Modeling and Intelligent Systems (CMIS-2020)*, Zaporizhzhia, 2020. P. 690–703. doi: https://doi.org/10.32782/cmis/2608-53. Режим доступу до ресурсу: https://ceur-ws.org/Vol-2608/paper53.pdf (Scopus).

5. Турчина В. А., Караваєв К. Д. Дослідження оцінок довжини паралельного упорядкування вершин графу. *Збірник наукових праць «Питання прикладної математики і математичного моделювання»*, м. Дніпро, 2018. Вип. 18. С. 186–195. doi: https://doi.org/10.15421/321819. Режим доступу до ресурсу: https://pm-mm.dp.ua/index.php/pmmm/article/view/236.

Наукові праці, які засвідчують апробацію матеріалів дисертації:

6. Турчина В. А., Караваєв К. Д. Контрприклад до алгоритму перерахування паралельно-послідовних графів без ізоморфізму. *Математичне та програмне забезпечення інтелектуальних систем (MSSIS-2023): Матеріали XXI міжнародної науково-практичної конференції, 22-24 листопада 2023 р.*, м. Дніпро, 2023. С. 291–292. Режим доступу до ресурсу: http://mpzis.dnu.dp.ua/wp-content/uploads/2023/11/mpzis-2023.pdf.

7. Караваєв К. Д., Турчина В. А. Про необхідні та достатні умови наявності автоморфізму у паралельно-послідовних графах. *Комбінаторні конфігурації та їхні застосування: Матеріали XXV Міжнародного науково-практичного семінару імені А. Я. Петренюка, (Запоріжжя – Кропивницький, 14-16 червня 2023 року)* / за ред. Г.П. Донця, м. Запоріжжя - Кропивницький, 2023. С. 122–129. Режим доступу до ресурсу: https://zp.edu.ua/uploads/dept_s&r/2023/conf/1.4/Petrenyuk_ISPS-25-proc.pdf.

8. Караваєв К. Д. Використання інваріантів графів для скорочення напрямленого перебору у задачах упорядкування. *Математичне та програмне забезпечення інтелектуальних систем (MSSIS-2022): Матеріали XX ювілейної міжнародної науково-практичної конференції, 23-25 листопада 2022 р.*, м. Дніпро, 2022. С. 96–97. Режим доступу до ресурсу: http://mpzis.dnu.dp.ua/wp-content/uploads/2022/12/MPZIS-2022-1.pdf.

9. Караваєв К. Д., Турчина В. А. Про зв’язок задач ізоморфізму графів та упорядкування їх вершин. *Комбінаторні конфігурації та їхні застосування: Матеріали XXIV Міжнародного науково-практичного семінару імені А. Я. Петренюка, (Запоріжжя – Кропивницький, 13-14 травня 2022 року)* / за ред. Г.П. Донця, м. Запоріжжя - Кропивницький, 2022. С. 30–33. Режим доступу до ресурсу: https://zp.edu.ua/uploads/dept_s&r/2023/conf/1.4/Petrenyuk_ISPS-25-proc.pdf.

10.Караваєв К. Д., Турчина В. А. Деякі узагальнення задачі паралельного упорядкування. *Комбінаторні конфігурації та їхні застосування: Матеріали XXIII Міжнародного науково-практичного семінару імені А. Я. Петренюка, присвяченого 70-річчю Льотної академії Національного авіаційного університету, (Запоріжжя –*

Кропивницький, 13-15 травня 2021 року) / за ред. Г.П. Донця, м. Запоріжжя - Кропивницький, 2021. С. 93–97. Режим доступу до ресурсу: https://www.glau.kr.ua/images/docs/sbornik/materiali_23_mnp_seminaru.pdf.

11.Karavaiev K., Turchyna V., Hurko O. The problem of consistencing the architecture of computational systems and algorithms. *Сучасні науково-технічні дослідження у контексті мовного простору (іноземними мовами) 13 травня 2021 року: матеріали X Регіональної науково-практичної конференції молодих учених та студентів*, м. Дніпро, 2021. С. 142–145. Режим доступу до ресурсу: https://www.dnu.dp.ua/docs/ndc/2021/19_Сучасні%20науково-технічні%20дослідження%20у%20контексті%20мовного%20простору.pdf.

12.Турчина В. А., Караваєв К. Д. Дослідження оцінки точності алгоритму, заснованого на лексикографічному порядку. *Математичне та програмне забезпечення інтелектуальних систем (MSSIS-2019): Матеріали XVII міжнародної науково-практичної конференції, 20-22 листопада 2019 р.*, м. Дніпро, 2019. С. 258–259. Режим доступу до ресурсу: http://mpzis.dnu.dp.ua/wp-content/uploads/2019/12/MPZIS_2019.pdf.

13.Турчина В. А., Караваєв К. Д. Застосування рівневого принципу до аналізу задач паралельного упорядкування та їх узагальнення. *Міжнародний науковий симпозіум «Інтелектуальні рішення». Обчислювальний інтелект (результати, проблеми, перспективи): праці міжнар. наук.-практ. конф., 15-20 квітня 2019 р.*, м. Ужгород, 2019. С. 56–57. Режим доступу до ресурсу: https://er.chdtu.edu.ua/bitstream/ChSTU/3192/1/OI-2019.pdf.

ЗМІСТ

ВСТУП

Актуальність теми. Математична теорія паралельного упорядкування є одним з потужних інструментів розв’язання багатьох теоретичних та практичних задач, що можуть бути зведеними до оптимізаційних задач на графах. Її результати застосовуються при плануванні схем та розкладів виробництва, побудові мікросхем, багатопотоковій обробці даних, паралелізації та розподіленні обчислень та в інших сферах, де на порядок виконання задач накладається деяка множина несуперечливих технологічних обмежень. У загальному випадку більшість таких задач, навіть у найпростіших постановках, відносяться до класу NP-важких, тому важливим є як пошук підкласів задач, для яких існують точні поліноміальні алгоритми, так і побудова наближених алгоритмів з прийнятним рівнем похибки отриманих розв’язків.

Історично перші поліноміальні алгоритми розв’язання таких задач були засновані на рівневому принципі, використанні максимального паросполучення та лексикографічного помічення. Перший є точним для графів-лісів, інші – для класичної постановки задачі з двома виконавцями. Ці алгоритми легко модифікувати, що дозволяє значно розширити межі їх застосовності.

Точні поліноміальні алгоритми, отримані в подальших дослідженнях, найчастіше були засновані на переборі або мали експоненційну алгоритмічну складність. На практиці вони є малопридатними, що додатково підтверджує актуальність отримання наближених алгоритмів прийнятної точності з малою алгоритмічною складністю.

Класична постановка задачі передбачає, що всі виконавці є універсальними, їх кількість є сталою і час виконання кожної роботи є однаковим. Вона є далекою від реальних умов, що зумовило розробку її узагальнень, що детальніше враховують практичні аспекти процесів, які моделюються. До них належать: наявність затримок, різний час виконання робіт, можливість переривання виконання, зазначення проміжків часу, в які має бути завершена кожна або деякі з робіт (директивні строки); завдання, виконання яких потребує одночасної роботи декількох, можливо

конкретних, виконавців; змінна кількість виконавців, різні варіанти цільових функцій, тощо. Задачі, що враховують перелічені узагальнення, окрім того, що також найчастіше належать класу NP-важких, можуть й зовсім не мати розв'язків. Це та той факт, що для моделювання реальних процесів зазвичай необхідно враховувати якусь комбінацію з наведених обмежень, підкреслюють важливість побудови ефективних наближених підходів та алгоритмів як універсальних, так і специфічних для конкретних постановок.

Зв'язок роботи з науковими програмами, планами, темами. Дисертаційне дослідження проводилося в рамках тем науково-дослідних робіт Міністерства освіти і науки України «Математичні моделі, методи та алгоритми розв'язання задач аналізу складних систем» (№ держреєстрації 0119U101302, 2019–2021 рр.), «Розробка та реалізація методів оптимального функціонування складних систем» (№ держреєстрації 0122U001466, 2022–2024 рр.) при кафедрі обчислювальної математики та математичної кібернетики Дніпровського національного університету імені Олеся Гончара.

Мета і завдання дослідження. *Метою дисертаційної роботи* є подальша розробка теоретичного апарату, що лежить в основі методів і точних та наближених алгоритмів розв'язання задач паралельного упорядкування.

Завданнями дослідження для досягнення поставленої мети були:

- подальший розвиток теорії розв'язання задач дискретної оптимізації та її використання в задачах упорядкування;
- розробка нового апарату для отримання точних розв'язків задач у загальних постановках;
- розробка методів та алгоритмів поліноміальної складності для отримання точних та наближених розв'язків та їх обґрунтування;
- розробка комплексу програм, що реалізує запропоновані алгоритми;
- проведення обчислювальних експериментів для перевірки гіпотез та результатів для отриманих наближених алгоритмів.

Об'єкт дослідження – задачі оптимального упорядкування вершин орграфів.

Предметом дослідження є методи і алгоритми розв’язання класичних та узагальнених задач оптимального паралельного упорядкування вершин орграфів.

Методи дослідження: для вирішення поставлених завдань використані методи теорії оптимального упорядкування, теорії графів, дискретної та комбінаторної оптимізації, теорії розкладів.

Наукова новизна одержаних результатів полягає у наступному:

– *вперше* теоретично обґрунтовано можливість зведення будь-якої класичної задачі оптимального упорядкування до задачі із щільним упорядкуванням та шириною заданої парності;

– *дістав подальшого розвитку* класичний алгоритм розв’язання для двох виконавців для побудови щільних упорядкувань;

– удосконалено оцінки знизу довжини упорядкування для класичної задачі паралельного упорядкування;

– *вперше* запропоновано підхід для скорочення перебору у методі гілок та меж за рахунок виключення гілок, що відповідають ізоморфним підграфам: розглянуті точні та наближені варіанти його реалізації;

– *вперше* отримана необхідна умова існування щільних упорядкувань та запропоновані ефективні алгоритми її перевірки для загальної та спеціальної структур графів;

– *вперше* побудовано наближений алгоритм для задач з щільними упорядкуваннями, який заснований на методі гілок та меж з обмеженою глибиною пошуку;

– *вперше* побудовано послідовність оцінок знизу довжини упорядкування, в якій кожна наступна оцінка, теоретично, є точнішою за попередні;

– *дістав подальшого розвитку* спосіб визначення діапазону допустимих місць вершин шляхом врахування ширини упорядкування;

– *вперше* теоретично обґрунтовано можливість зведення задачі зі змінним значенням ширини упорядкування до класичної задачі;

– *отримані нові наукові дані* про поліноміальні наближені алгоритми для розв’язання задачі зі змінною шириною для вхідних бінарних дерев: запропоновано та досліджено online-алгоритм, продемонстровано, що побудова точного online-алгоритму для такого класу задач в принципі неможлива;

– *вперше* розглянуто клас узагальнених задач оптимального упорядкування з неповним завантаженням: показана можливість зведення до інших відомих класів задач паралельного упорядкування, запропоновано наближений алгоритм до розв’язання задач з цього класу;

– створено програмний продукт, використовуючи засоби об’єктно-орієнтованої мови програмування C#;

– проведено обчислювальні експерименти для визначення ефективності запропонованих методів та алгоритмів.

Практичне значення одержаних результатів. У дисертаційній роботі досліджуються алгоритми та методи точного та наближеного розв’язання класичних та узагальнених задач оптимального упорядкування.

Отримані в результаті дослідження методи й алгоритми можуть бути застосовані до розв’язання прикладних задач, що зводяться в математичній постановці до задач оптимального упорядкування вершин орієнтованих графів. Отримані результати можуть підвищити ефективність побудови розкладів виконання завдань у різних практично важливих сферах та галузях, зокрема у промисловому виробництві, питаннях логістики, пріоритетності використання та розподілу ресурсів, паралелізації та розподілення обчислень, обробки та моніторингу даних в режимі реального часу тощо. Також вони можуть мати важливе значення для подальшого розвитку сучасних технологій, зокрема автоматичного розпаралелення, та слугувати підґрунтям для нових напрямків наукових пошуків за цією тематикою.

Окремі теоретичні результати дисертаційного дослідження включено до змісту дисципліни «Методи і алгоритми розв’язання задач дискретної оптимізації», яка викладається на кафедрі обчислювальної математики та математичної кібернетики факультету прикладної математики Дніпровського національного університету імені

Олеся Гончара для здобувачів вищої освіти рівня PhD за спеціальністю 113 «Прикладна математика» освітньо-наукової програми «Прикладна математика» та дисципліни «Паралельні алгоритми та системи» для здобувачів другого (магістерського) рівня освіти зі спеціальності 124 «Системний аналіз» освітньої програми «Системний аналіз». Результати дисертації також використовуються при виконанні курсових та дипломних робіт студентами, які навчаються за спеціальностями 113 «Прикладна математика» та 124 «Системний аналіз».

Особистий внесок здобувача. Результати дисертаційної роботі представлені в 13 наукових працях [1-13]. Усі результати дисертації, що виносяться на захист, отримані автором особисто. У працях, що опубліковані у співавторстві, здобувачеві належить: [2] – постановка задачі з вихідними, порівняння з іншими відомими постановками, наближений алгоритм її розв’язання та аналіз його властивостей; [3] – новий підхід до скорочення перебору за рахунок видалення гілок, які відповідають ізоморфним підграфам, дослідження взаємозв’язку між вершинами, які видаляються, та відповідними підграфами, алгоритм розгалуження без ізоморфізму та дослідження його властивостей; [4] – твердження, що дозволяють звести класичні задачі упорядкування до задач з щільними упорядкуваннями, твердження про існування максимального паросполучення, за яким можна побудувати оптимальний розв’язок, модифікації алгоритму, заснованого на максимальному паросполученні, порівняльний аналіз цих модифікацій на основі обчислювальних експериментів; [5] – уточнена оцінка знизу довжини упорядкування та порівняльний аналіз з іншими відомими оцінками; [6,7] – дослідження властивостей алгоритму розгалуження без ізоморфізму при його застосуванні до паралельно-послідовних графів; [9] – дослідження можливості побудови інваріантів графів, заснованих на задачах оптимального упорядкування; [10,11] – огляд узагальнених задач оптимального упорядкування; [12,13] – дослідження властивостей алгоритмів, заснованих на рівневому принципі.

Апробація результатів дисертації. Основні положення та результати дисертаційної роботи доповідались і обговорювались на семінарі «Сучасні питання

оптимізації та дискретної математики» при Науковій раді НАН України з проблеми «Кібернетика», який функціонує при Дніпровському національному університеті імені Олеся Гончара; на міжнародних наукових конференціях «Математичне та програмне забезпечення інтелектуальних систем» (м. Дніпро, 2019, 2022, 2023 рр.); «Комбінаторні конфігурації та їхні застосування» (м. Запоріжжя – Кропивницький, 2021, 2022, 2023 рр.); «Інтелектуальні рішення» (м. Ужгород, 2019 р.) та на регіональній науковій конференції «Сучасні науково-технічні дослідження у контексті мовного простору (іноземними мовами)» (м. Дніпро, 2021 р.).

Публікації. Основні результати дисертаційної роботи опубліковано в 13 наукових працях: 1 стаття [4] у виданні, що індексується наукометричною базою Scopus; 3 статті ([1, 2, 3]) у наукових фахових виданнях України категорії «Б» з фізико-математичних наук, 1 стаття [5] в інших наукових фахових виданнях України з фізико-математичних наук, 8 тез доповідей у збірниках матеріалів міжнародних та регіональних наукових конференцій і семінарів ([6-13]).

Структура та обсяг дисертації. Дисертаційна робота складається зі вступу, чотирьох розділів, висновків, переліку використаних джерел, що містить 116 найменувань на 12 сторінках та додатків на 5 сторінках. Загальний обсяг дисертації – 178 сторінок, обсяг основного тексту – 161 сторінка. Робота містить 42 рисунки та 16 таблиць.

Розділ 1. ЗАДАЧІ ПАРАЛЕЛЬНОГО УПОРЯДКУВАННЯ. АЛГОРИТМИ ТА МЕТОДИ ЇХ РОЗВ'ЯЗАННЯ. ВИБІР НАПРЯМКУ ДОСЛІДЖЕННЯ

1.1 Прикладні задачі, що зводяться до задач упорядкування вершин орграфів

Серед прикладних задач, що зводяться до задач дискретної оптимізації, виділяють ті, в яких скінченну множину завдань, порядок виконання яких зумовлений деякими технологічними особливостями, необхідно виконати або за мінімальний час при заданих обмеженнях на ресурси, або мінімізувавши ресурси при заданому часі завершення робіт. При їх формалізації найчастіше отримуємо задачі оптимального упорядкування вершин орграфів.

Першим поштовхом до розвитку задач оптимального упорядкування вершин графів усередині минулого століття стала проблема, пов'язана з оптимізацією порядку виконання робіт на конвеєрах автомобільного виробництва [14]. Ефективний розв'язок формалізованої задачі був знайдений відносно швидко, тому панувала думка, що задача може бути розв'язана у загальному випадку за поліноміальний час.

Зокрема такі задачі виникають у багатьох сферах та ґалузях виробництва товарів та послуг, таких як важка та хімічна промисловість, будівництво, сільське господарство, товарна індустрія тощо [15–21]; при плануванні виробництва відповідно до пріоритетів в умовах обмежених ресурсів [22,23]; у задачах логістики, розподілу ресурсів та побудови розкладів для періодичних подій [23].

В подальшому істотним імпульсом для дослідження цього напрямку стала необхідність розробки обчислювальної техніки нової архітектури. Спочатку розвиток електронних обчислювальних машин (ЕОМ) відбувався екстенсивним шляхом, він був зумовлений тим, що нові класи задач великої розмірності потребували все більших обчислювальних потужностей. Так кожне експериментальне обчислення в задачах кліматології, зокрема при моделюванні динаміки атмосфери Землі, потребує виконання близько 10^{16}-10^{17} операцій, в задачах авіабудування, таких як моделювання обтікаючого потоку літальних апаратів, – близько 10^{15}-10^{16} операцій, в задачах моделювання фізичних процесів у тривимірних об'єктах – близько 10^{11}-10^{14} операцій

[24-25]. Проте в рамках класичної архітектури отримати розв’язки таких задач за прийнятний час все одно не вдавалося. В зв’язку з цим було запропоновано інший, інтенсивний напрямок розвитку комп’ютерів, який нероздільно пов'язаний з терміном «паралельність»: для збільшення кількості виконуваних в секунду операцій обчислення розподілялися між паралельно працюючими процесорами.

Програмні та апаратні архітектури сучасних комп’ютерів неможливо уявити без паралелізму: в операційних системах одночасно можуть виконуватись десятки програм, робота яких забезпечується паралельним використанням ядер процесору, пристроїв введення/виведення, розподілених ресурсів тощо. В умовах, коли за останні роки інженери впритул наблизилися до граничного технологічного процесу 1 нм, розвиток паралелізму набуває особливого значення.

В зв’язку з розвитком туманних обчислень (архітектура, в якій розподілення обчислень, зберігання даних та комунікацій відбувається не тільки у хмарних сервісах, а й у локальних мережах пристроїв) протягом останніх років, багато робіт [26–33] присвячені дослідженню таких задач в рамках саме цієї сфери застосування. Авторами розглядаються задачі упорядкування в контексті забезпечення роботи складних мобільних додатків для мінімізації затримок комунікації, часу виконання обчислень, витрат енергії тощо.

Особливу увагу цим задачам також приділяють при розробці систем реального часу [34–35], які мають забезпечувати своєчасне надання коректних результатів. До них належать системи моніторингу пацієнтів та атомних електростанцій, авіаційні та телекомунікаційні системи тощо.

Одним з найперспективніших застосувань результатів теорії упорядкувань є автоматичне розпаралелення програмного коду [36–38]. Теоретично, такі компілятори зможуть самостійно будувати оптимальні схеми розпаралелення виконання команд машинного коду без участі розробника, що в загальному випадку є дуже трудомісткою задачею. Розвиток цієї технології безпосередньо обмежений відсутністю математичного апарату, який дозволяв би ефективно розв’язувати задачі побудови упорядкувань графів надвеликої розмірності за прийнятний час.

Відзначимо окремо, що особливе значення для застосування розподіленої обробки вхідних потоків даних мають задачі упорядкування, в яких технологічні обмеження можуть бути представлені у вигляді паралельно-послідовних графів [39].

1.2 Класичні постановки задач паралельного упорядкування

Для формалізації задач про побудову розкладу виконання скінченної множини робіт, в яких вихідні результати деяких з них є вхідними для інших, прийнято використовувати орієнтовані графи. Тому задачі оптимального упорядкування в цій роботі подаються у формі оптимізаційних задач на орграфах. Терміни з теорії графів використовуються у значенні, відповідно до [40].

Введемо деяке бієктивне відображення $\dot{f}$ з множини, що складається з n робіт, у підмножину натуральних чисел $V = \{1,2, \dots, n\}$. У подальшому обмежимося розглядом лише такої множини V, що природно пов'язана із множиною робіт (завдань, операцій, проектів, процесів, тощо). Введемо також у розгляд множину $U \subset V \times V$, де пара $(i,j) \in U$, якщо результати роботи $\dot{f}^{-1}(i)$ безпосередньо використовуються при виконанні роботи $\dot{f}^{-1}(j)$. Шляхом комбінування цих множин може бути побудований орграф $G = (V,U)$, який буде відображати усю множину робіт та виробничих зв'язків між ними.

Перш ніж формулювати задачі введемо деякі відомі означення [41].

Означення 1.1. Лінійним упорядкуванням S скінченної множини V, що складається з n елементів, називається розміщення її елементів по n місцях, розташованих у лінію так, що кожен знаходиться тільки на одному місці.

Означення 1.2. Довжиною l упорядкування S називається число непорожніх місць в ньому: $l(S) = \sum_{i=1}^{n} sign|S[i]|$, де $S[i]$ – множини елементів, що знаходяться в упорядкуванні S на місці i.

Означення 1.3. Шириною h упорядкування S називається величина, яка дорівнює найбільшій кількості елементів, що розташовані на одному місці: $h(S) = \max\limits_{1 \le i \le n} |S[i]|$.

Означення 1.4. Паралельним упорядкуванням вершин орієнтованого графу $G = (V, U)$ називається таке лінійне упорядкування його вершин, при якому з того, що пара $(i, j) \in U$ випливає, що вершина i розташовується в упорядкуванні S лівіше вершини j, тобто з того, що $(i, j) \in U \wedge (i \in S[p], j \in S[q])$ випливає, що $p < q$.

Перейдемо тепер до класичних оптимізаційних задач упорядкування вершин.

Задача 1. По заданим орграфу G і значенню ширини h побудувати паралельне упорядкування мінімальної довжини. Задачу позначатимемо у подальшому $P_S(G, h, l)$.

Задача 2. По заданим орграфу G і значенню довжини l побудувати паралельне упорядкування мінімальної ширини. Цю задачу позначимо через $P_S(G, l, h)$.

Зрозуміло, що для існування паралельного упорядкування необхідно і достатньо, щоб орграф був ациклічним. Паралельні упорядкування, що є розв'язками наведених задач називають оптимальними.

Примітка. Для спрощення викладення матеріалу у подальшому під «графом» будемо розуміти «орієнтовний ациклічний граф», а під «упорядкуванням» – «паралельне упорядкування», якщо не зазначено інакше.

Узагальнене позначення для цих задач можна подати у вигляді $P_S(A, B, C)$, де A – клас графів, які розглядаються у задачі (G – довільний орієнтований граф, T – орієнтовне дерево або ліс, $G_{\text{ПП}}$ – паралельно-послідовний граф), B – відома величина ($h(S)$ – *задача 1*, $l(S)$ – *задача 2*, тощо), C – цільова функція ($l(S)$ – *задача 1*, $h(S)$ – *задача 2*, тощо).

При дослідженні деяких питань у роботі використані упорядкування спеціального виду $\underline{S}, \overline{S}$.

Наявність технологічних обмежень дозволяє для кожної вершини знайти крайнє ліве та крайнє праве місця, на яких вона може стояти в допустимому упорядкуванні.

Побудуємо спеціальне упорядкування $\underline{S}$ за наступним правилом [42,43]:

1. Вважаємо всі місця в упорядкуванні $\underline{S}$ порожніми, $i = 1$.

2. У графі G шукаємо вершини, що не мають вхідних дуг, і заносимо на i-те місце в упорядкуванні $\underline{S}$.

3. Вилучаємо з графу G розглянуті вершини разом з інцидентними дугами.

4. Якщо множина V порожня, то кінець алгоритму.

5. Перепозначимо: $G \coloneqq G, i \coloneqq i + 1$, перехід на пункт 2.

Довжину упорядкування $\underline{S}$ позначимо $\underline{l}$.

Упорядкування $\overline{S}$ будуємо в такий спосіб:

1. Вважаємо місця з 1-го по $\underline{l}$ в упорядкуванні $\overline{S}$ порожніми, $i = \underline{l}$.

2. У графі G шукаємо вершини, що не мають вихідних дуг, і заносимо на i-те місце в упорядкуванні $\overline{S}$.

3. Вилучаємо з графу G розглянуті вершини разом з інцидентними дугами.

4. Якщо множина V порожня, то кінець алгоритму.

5. Перепозначимо: $G \coloneqq G, i \coloneqq i - 1$, перехід на пункт 2.

Довжину упорядкування $\overline{S}$ позначимо $\overline{l}$.

Крайнє ліве припустиме місце вершини i у шуканому упорядкуванні відповідає її місцю в упорядкування $\underline{S}$, а крайнє праве визначається як $\overline{\mu_i} + (n - \underline{l})$, де $\overline{\mu_i}$ – місце вершини i в упорядкуванні $\overline{S}$.

1.3 Узагальнення задач паралельного упорядкування

Розглянемо деякі узагальнення класичних задач оптимального упорядкування, які природним чином виникають через необхідність врахування додаткових обмежень та вхідних даних, окрім кількості завдань та обмежень на їх порядок виконання [44].

Узагальнення класичної задачі охоплюють усі складові задачі: вводяться додаткові умови на виконання робіт, змінюється структура завдань, властивості виконавців або цільова функція.

Узагальнення із ускладненою структурою завдань. Необхідність враховувати різний час виконання завдань стала поштовхом до дослідження задач, в

яких одна вершина може займати декілька місць. Окрім цього, для виконання деяких завдань одного виконавця може бути недостатньо, чи можуть бути потрібні деякі конкретні виконавці, тоді розглядають задачі, в яких одна вершина може займати декілька позицій на одному місці.

З**адачі з ускладненою структурою виконавців**. Для випадків, коли в різні моменти часу доступна різна кількість виконавців, вводяться задачі, де кількість вершин, які можна ставити на i-те місце, не більша ніж h_i. Задачі з різним часом обробки завдань додатково поділяються на задачі із перериваннями (можна зупинити виконання завдання та продовжити його потім), які є імітацією пріоритетної багатозадачності, та без переривань. Також розглядаються часові затримки між виконанням завдань, що можуть бути пов’язані з необхідністю передачі результатів між виконавцями. Виділяють задачі, де затримки є сталими і виникають лише за необхідності комунікації, та задачі, в яких затримки залежать від виконуваних завдань.

Задачі з додатковими умовами пов’язують з наявністю директивних термінів виконання та часом початку виконання завдань. Перші виникають, коли є необхідність завершити деякі завдання у заданий термін через обмежену в часі наявність того чи іншого ресурсу. Такі задачі поділяють на задачі зі строгими (не можна порушувати) та нестрогими (можна порушувати) термінами. Відмітимо також, що задача із строгими директивними термінами може не мати допустимих розв’язків. Потреба розглядати другий тип задач виникає в аналогічних умовах, проте регламентується час початку виконання завдання, а не його кінець. Окремо виокремлюють онлайн варіант задачі, де про завдання та структуру графу G дізнаємося протягом виконання завдань, що відповідає розподіленню обчислень між процесорами у обчислювальній техніці.

Окрім класичної цільової функції (довжина упорядкування) розглядаються також сума часів виконання всіх завдань та її зважений варіант. Для задачі з нестрогими директивними строками розглядають сумарне запізнення, його зважений

варіант та кількість робіт, виконаних вчасно. Також вивчаються задачі з комбінацією цих узагальнень та цільових функцій.

Наведені задачі у загальному випадку є також NP-важкими. Більшість з них також є NP-важкими для випадків, коли граф G – є лісом або набором ланцюжків, та навіть у випадках з одним виконавцем. Поліноміальні алгоритми для таких задач відомі лише для специфічних умов.

Задачі оптимального упорядкування з затримками

Одним з чинників, що інколи необхідно враховувати при пошуку оптимальних розкладів виконання низки процесів є комунікаційні часові витрати. Вони виникають, наприклад, на виробництві, якщо для подальшої обробки матеріали потрібно транспортувати до іншого цеху, та при передачі результатів обробки від одного виконавця до іншого для подальшого використання. Природним чином такі затримки можуть бути однаковими для всіх пар робіт (обробка на процесорах), або їх тривалість може залежати від конкретної пари робіт (різні підрозділи чи цехи).

Зупинимося на випадку, коли комунікаційні затримки не залежать від пари робіт, що розглядається. Дослідження таких задач було започатковано у статті [45] у наступній постановці.

Нехай задана множина робіт V та деякі обмеження на порядок їх виконання, які можна подати у вигляді ациклічного орієнтованого графу $G(V, U)$. Для кожної пари робіт i та j таких, що $(i, j) \in U$, якщо вони виконуються на різних процесорах, необхідно додатково витратити одиницю часу на те, щоб перенести інформацію з одного процесора на інший (комунікаційні витрати). Якщо ж вони виконуються на одному процесорі, тоді комунікаційних витрат немає. Вважається, що перенесення інформації між процесорами не впливає на їх доступність для використання.

У [45] було показано, що така задача є NP-важкою, а також доведено, що наближений алгоритм, який будує «активне» упорядкування, дає відносну похибку не більшу за $3 - \frac{2}{h}$. Під «активним» упорядкуванням тут розуміється таке, в якому жодну роботу не можна розпочати раніше, при цьому не починаючи деяку іншу

роботу пізніше. Розробка алгоритму з множником похибки меншим за 3 є відкритою проблемою.

У [46] показано, що у випадку, коли граф G є лісом, задача також є NP-важкою, також авторами отримано точний алгоритм лінійної складності для випадку $h = 2$. У [47] зазначено, що у випадку, коли додатково h фіксоване, задача може бути розв'язана за допомогою динамічного програмування. Складність задачі у випадку довільної структури графу G та фіксованого h не встановлена навіть для $h = 2$.

Для задачі визначення чи можливо побудувати упорядкування довжини не більше 3 у випадку довільного графу G у [48] було запропоновано поліноміальний алгоритм. З іншого боку, для аналогічної задачі без комунікаційних затримок у [49] доведено, що вона є NP-важкою. У [50] обґрунтовано, що визначення того, чи існує упорядкування довжини не більше 4 вже є NP-важкою задачею. Також у [50] показано, що найменшим можливим множником похибки для цієї задачі є 5/4, проте алгоритм із такою похибкою не відомий.

Задачі оптимального упорядкування зі змінною шириною упорядкування

Для випадків, коли в різні моменти часу доступна різна кількість виконавців, вводяться задачі, де кількість вершин, які можна ставити на i-те місце, не більша ніж h_i. Така задача моделює постійну зміну кількості вільних процесорів у ЕОМ.

Задача 1а. По заданим графу G і вектору значень ширини $h_i, i = \overline{1,n}$ побудувати паралельне упорядкування мінімальної довжини, в якому на i-ому місці буде стояти не більше h_i вершин.

У [51] отримані поліноміальні алгоритми для випадків вихідних лісів (графи-ліси, утворені з дерев, всі дуги яких спрямовані від кореня) з неспадаючими місткостями та вхідних лісів (графи-ліси, утворені з дерев, всі дуги яких спрямовані до кореня) з незростаючими місткостями, алгоритмічна складність яких складає $O\left(n^{m^2+m-6}\log n\right)$; а також для змішаних лісів (графи-ліси, утворені комбінацією вхідних та вихідних дерев) та сталих місткостей зі складністю $O\left(n^{m^2+2m-5}\log n\right)$, де $m = \max\limits_{1\le i\le n} h_i$. Наявність логарифмічного множника зумовлена тим, що алгоритм не

гарантує знаходження упорядкування з мінімальною довжиною, а лише з довжиною не більше заданого значення D, якщо таке упорядкування існує. Тому для знаходження розв'язку необхідно застосовувати бінарний пошук. Алгоритм для графів, які є змішаними лісами, полягає у підборі таких значень h_i' для розміщення вхідного лісу цього графу, щоб могли розташувати вихідний з місткостями $h_i - h_i'$ для тієї ж довжини упорядкування. Для класичної *задачі 1* при $h = 3$ запропоновано алгоритм лінійної складності для упорядкування змішаних лісів. Також у роботі доведено, що згадані задачі є NP-важкими у випадку, коли m є змінною задачі.

У [52] вдалося покращити показники для аналогічних задач: для вхідних лісів та довільних місткостей отримано алгоритм, що має складність $O(n^{m-1})$, для вихідних лісів та довільних місткостей – $O(n^{m-1}\log n)$, для змішаних лісів та сталих місткостей – $O(n^{2m-2}\log n)$. Проте всі вони потребують $O(n^{m-1})$ комірок пам'яті, на відміну від попередніх алгоритмів, потреби пам'яті для яких складають $O(n)$. Запропоновані алгоритми спираються на використання декомпозиції графу на верхній та нижній підграфи, утворені зі зв'язних підграфів, залежно від співвідношення між довжиною їх критичного шляху та значенням медіани графу, яке залежить від m. Доведено, що за відомим оптимальним упорядкуванням для верхнього підграфу та місткостей h_i можна побудувати оптимальне упорядкування для вихідного графу за $O(|V| + |E|)$.

У [53] отримано алгоритм для довільних графів з обмеженою довжиною критичного шляху $\underline{l}$ та довільних місткостей, який має складність $O\left(n^{\underline{l}(m-1)+1}\right)$. Ідея алгоритму спирається на існування нуль-скорегованого розкладу, для першого місця з ізольованою вершиною i якого виконуються наступні умови:

– якщо в ньому наявні вільні позиції, то воно містить всі відкриті вершини поточного підграфу на момент розташування;

– довжина оптимального упорядкування для підграфу з вершин, що знаходяться на місцях з 1 по $i-1$, за тих же значень місткостей, складає $i-1$;

– місце i містить всі відкриті неізольовані вершини V_H поточного підграфу.

Доведено, що оптимальні нуль-скореговані розклади завжди існують та повністю визначаються множиною вершин V_H. Пошук оптимального розв'язку відбувається шляхом перебору всіх можливих множин V_H та побудові відповідних нуль-скорегованих розкладів.

Задачі оптимального упорядкування з різним часом виконання завдань

Для моделювання випадків, коли роботи мають різний час виконання, розглядаються задачі, в яких одна вершина може займати декілька місць. Окремо розглядаються випадки, де переривання виконання завдання є можливим (перерва не вплине на кінцевий результат) та неможливим (перерва може вплинути на результат виконання завдання).

Задача 3. По заданим графу G, значенню ширини h та часу виконання робіт p_i побудувати паралельне упорядкування мінімальної довжини, в якому кожна робота стоїть на p_i місцях (на сусідніх місцях поспіль для задачі без переривань чи не обов'язково поспіль для задачі з перериваннями).

У [54] доведено, що навіть при $h = 2$ та $p_i \in \{1,2\}$ задача є NP-важкою.

У роботах [55–57] досліджується виграш від можливості переривати виконання завдань для спеціальних класів графів, зокрема для графів-зірок та повних дводольних графів, залежно від їх структурних властивостей.

Задачі оптимального упорядкування з директивними термінами

Необхідність обмежувати часові рамки, в межах яких мають бути виконані всі чи деякі з завдань, зумовила розгляд задач з відповідними обмеженнями. Такі задачі є особливо значущими для систем реального часу (див. підрозділ 1.1).

У статті [58] автори розглядають задачу з двома процесорами, в якій для кожної роботи вказані директивні строки на завершення виконання. Запропонований алгоритм, заснований на розрахунку модифікованих директивних строків за формулою $d_i = \min_{d'} \left\{ d' - \left\lceil \frac{1}{2} g(i, d') \right\rceil \right\}$, де d' - деякий з модифікованих директивних строків наступників вершини, $g(i, d')$ - кількість наступників, модифіковані директивні строки яких менші або дорівнюють d'. Для їх розрахунку потрібно попередньо знайти транзитивне замикання графу. У роботі [58] доводиться, що

алгоритм, який розміщує вершини відповідно до модифікованих директивних строків, знаходить оптимальний розв'язок для вказаної задачі. Також обґрунтовується, що задача пошуку упорядкування, в якому порушено директивні строки не більше заданої кількості робіт, є NP-важкою навіть при $h = 1$. Для $h \geq 3$ алгоритм є наближеним.

У [59] отримано поліноміальний алгоритм для випадку двох процесорів та заданих директивних строків на початок та кінець виконання, який має алгоритмічну складність $O(n^3)$. Аналогічно до [58], основою алгоритму є побудова списку пріоритетів вершин за спеціальним правилом. На кожному кроці обирається підмножина вершин, доступних до розміщення, що мають найвищі пріоритети.

Дослідженню цих та інших задач дискретної та комбінаторної оптимізації присвятили роботи багато вітчизняних авторів, зокрема Сергієнко І.В., Яковлєв С.В., Кісельова О.М., Пічугіна О.С., Козін І.В., Гук Н.І., Перепелиця В.О., Семенова Н.В., Романова Т.Є., Новожилова М.В., Колєчкіна Л.М., Донець Г.П., Семенюта М.Ф., Гуляницький Л.Ф., Ходзінський О.М., Петренюк А.Я., Ємець О.О., Стецюк П.І., Бурдюк В.Я., Кукса А.І., Турчина В.А. [60-82].

1.4 Огляд точних методів розв'язання задач паралельного упорядкування

1.4.1 Метод повного перебору

Метод повного перебору для задачі оптимального упорядкування можна розглядати як один з варіантів методу гілок та меж (див. пункт 1.4.2), де оцінка знизу довжини упорядкування приймає стале значення. Помітимо, що у такому випадку перехід на наступну гілку дерева варіантів не відбудеться доти, доки не вичерпаються усі варіанти на поточній гілці. Очевидно, що метод повного перебору є точним, оскільки метод гілок та меж є точним, більш того, метод повного перебору – найгірший випадок методу гілок та меж, при якому вони потребують однакову кількість кроків.

Повний перебір також може бути організований за якоюсь іншою схемою.

1.4.2 Алгоритм методу гілок та меж

Метод гілок та меж [83] є класичним і найбільш поширеним точним методом розв'язання задач дискретної оптимізації. Це метод напрямленого перебору множини допустимих розв'язків. Схему методу зручно зображати у вигляді так званого дерева варіантів (розгалуження). При розробці методу для конкретної задачі необхідно визначитися з двома питаннями:

1) як саме розбивати допустиму множину на підмножини;

2) як саме оцінювати значення цільової функції при вибраному розбитті.

Зупинимося на варіанті методу для *задачі 1*.

Розглянемо дерево варіантів з вершинами $T_{opt} = \{T_i(G_i, S_i)\}$, де G_i та S_i – відповідно граф та упорядкування, що відповідають вершині i дерева. Задана ширина упорядкування h.

1) Визначимо корінь дерева T_0. Нехай $G(V, U)$ – початковий граф, S – порожнє упорядкування. Розглянемо $T_0(G, S)$, $m = 1$.

2) Для вершини дерева, що розглядається, знаходимо множину вершин R графу $G(V, U)$, що їй відповідає, які не мають вхідних дуг (це вершини, які належать множині $\underline{S}[1]$ графу, який розглядається на даному кроці). Визначаємо крайнє ліве порожнє місце k в упорядкуванні S, що відповідає цій вершині дерева.

3) Якщо $|R| \le h$, то додаємо дочірню вершину $T_m(G \backslash R, S \oplus R)$, $m := m + 1$. (Розташовуємо ці вершини на k-ому місці: $S[k] = R$)

Примітка. Тут і далі $G \backslash \cdot$ позначає підграф графу G, утворений шляхом видалення однієї чи декількох вказаних вершин з усіма вхідними та вихідними дугами.

4) Якщо $|R| > h$, то знаходимо усі $p = C_{|R|}^{h}$ комбінацій $R_q, q = \overline{1, p}$ з елементів R і додаємо p дочірніх вершин $T_{m+q-1}\left(G \backslash R_q, S \oplus R_q\right)$, $q = \overline{1, p}$, $m := m + p$.

5) Для кожної доданої вершини $T_j\left(G_j, S_j\right)$ визначаємо оцінку знизу цільової функції, що дорівнює $k + f\left(G_j, h\right)$, де $f(\cdot)$ – деяка нижня оцінка довжини упорядкування.

6) Обираємо вершину $T_r(G_r, S_r)$ з найменшою нижньою оцінкою (межею), якщо таких вершин декілька, то ту, яка є найбільш віддаленою від кореня дерева (якщо їх декілька, то довільну з них).

7) Якщо $V_r = \emptyset$, то кінець алгоритму, інакше $G := G_r, S := S_r$ і перехід на 2.

При використанні методу гілок та меж для розв'язання *задачі 1а*, замість ширини h нам відомі обмеження h_i на кількість вершин на кожному місці i $(i = \overline{1,n})$.

В цьому випадку алгоритм методу гілок та меж буде відрізнятися тим, що у пунктах 3) та 4) замість h потрібно порівнювати потужність множини вершин без вхідних дуг із величиною h_k, де k – номер крайнього лівого порожнього місця в поточному упорядкуванні S. Окрім цього, у пункті 5) будуть застосовуватися деякі інші нижні оцінки довжини упорядкування $f\left(G_j, H_k = \{h_{k+1}, \dots, h_n\}\right)$, які тепер залежать від h_i.

Приклад 1.1. Застосуємо метод гілок та меж для побудови $P_S(G, h, l)$ для графу з рис. 1.1 та $h = 3$, в якості нижньої оцінки обираємо $f(G, h) = \left\lceil \frac{n}{h} \right\rceil$.

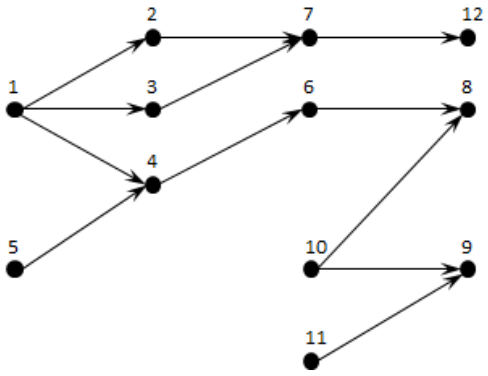

Рис. 1.1. Зображення графу з прикладу 1.1

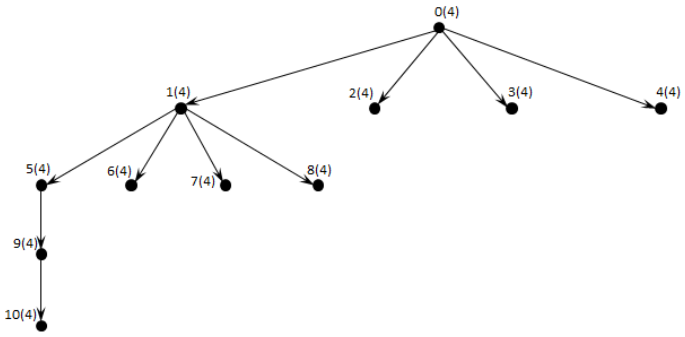

Рис. 1.2. Дерево варіантів для графу з рис. 1.1

Граф, що відповідає дереву розгалуження, зображено на рис. 1.2. Нумерація вершин дерева відповідає порядку розгалуження. Упорядкування S_i, які відповідають i-ій вершині дерева, будуть мати наступний вигляд:

$$S_1 = \begin{Bmatrix} 1 \\ 5, \dots \\ 10 \end{Bmatrix}; S_2 = \begin{Bmatrix} 1 \\ 5, \dots \\ 11 \end{Bmatrix}; S_3 = \begin{Bmatrix} 1 \\ 10, \dots \\ 11 \end{Bmatrix}; S_4 = \begin{Bmatrix} 5 \\ 10, \dots \\ 11 \end{Bmatrix}; S_5 = \begin{Bmatrix} 1 & 2 \\ 5, & 3, \dots \\ 10 & 4 \end{Bmatrix};$$
$$S_6 = \begin{Bmatrix} 1 & 2 \\ 5, & 3, \dots \\ 10 & 11 \end{Bmatrix}; S_7 = \begin{Bmatrix} 1 & 2 \\ 5, & 4, \dots \\ 10 & 11 \end{Bmatrix}; S_8 = \begin{Bmatrix} 1 & 3 \\ 5, & 4, \dots \\ 10 & 11 \end{Bmatrix}; S_9 = \begin{Bmatrix} 1 & 2 & 6 \\ 5, & 3, & 7, \dots \\ 10 & 4 & 11 \end{Bmatrix};$$

$$S_{10} = \begin{Bmatrix} 1 & 2 & 6 & 8 \\ 5, & 3, & 7, & 9 \\ 10 & 4 & 11 & 12 \end{Bmatrix}.$$

Розгалуження проводиться відповідно до значення нижньої межі довжини кожного упорядкування, а саме:

$$\tau(S_1) = \tau(S_2) = \tau(S_3) = \tau(S_4) = \tau(S_5) = \tau(S_6) = \tau(S_7) = \tau(S_8) = \tau(S_9) = \tau(S_{10}) = 4.$$

Вершина із номером 10 відповідає оптимальному паралельному упорядкуванню, яке має вигляд

$$S^* = \begin{Bmatrix} 1 & 2 & 6 & 8 \\ 5, & 3, & 7, & 9 \\ 10 & 4 & 11 & 12 \end{Bmatrix}.$$

При зміні нумерації вершин графу (тобто виборі графу, ізоморфного даному) кількість вершин дерева варіантів може значно зрости.

Приклад 1.2. Застосуємо метод гілок та меж для побудови $P_S(G, h, l)$ для графу з рис. 1.3 та $h = 3$.

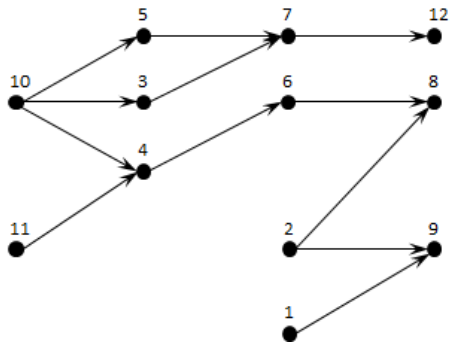

Рис. 1.3. Зображення графу з прикладу 1.2

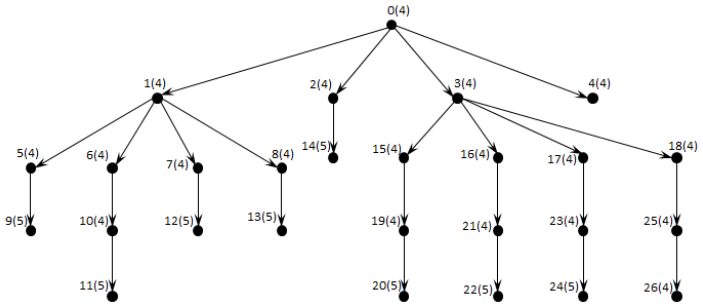

Рис. 1.4. Дерево варіантів для графу з рис. 1.3

Граф, що відповідає дереву розгалуження, зображено на рис. 1.4. Нумерація вершин дерева відповідає порядку розгалуження. Упорядкування S_i, які відповідають i-ій вершині дерева, будуть мати наступний вигляд:

$$S_1 = \begin{Bmatrix} 1 \\ 2, \dots \\ 10 \end{Bmatrix}; S_2 = \begin{Bmatrix} 1 \\ 2, \dots \\ 11 \end{Bmatrix}; S_3 = \begin{Bmatrix} 1 \\ 10, \dots \\ 11 \end{Bmatrix}; S_4 = \begin{Bmatrix} 2 \\ 10, \dots \\ 11 \end{Bmatrix}; S_5 = \begin{Bmatrix} 1 & 3 \\ 2, & 5, \dots \\ 10 & 9 \end{Bmatrix}; S_6 = \begin{Bmatrix} 1 & 3 \\ 2, & 5, \dots \\ 10 & 11 \end{Bmatrix};$$

$$S_7 = \begin{Bmatrix} 1 & 3 \\ 2, & 9, \dots \\ 10 & 11 \end{Bmatrix}; S_8 = \begin{Bmatrix} 1 & 5 \\ 2, & 9, \dots \\ 10 & 11 \end{Bmatrix}; S_9 = \begin{Bmatrix} 1 & 3 & 7 \\ 2, & 5, & 11, \dots \\ 10 & 9 & \end{Bmatrix}; S_{10} = \begin{Bmatrix} 1 & 3 & 4 \\ 2, & 5, & 7, \dots \\ 10 & 11 & 9 \end{Bmatrix}; S_{11} = \begin{Bmatrix} 1 & 3 & 4 & 6 \\ 2, & 5, & 7, & 12, \dots \\ 10 & 11 & 9 & \end{Bmatrix};$$

$$S_{12}=\begin{Bmatrix}1 & 3 & \\ 2, & 9, & 4\\ 10 & 11 & 5\end{Bmatrix},\dots\};S_{13}=\begin{Bmatrix}1 & 5 & \\ 2, & 9, & 3\\ 10 & 11 & 4\end{Bmatrix},\dots\};S_{14}=\begin{Bmatrix}1 & & 9\\ 2, & & 10\\ 11 & & \end{Bmatrix}\dots\};S_{15}=\begin{Bmatrix}1 & 2\\ 10, & 3, & \dots\\ 11 & 4\end{Bmatrix};S_{16}=\begin{Bmatrix}1 & 2\\ 10, & 3, & \dots\\ 11 & 5\end{Bmatrix};$$

$$S_{17}=\begin{Bmatrix}1 & 2\\ 10, & 4, & \dots\\ 11 & 5\end{Bmatrix};S_{18}=\begin{Bmatrix}1 & 3\\ 10, & 4, & \dots\\ 11 & 5\end{Bmatrix};S_{19}=\begin{Bmatrix}1 & 2 & 5\\ 10, & 3, & 6, & \dots\\ 11 & 4 & 9\end{Bmatrix};S_{20}=\begin{Bmatrix}1 & 2 & 5 & \\ 10, & 3, & 6, & 7\\ 11 & 4 & 9 & 8\end{Bmatrix},\dots\};S_{21}=\begin{Bmatrix}1 & 2 & 4\\ 10, & 3, & 7, & \dots\\ 11 & 5 & 9\end{Bmatrix};$$

$$S_{22}=\begin{Bmatrix}1 & 2 & 4 & \\ 10, & 3, & 7, & 6\\ 11 & 5 & 9 & 12\end{Bmatrix},\dots\};S_{23}=\begin{Bmatrix}1 & 2 & 3\\ 10, & 4, & 6, & \dots\\ 11 & 5 & 9\end{Bmatrix};S_{24}=\begin{Bmatrix}1 & 2 & 3 & \\ 10, & 4, & 6, & 7\\ 11 & 5 & 9 & 8\end{Bmatrix},\dots\};$$

$$S_{25}=\begin{Bmatrix}1 & 3 & 2\\ 10, & 4, & 6, & \dots\\ 11 & 5 & 7\end{Bmatrix};S_{26}=\begin{Bmatrix}1 & 3 & 2 & 8\\ 10, & 4, & 6, & 9\\ 11 & 5 & 7 & 12\end{Bmatrix}.$$

Розгалуження проводиться відповідно до значення нижньої межі довжини кожного упорядкування, а саме:

$$\tau(S_1)=\tau(S_2)=\tau(S_3)=\tau(S_4)=\tau(S_5)=\tau(S_6)=\tau(S_7)=\tau(S_8)=4,\tau(S_9)=5,\tau(S_{10})=4,$$
$$\tau(S_{11})=\tau(S_{12})=\tau(S_{13})=\tau(S_{14})=5,\tau(S_{15})=\tau(S_{16})=\tau(S_{17})=\tau(S_{18})=\tau(S_{19})=4,$$
$$\tau(S_{20})=5,\tau(S_{21})=4,\tau(S_{22})=5,\tau(S_{23})=4,\tau(S_{24})=5,\tau(S_{25})=4,\tau(S_{26})=4.$$

Вершина із номером 26 відповідає оптимальному паралельному упорядкуванню, яке має вигляд

$$S^*=\begin{Bmatrix}1 & 3 & 2 & 8\\ 10, & 4, & 6, & 9\\ 11 & 5 & 7 & 12\end{Bmatrix}.$$

1.4.2.1 Оцінки значень цільових функцій в задачах упорядкування

Нехай маємо деякий граф G. Наведемо відомі оцінки значень l та h для *задачі 1* та *задачі 2*, відповідно. Необхідність знаходження оцінок точних значень цільових функцій пов'язана з тим, що, якщо маємо їх точне значення, то задача зводиться до задачі розташування вершин по вже заданих місцях. Така задача є простішою, хоча б тому, що містить менше невідомих (обмеження на можливі місця вершин у шуканому упорядкуванні більш жорсткі, значення цільового фунціоналу детерміноване), і потребує лише знаходження алгоритму відповідного розташування вершин.

Очевидно, що значення l обмежене знизу величиною $\max\left\{\underline{l},\left\lceil\frac{n}{h}\right\rceil\right\}$, де $\lceil\cdot\rceil$ – округлення до більшого цілого, n – кількість вершин у графі, $\underline{l}$ – довжина упорядкування $\underline{S}$, h – ширина упорядкування; це і є найпростіша (базова) оцінка довжини упорядкування l для *задачі 1*. Точне значення цільової функції l можна визначити, якщо граф G є орієнтованим кореневим лісом [14]:

$$l = \max_{1 \le k \le \underline{l}} \left\{ \underline{l}, \left\lceil \frac{1}{h} \sum_{i=k}^{\underline{l}} \left| \overline{S}[\underline{l} - i + 1] \right| \right\rceil + k - 1 \right\}.$$

Проте, якщо граф G довільний, то це співвідношення є уточненою нижньою оцінкою для l.

За допомогою використання обернених графів, у роботі [5] вдалося покращити наведену оцінку наступним чином:

$$l \ge \max \left\{ \underline{l}, \max_{1 \le k \le \underline{l}} \left\{ \left\lceil \frac{1}{h} \sum_{i=k}^{\underline{l}} \left| \bar{S}[\underline{l} - i + 1] \right| \right\rceil + k - 1 \right\}, \max_{1 \le k \le \underline{l}} \left\{ \left\lceil \frac{1}{h} \sum_{i=k}^{\underline{l}} \left| \underline{S}[i] \right| \right\rceil + k - 1 \right\} \right\}.$$

Експериментальні дані підтвердили перевагу цієї оцінки над уточненою, а отже і доцільність її використання.

Найбільш поширеною оцінкою довжини l^* для *задачі 1а* є аналогічна до базової для *задачі 1*, тобто $\max\{\underline{l}, K\}$, де $\underline{l}$ – довжина упорядкування $\underline{S}$ для графу G, а K – мінімальна необхідна кількість місць для розміщення n вершин, тобто яка задовольняє наступному співвідношенню:

$$K = \min_{1 \le k \le n} \left\{ k : \sum_{i=1}^{k} h_i \ge n \right\},$$

де $h_i, i = \overline{1, n}$ – задані місткості відповідних місць паралельного упорядкування.

Для параметра h найпростішою оцінкою, аналогічно до l, буде $\left\lceil \frac{n}{l} \right\rceil$. Цю оцінку також можна уточнити, якщо детальніше дослідити структуру графу G, наступним чином:

$$h \ge \left\lceil \max_{1 \le k \le \underline{l}} \left\{ \frac{1}{k + (l - \underline{l})} \sum_{i=1}^{k} \left| \overline{S}[i] \right| \right\} \right\rceil.$$

Уточненню оцінок параметрів l та h присвячені роботи [84-86]. Проте всі спроби отримати точні значення для l та h поки що не призвели до бажаного успіху.

1.4.3 Точні алгоритми для спеціальних випадків

Алгоритм, заснований на рівневому принципі

Нехай маємо граф G, який є орієнтованим деревом чи лісом, ширина упорядкування h довільна. У цьому випадку для того, щоб побудувати шукане оптимальне упорядкування, можна скористатися алгоритмом, який отримав назву рівневий принцип [14].

Алгоритм рівневого принципу:

1) Вершинам множини $\bar{S}[\,\bar{l}\,]$ ставимо у відповідність позначку 1. Покладаємо $k = \bar{l} - 1$.

2) Допоки $k \neq 0$, вершини з множини $\bar{S}[k]$ позначаємо $\bar{l} - k + 1$ та покладаємо $k \coloneqq k - 1$.

3) На цьому кроці усі вершини вже мають позначки. Переходимо до побудови оптимального упорядкування. Маємо порожнє упорядкування S, покладемо $i = 1$.

4) Якщо G порожній, то кінець алгоритму.

5) Нехай у графі G на цьому кроці алгоритму є r вільних вершин (вершин без вхідних дуг), обираємо з них $\min\{r, h\}$ вершин в порядку незростання позначок і ставимо їх на місце i в упорядкуванні S. Після цього видаляємо обрані вершини з графу разом з вихідними дугами, приймаємо $G \coloneqq G, i \coloneqq i + 1$ та переходимо на 4.

Доведено [14], що упорядкування, побудоване за цим алгоритмом, буде оптимальним. Складність цього алгоритму складає $O(|E| + |V|)$. Він також може бути застосований для графів довільної структури, проте такий алгоритм буде лише наближеним. Алгоритм можна аналогічно адаптувати до *задачі 1а*, вибираючи на кожне i-те місце $\min\{r, h_i\}$ вершин, але він так само буде наближеним.

Приклад 1.3. Застосуємо алгоритм, заснований на рівневому принципі, для розв'язання $P_S(T, h, l)$ для графу з рис. 1.5 та $h = 3$.

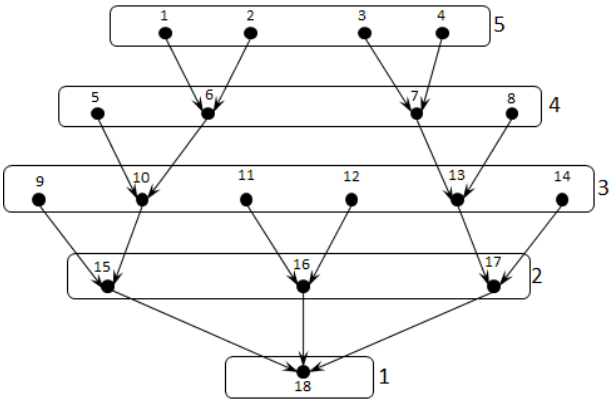

Рис. 1.5. Зображення графу з прикладу 1.3

Відповідно до алгоритму побудуємо спочатку упорядкування $\overline{S}$:

$$\overline{S} = \left\{ \begin{matrix} & & 9 & & \\ 1 & 5 & 10 & & 15 \\ 2 & 6 & 11 & & \\ 3, & 7, & 12, & 16, & 18 \\ 4 & 8 & 13 & & 17 \\ & & 14 & & \end{matrix} \right\}.$$

Тоді вершини отримають наступні мітки: вершина 18 – мітку 1, вершини 15-17 – мітку 2, 9-14 – мітку 3, 5-8 – мітку 4 та 1-4 – мітку 5. Мітки та рівні дерева позначені також на рис. 1.5 прямокутниками.

Переходимо до побудови упорядкування:

$$S^* = \left\{ \begin{matrix} 1 & 4 & 7 & 10 & 13 & & \\ 2, & 5, & 8, & 11, & 14, & 16 & \\ 3 & 6 & 9 & 12 & 15 & 17, & 18 \end{matrix} \right\}.$$

Алгоритм, заснований на лексикографічному помічені

Нехай маємо ациклічний орієнтований граф G та $h = 2$. Побудуємо для нього спеціальні упорядкування $\underline{S}$ та $\overline{S}$, якщо хоча б в одному з них на кожному місці стоїть парна кількість вершин, то алгоритм побудови шуканого оптимального упорядкування є тривіальним: заносимо на кожне вільне місце по дві вершини з відповідної множини $\underline{S}[i]$ чи $\bar{S}[i]$, починаючи з першого, рухаючись зліва направо.

У випадку, коли побудувати оптимальне упорядкування графу за наведеним алгоритмом не вдається, тоді, видаливши з графу транзитивні дуги, можна

застосувати алгоритм, заснований на лексикографічному помічені вершин [87]. Він складається з двох етапів: розстановка поміток та побудова упорядкування.

Етапу розстановки поміток відповідають пункти 1-2 схеми, наведеної нижче, а побудові упорядкування – пункти 3-5.

Введемо допоміжне означення.

Означення 1.5. Лексикографічним порядком називають відношення лінійного порядку на множині слів над деяким упорядкованим алфавітом Σ.

Слово α передує слову β ($\alpha \prec \beta$), якщо:

- або перші $m \geq 0$ символів цих слів співпадають, а $(m+1)$-ий символ слова α менший (відносно заданого у Σ порядку) ніж $(m+1)$-ий символ слова β;
- або слово α є початком слова β.

Далі розглядаються послідовності (слова) над алфавітом, що складається з натуральних чисел, упорядкованих за зростанням.

Алгоритм, заснований на лексикографічному помічені

1) Вершинам множини $\bar{S}[\,\bar{l}\,]$ ставимо у відповідність позначки $1,2,\ldots,|\bar{S}[\,\bar{l}\,]|$ у довільному порядку. Покладаємо $k = \bar{l} - 1$.

2) Допоки $k \neq 0$, для кожної вершини з множини $\bar{S}[k]$ будуємо спадну послідовність з позначок безпосередніх послідовників, і присвоюємо їм помітки у порядку лексикографічного неспадання відповідних послідовностей, починаючи з мінімального вільного номера позначки. Покладаємо $k := k - 1$.

3) На цьому кроці усі вершини вже мають позначки. Переходимо до побудови оптимального упорядкування. Маємо порожнє упорядкування S, покладемо $i = 1$.

4) Якщо G порожній, то кінець алгоритму.

5) Нехай у графі G на цьому кроці алгоритму є r вільних вершин, обираємо з них $\min\{r, 2\}$ вершин з найбільшими позначками, і ставимо їх на місце i в упорядкуванні S. Після цього видаляємо ці вершини з графу разом з вихідними дугами, і приймаємо $G := G, i := i + 1$ та переходимо на 4.

У [87] доведено, що отримане за цим алгоритмом упорядкування буде оптимальним. Алгоритм має складність $O(|E| + |V| \log|V|)$. Цей алгоритм можна застосовувати для довільного значення ширини упорядкування h. Він буде відрізнятися тим, що у пункті 5 потрібно обирати $\min\{r, h\}$ вершин з найбільшими позначками, проте такий алгоритм буде лише наближеним. Алгоритм можна аналогічно адаптувати до *задачі 1а*, вибираючи на кожне i-те місце $\min\{r, h_i\}$, але він так само в загальному випадку буде наближеним. У [88] доведено, що він є точним для *задачі 1а* у випадку квазі-інтервальних та над-інтервальних графів для довільних місткостей упорядкування.

Приклад 1.4. Застосуємо алгоритм, заснований на лексикографічному помічені, для розв'язання $P_S(G, 2, l)$ для графу з рис. 1.6.

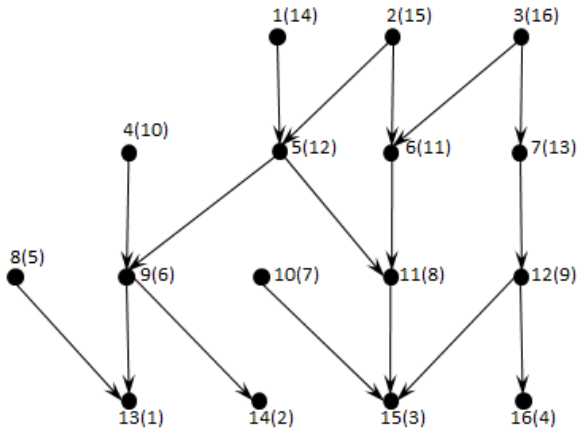

Рис. 1.6. Зображення графу з прикладу 1.4

Відповідно до алгоритму розпочнемо з розставлення міток. Спочатку у довільному порядку помітимо вершини без вихідних дуг, нехай, наприклад, вершина 13 матиме мітку 1, вершина 14 – мітку 2, 15 – мітку 3, а 16 – мітку 4. Для того, щоб визначити мітки для вершин 8-12 складемо для них неспадні послідовності міток безпосередніх послідовників: для вершини 8 отримаємо (1), для вершини 9 – (2, 1), для 10 – (3), для 11 – (3), для 12 – (4, 3). Отже, маємо $(4,3) \succ (3) \succcurlyeq (3) \succ (2,1) \succ (1)$, тоді вершина 8 отримає мітку 5, вершина 9 – мітку 6, 10 – мітку 7, 11 – мітку 8 та 12 – мітку 9. Для наступного рівня отримаємо: 4 – (6), 5 – (8, 6), 6 – (8), 7 – (9), тоді, оскільки $(9) \succ (8,6) \succ (8) \succ (6)$, вершина 4 отримає мітку 10, вершина 5 – 11, 6 –

мітку 12 та 7 – мітку 13. Аналогічно для останнього рівня: 1 – (12), 2 – (12, 11), 3 – (13, 11). Маємо $(13, 11) \succ (12{,}11) \succ (12)$, а отже вершина 1 отримає мітку 14, 2 – 15, а 3 – мітку 16.

Всі вершини помічені, тому можемо переходити до побудови упорядкування:

$$S^* = \begin{Bmatrix} 2 & 1 & 5 & 4 & 10 & 8 & 15 & 13 \\ 3' & 7' & 6' & 12' & 11' & 9' & 16' & 14 \end{Bmatrix}.$$

Алгоритм, заснований на максимальному паросполученні

Нехай маємо довільний ациклічний орієнтований граф G та $h = 2$. У цьому випадку для того, щоб побудувати шукане оптимальне упорядкування можна скористатися алгоритмом, що був історично першим алгоритмом побудови оптимальних упорядкувань для $h = 2$. Він є складнішим, аніж алгоритм, заснований на лексикографічному помічені, проте не потребує видалення транзитивних дуг з графу. Такий алгоритм побудови упорядкування S [89] у подальшому називатимемо алгоритмом, заснованим на максимальному паросполученні.

Алгоритм, заснований на максимальному паросполученні:

1) Для графу $G(V, U)$ будуємо неорієнтований граф $\overline{G}(V, E)$, де $(i, j) \in E$, якщо у графі G немає шляху ані з вершини i у j, ні з j у i. Такий граф у подальшому будемо називати графом досяжності.

2) В отриманому графі $\overline{G}$ знаходимо максимальне паросполучення, яке позначимо $M \subseteq E$, тобто підмножину ребер E максимальної потужності, що не мають спільних вершин.

3) В шуканому упорядкуванні S покладаємо всі місця вільними і $k = 1$.

4) Якщо G порожній, то кінець алгоритму.

5) Можливий один з наступних випадків:

а) Серед відкритих вершин графу G існує така, яка не належить жодному з ребер у M, тоді обираємо для розташування її.

б) У множині M існує таке ребро (i, j), що вершини i та j є відкритими, тоді обираємо їх для розташування і видаляємо (i, j) з M.

в) У множині M знайдеться така пара ребер (i,p) та (j,q), що вершини i та j є відкритими, а між вершинами p та q є ребро у графі $\overline{G}$, тоді для розташування обираємо вершини i та j, ребра (i,p) та (j,q) видаляємо з M, а ребро (p,q) додаємо до M.

6) Обрані вершини розташовуємо на місце k в упорядкуванні S та видаляємо їх з графу G разом з вихідними дугами, якщо вони є. Приймаємо $G \coloneqq G, k \coloneqq k+1$ та переходимо на 4.

Важливо зазначити, що для знаходження M для довільних неорієнтованих графів відомі алгоритми поліноміальної складності, наприклад алгоритм стиснення квіток [90].

У [89] доведено, що отримане за цим алгоритмом упорядкування буде оптимальним. Алгоритм має складність $O(|V|^3)$. Його застосування для довільних значень ширини упорядкування h без попередніх модифікацій є проблематичним через специфічність операцій у пункті 5, а отже і для *задачі 1а*.

Приклад 1.5. Застосуємо алгоритм, заснований на максимальному паросполученні, для розв'язання $P_S(G,2,l)$ для графу з рис. 1.7.

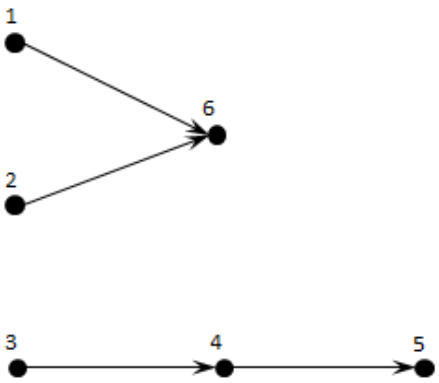

Рис. 1.7. Зображення графу з прикладу 1.5

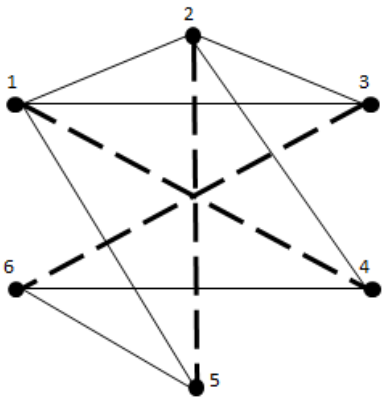

Рис. 1.8. Граф досяжності для графу з рис. 1.7

Відповідно до алгоритму розпочнемо з побудови графу досяжності. Вершина 1 буде з'єднана з усіма іншими, окрім вершини 6; аналогічно вершина 2, з вершини 3 існують шляхи лише до вершин 4 та 5, тому вона буде з'єднана з усіма вершинами, окрім них; аналогічно з вершини 4 будуть йти ребра до усіх вершин, окрім 3 та 5;

з 5 – до усіх, окрім 3 та 4; а з 6 – до усіх, окрім 1 та 2. Тепер побудуємо максимальне паросполучення M. Зрозуміло, що $|M| \leq 3$. Оберемо, наприклад, таке $M = \{(1,4), (2,5), (3,6)\}$. Отриманий граф досяжності та обране максимальне паросполучення можна побачити на рис. 1.8.

Переходимо до побудови упорядкування S^*.

Відкритими є вершини 1, 2 та 3. Бачимо, що всі відкриті вершини належать до якогось з елементів множини M, оскільки вона покриває усі вершини графу. Також жодне з ребер не містить деякі дві з відкритих вершин. Таким чином маємо розглядати усі пари ребер: пару $(1,4), (2,5)$ обрати не можемо, оскільки вершини 4 та 5 не з'єднані у графі досяжності; для пар $(1,4), (3,6)$ та $(2,5), (3,6)$ необхідні умови виконуються, тому можемо обрати будь-яку з них. Оберемо, наприклад, пару ребер $(1,4), (3,6)$, тоді ребра $(1,4)$ та $(3,6)$ видаляємо з M, а $(4,6)$ додаємо. Ставимо вершини 1 та 3 на перше місце в S^*, видаляємо вершини з G разом з вихідними дугами.

Тепер відкритими є вершини 2 та 4, M має вигляд $M = \{(2,5), (4,6)\}$. Аналогічно, не можемо обрати одне ребро, тому розглянемо пару ребер $(2,5), (4,6)$. Вона задовольняє усім потрібним вимогам: з вершини 5 не існує шляху у 6 та навпаки. Тому обрану пару ребер з M видаляємо, пару $(5,6)$ додаємо, вершини 2 та 4 ставимо на друге місце в S^* та видаляємо з G разом з вихідними дугами.

Залишилися відкритими вершини 5 та 6, $M = \{(5,6)\}$. У M єдине ребро, в якого обидві вершини є відкритими, тому ставимо їх на третє місце у S^* та видаляємо їх з G. Отримали порожній граф, а отже алгоритм завершений.

Отримали наступне оптимальне упорядкування:

$$S^* = \begin{Bmatrix} 1 & 2 & 5 \\ 3' & 4' & 6 \end{Bmatrix}.$$

Алгоритм побудови HLF упорядкування

У роботі [91] автор приділяє увагу тому, що для всіх відомих точних алгоритмів знаходження оптимального упорядкування при $h = 2$ попередня обробка графу (пошук максимального паросполучення, транзитивне замикання та транзитивне

спрощення) має більшу алгоритмічну складність, аніж самі алгоритми, які зазвичай мають складність близьку до лінійної.

Запропонований алгоритм побудови HLF (highest-level-first, спочатку найвищій рівень) упорядкування, який є особливим випадком рівневого принципу. Відповідно до рівневого принципу вершини графу розміщуються в упорядкуванні рівень за рівнем. Його основною відмінністю є вибір вершини для розміщення разом з останньою вершиною рівня. Вершини обираються таким чином, аби послідовність, що складається з рівнів цих вершин, була лексикографічно найбільшою. Це досягається за допомогою двоетапного алгоритму: перший етап довільним чином обирає вершини для розміщення, на другому етапі виправляються помилки попереднього етапу шляхом підміни вершин.

Доведено, що цей алгоритм знаходить оптимальний розв’язок, має майже лінійну складність та є найкращою зі стратегій, заснованих на рівневому принципі. Для $h \geq 3$ алгоритм є наближеним.

1.5 Огляд наближених методів розв’язання задач паралельного упорядкування

Для алгоритму, заснованого на рівневому принципі, відома наступна оцінка похибки зверху у випадку застосування його до довільного графу G [41]:

$$\frac{l_A(S)}{l(S^*)} \leq \frac{2h}{h+1}.$$

Її досяжність, проілюструємо на прикладі графу з рис. 1.9 при $h = 2$.

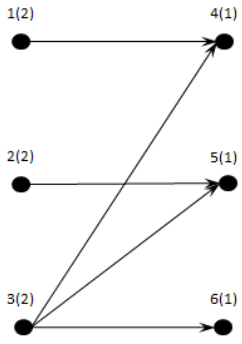

Рис. 1.9. Зображення графу з прикладу досяжності оцінки для алгоритму, заснованого на рівневому принципі

Побудуємо спочатку упорядкування використовуючи алгоритм, заснований на рівневому принципі. Вершини 4-6 мають мітки 1, а вершини 1-3 – мітки 2. Тоді, оскільки вершини з однаковими мітками обираємо довільним чином, можемо отримати наступне упорядкування:

$$S = \left\{\begin{matrix}1\\2\end{matrix}, 3, \begin{matrix}4\\5\end{matrix}, 6\right\}, l_A(S) = 4.$$

Проте оптимальним для цього графу буде наступне упорядкування:

$$S^* = \left\{\begin{matrix}1 & 2 & 5\\3, & 4, & 6\end{matrix}\right\}, l(S^*) = 3.$$

З цього випливає, що $\frac{l_A(S)}{l(S^*)} = \frac{2*2}{2+1} = \frac{4}{3}$, а отже оцінка досягається.

Відома оцінка зверху помилки у випадку застосування алгоритму, заснованого на лексикографічному помічені, до довільної *задачі 1* [41]:

$$\frac{l_A(S)}{l(S^*)} \leq 2 - \frac{2}{h}.$$

Ця оцінка є досяжною при $h = 3$ та асимптотично досяжною для $h \geq 4$, проілюструємо це на прикладі графу з рис. 1.10 при $h = 3$.

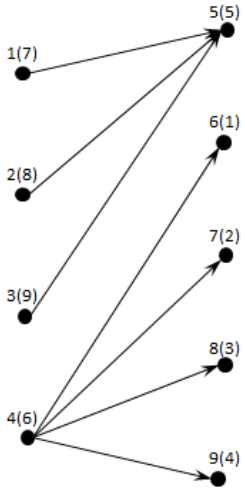

Рис. 1.10. Зображення графу з прикладу досяжності оцінки
для алгоритму, заснованого на лексикографічному помічені

Побудуємо спочатку упорядкування, використовуючи алгоритм, заснований на лексикографічному помічені. Вершини 6-9 мають мітки 1-4 відповідно, а вершина 5

– мітку 5. Для того, щоб визначити помітки для вершин 1-4 складемо для них спадні послідовності міток безпосередніх послідовників: для вершин 1-3 отримаємо (5), для вершини 4 – (4, 3, 2, 1). Отже, маємо $(5) \succcurlyeq (5) \succcurlyeq (5) \succ (4,3,2,1)$, тоді вершина 4 отримає мітку 6, вершина 1 – мітку 7, 2 – мітку 8, 3 – мітку 9. Отримаємо наступне упорядкування:

$$S = \begin{Bmatrix} 1 & 4 & 6 \\ 2, & 5, & 7, 9 \\ 3 & & 8 \end{Bmatrix}, l_A(S) = 4.$$

Проте оптимальним для цього графу буде наступне упорядкування:

$$S^* = \begin{Bmatrix} 1 & 3 & 5 \\ 2, & 6, & 8 \\ 4 & 7 & 9 \end{Bmatrix}, l(S^*) = 3.$$

З цього випливає, що $\frac{l_A(S)}{l(S^*)} = 2 - \frac{2}{3} = \frac{4}{3}$, а отже оцінка досягається.

Зазначимо, що за іншого вибору поміток вершин останнього рівня отримане упорядкування могло бути оптимальним. Наприклад, за такогоє: вершини 6-9 мають мітки 2-5 відповідно, а вершина 5 – мітку 1. Така помилка – помилка через довільність вибору міток останнього рівня, окрім цієї можливою також є помилка методу: коли не існує такого вибору міток останнього рівня, за якого отримане упорядкування буде оптимальним.

Відмітимо, що обидва алгоритми мають асимптотичну похибку 2 при застосуванні до довільної *задачі 1*, тобто довжина упорядкування, отримана за рівневим принципом може бути майже в 2 рази більшою, ніж оптимальна. До їх переваг можна віднести швидкість і алгоритмічну простоту.

Можна також довести наступне твердження.

Твердження 1. Для будь-якого алгоритму А, що не залишає порожніх місць в упорядкуванні при наявності відкритих вершин у графі, виконується

$$\frac{l_A(S)}{l(S^*)} \leq 2 - \frac{1}{h}.$$

Доведення. Нехай деяке упорядкування вершин графу G з довжиною, рівною довжині критичного шляху, містить k_i рівнів, для розташування вершин яких необхідно принаймні i місць, $i = \overline{1,m}$. Позначимо k – коефіцієнт пропорційності. Тоді з необхідності досяжності оцінки природнім чином випливають дві умови.

По-перше, якщо в упорядкуванні більше аніж $h * k$ рівнів, то не зможемо побудувати оптимальне упорядкування довжини $h * k$, оскільки довжина упорядкування дорівнює довжині критичного шляху. По-друге, якщо для цього упорядкування оцінка зверху його довжини менша аніж $(2h - 1) * k$, то не зможемо досягти оцінки, не залишаючи штучно деякі місця порожніми.

Ці умови легко подати у вигляді системи лінійних алгебраїчних нерівностей:

$$\begin{cases} k_1 + k_2 + k_3 + \cdots + k_m \le h * k, \\ k_1 + 2k_2 + 3k_3 + \cdots + m * k_m \ge (2h - 1) * k. \end{cases}$$

Проведемо низку перетворень з цією системою:

$$\begin{cases} k_1 + k_2 + k_3 + \cdots \le h * k, \\ k_1 + 2k_2 + 3k_3 + \cdots \ge (2h - 1) * k, \end{cases} \Rightarrow \begin{cases} -k_1 - k_2 - k_3 - \cdots \ge -h * k, \\ k_1 + 2k_2 + 3k_3 + \cdots \ge (2h - 1) * k, \end{cases}$$

$$\begin{cases} -k_1 - k_2 - k_3 - \cdots \ge -h * k, \\ k_2 + 2k_3 + \cdots \ge (h - 1) * k, \end{cases} \Rightarrow \begin{cases} k_1 + 2k_2 + 3k_3 + \cdots \ge (2h - 1) * k, \\ (h-1)k_2 + 2(h-1)k_3 + \cdots \ge (h-1)^2 * k, \end{cases}$$

$$\begin{cases} k_1 + 2k_2 + 3k_3 + \cdots \ge (2h - 1) * k, \\ k_1 + (h+1)k_2 + (2h+1)k_3 + \cdots \ge (h^2 - 2h + 1 + 2h - 1) * k = h^2 * k. \end{cases}$$

Помітимо, що оскільки довжина оптимального упорядкування дорівнює $h * k$, то граф G повинен містити не більше $h^2 * k$ вершин (усі $h * k$ місць заповнені повністю).

У останній нерівності зліва стоять k_i з коефіцієнтами, що дорівнюють мінімальній кількості вершин на рівні, щоб для їх розташування знадобилося i місць, отже це мінімальна кількість вершин у графі. І ця величина більша або дорівнює максимальній кількості вершин у графі, а отже остання нерівність виконується як рівність, а тоді й усі інші нерівності виконуються як рівності:

$$\begin{cases} k_1 + k_2 + k_3 + \cdots + k_m = h * k, \\ k_1 + 2k_2 + 3k_3 + \cdots + m * k_m = (2h - 1) * k. \end{cases}$$

З цього випливає, що оцінка зверху не може перевищити $(2h-1)*k$, а отже оцінка не може бути перевищена будь-яким алгоритмом, що не залишає порожніх місць в упорядкуванні при наявності відкритих вершин у графі. ■

З доведення випливає також, що для того, щоб оцінка була досяжною для графу, необхідно, щоб довжина його критичного шляху дорівнювала $h*k$, а кількість вершин на кожному місці в одному з його упорядкувань має давати остачу 1 при діленні на h. Також останню систему можна застосовувати для знаходження k_i.

Помітимо також, що оцінки точності для алгоритмів, заснованих на рівневому принципі та лексикографічному помічені, є строго меншими аніж ця оцінка точності, а отже таким чином врахована інформація завжди дає кращий результат.

У статі [92] автори запропонували алгоритм, що покращує оцінку точності до $2-\frac{7}{3h+1}$ для ширини упорядкування $h>3$. Алгоритм складається з двох основних складових. Перша складова заснована на розрахунку модифікованих директивних строків з [58] (див. підрозділ 1.3). Вони обчислюються за формулою $\mathrm{D}(v)=\min\limits_{L,D}\left\{D-L-\left\lceil\frac{|N(v,D,L)|}{h}\right\rceil\right\}$, де $N(v,D,L)$ – найбільша множина вершин u, довжина шляху до яких від вершини v складає принаймні $L+1$, для яких $D(u)\leq D$. Легко побачити, що $g(v,D)=N(v,D,0)$. Друга складова пов'язана з переупорядкуванням попередників вершини, що розміщується, які стоять на поточному та попередньому місцях в упорядкуванні. Доводиться, що цей алгоритм знаходить оптимальне упорядкування для $h=2$, для $h=3$ оцінка точності складає $2-\frac{2}{h}$, а для $h>3$ алгоритм покращує її до $2-\frac{7}{3h+1}$. Також доводиться, що існують графи, для яких оцінка $2-\frac{2}{h}$ є досяжною для алгоритму, заснованого на лексикографічному помічені.

Значне просування протягом останніх років здійснено в напрямку побудови $(1+\varepsilon)$-наближених алгоритмів, що мають квазіполіноміальну складність [93–96]. Також ведеться активна розробка алгоритмів, які засновані на метаевристиках, таких як Баєсова оптимізація та генетичні алгоритми [34,35].

Відмітимо, що перелік розглянутих у розділі задач, алгоритмів та методів теорії розкладів не є вичерпним, інші постановки задач та підходи до їх розв'язання представлені у [97].

1.6 Висновки до розділу

Простота постановок задач привертає увагу науковців до їх дослідження. До фундаментальних результатів слід віднести наступні точні алгоритми: метод гілок та меж, що дає точні розв'язки задачі побудови оптимального паралельного упорядкування мінімальної довжини при заданій ширині для довільного ациклічного графу; алгоритм поліноміальної складності, заснований на рівневому принципі, який є точним для орієнтованих лісів та застосовується як евристика для довільних орграфів; точні алгоритми розв'язання задачі для часткового випадку при ширині упорядкування $h = 2$.

Проте багато питань є невирішеними, зокрема:

- Залишається недостатньо дослідженою задача побудови паралельного упорядкування мінімальної ширини при заданій довжині.
- Уточнення нижніх оцінок довжини та ширини упорядкування. Відомо, що в задачах оптимізації інколи розглядають одну з двох задач: знаходження оптимального розв'язку та відповідного значення цільової функції або лише оптимального значення цільової функції. Якби для задач, що вивчаються, можна було отримати розв'язок другої задачі, то питання зводилося б лише до розробки алгоритмів, що будують сам розв'язок. Зрозуміло, що в схемах напрямленого перебору в цьому випадку можна було б відкидати непридатні варіанти раніше, ніж в класичній схемі методу гілок та меж. Оскільки точних значень цільової функції отримати поки що не вдалося, то актуальним є саме уточнення нижніх оцінок для цих функцій.
- Додаткового дослідження потребують також інші підходи до скорочення перебору методом гілок та меж. Одним з перспективних підходів може виступити видалення розгалужень, що відповідають ізоморфним підграфам, утвореним при

видаленні деяких відкритих вершин. Це є важливим через те, що всі такі підграфи матимуть однакові значення оцінки цільової функції. Оскільки поліноміальний алгоритм перевірки на ізоморфізм досі не був отриманий, то для можливості застосування цього підходу особливо важливим є дослідження особливостей підграфів, що утворюються саме у випадку видалення відкритих вершин, зокрема за наявності автоморфізму, та для конкретних класів графів. Цікавою альтернативою також є наближене визначення ізоморфності за допомогою потужних інваріантів, таких як алгоритм уточнення кольорів.

- Виділення підкласів графів, для яких або відомі точні алгоритми поліноміальної складності дають оптимальні розв’язки, або розробка нових ефективних алгоритмів для таких підкласів.
- Дослідження питання, що стосується результатів застосування до довільних орграфів алгоритмів поліноміальної складності, що є точними для певних підкласів.
- Аналіз задач, що більш точно моделюють реальні процеси, та використання відомого математичного апарату до їх розв’язання, зокрема у випадках неуніверсальності виконавців чи неповної зайнятості.

Саме ці питання більш глибоко досліджено у даній роботі.

Основні матеріали розділу опубліковані у [1,2,4,5,10-13].

Розділ 2. РЕЗУЛЬТАТИ ДЛЯ КЛАСИЧНИХ ЗАДАЧ ПАРАЛЕЛЬНОГО УПОРЯДКУВАННЯ

2.1 Зв'язок між різними випадками класичної задачі упорядкування

Розглянемо деякі твердження та наслідки з них, що дозволять звузити множину задач паралельного упорядкування, які потребують розгляду.

Означення 2.1. До класу графів, для яких існують щільні упорядкування, D_h належать такі графи, які для заданого значення ширини упорядкування h, мають упорядкування, в яких на кожному місці стоїть h вершин, а отже вони є оптимальними.

Твердження 2. Якщо існує точний алгоритм A розв'язання *задачі 1* поліноміальної складності для деякої фіксованої ширини упорядкування $\tilde{h}$, тоді існує поліноміальний алгоритм і для будь-якого $h' < \tilde{h}$.

Доведення. Нехай маємо деякий непорожній граф G, що має n вершин, та деяке фіксоване значення $h' < \tilde{h}$. Зрозуміло, що довжина його оптимального упорядкування l^* буде знаходитися у проміжку від 1 (якщо $n \leq h'$ та всі вершини є ізольованими) до n (граф є ланцюжком) включно.

Побудуємо новий граф G', який отримаємо з графу G шляхом додання до нього $(\tilde{h} - h')$ ланцюжків деякої довжини $l \in \overline{1,n}$. Побудуємо тепер упорядкування S графу G' при $h = \tilde{h}$ за алгоритмом A. Відомо, що воно буде оптимальним і може бути отримане за поліноміальний час.

Зрозуміло, що якщо $l = l^*$, тоді довжина S теж буде l^*. З одного боку, вона не може бути більшою за l^*, оскільки маємо $(\tilde{h} - h')$ ланцюжків, які займають по $(\tilde{h} - h')$ позицій на кожному з l^* місць в упорядкуванні, а вершини, що належать графу G, можна розмістити на l^* місцях при $h = h'$. А з іншого, довжина S не може бути меншою, аніж l^*, оскільки ланцюжки можемо розміщувати лише у лінію, а їх довжина l^*. Отже, якщо видалимо з S усі вершини, що належать ланцюжкам, то отримаємо оптимальне упорядкування S^* для вихідної задачі за поліноміальний час, оскільки додання та видалення ланцюжків можна виконати за лінійний час.

Отже, якщо зможемо знаходити довжину ланцюжків $l = l^*$, яка у загальному випадку є невідомою, за поліноміальний час, то зможемо використовувати алгоритм A для знаходження оптимальних упорядкувань для $h' < \tilde{h}$. Це можна зробити за допомогою бінарного пошуку.

Помітимо, що $l = l^*$ відповідає мінімальному значенню l, для якого в отриманому упорядкуванні S за алгоритмом A для графу G' при $h = \tilde{h}$ на всіх місцях буде знаходитись по $(\tilde{h} - h')$ вершин ланцюжків. Дійсно, для $l > l^*$ довжина S буде співпадати з l, оскільки потрібно розмістити ланцюжки, а вершини графу G можна розмістити й на меншій кількості місць. Для $l < l^*$ у S будуть місця, де не буде $(\tilde{h} - h')$ вершин, що належать ланцюжкам, інакше вершини граф G можна було б розмістити на меншу кількість місць, аніж l^* при $h = h'$.

Застосуємо бінарний пошук до розв'язання цієї задачі. Оберемо деяке значення $l_0 \in \overline{a_0, b_0} = \overline{1, n}$, що розбиває цей проміжок приблизно навпіл. Будуємо граф G', для нього за алгоритмом A знаходимо оптимальне упорядкування S при $h = \tilde{h}$. Якщо у S на кожному місці стоїть $(\tilde{h} - h')$ вершин, що належать ланцюжкам, то обираємо, аналогічно, $l_1 \in \overline{a_1, b_1} = \overline{1, l_0}$ та продовжуємо пошук на цьому проміжку. Якщо ж у S є місця, на яких не стоїть $(\tilde{h} - h')$ вершин ланцюжків, то знаходимо $l_1 \in \overline{a_1, b_1} = \overline{l_0 + 1, n}$ та аналогічно продовжуємо пошук. Збіжність алгоритму випливає зі збіжності бінарного пошуку для монотонних функцій. Пошук потребує логарифмічного часу, перевірка може бути виконана за лінійний час, а отже знайдемо шукане $l = l^*$ за поліноміальний час.

Оскільки усі кроки описаного алгоритму можна виконати за поліноміальний час, алгоритм A має поліноміальну складність за умовою, то таким чином отримали алгоритм, що є точним та має поліноміальну складність для будь-якого $h' < \tilde{h}$. ■

Зауваження. Якби алгоритм A, гарантував, що всі вершини в отриманому упорядкуванні розташовуються якомога раніше, то було б достатньо лише раз додати до графу ланцюжки довжиною $l = n$ та видалити їх з отриманого упорядкування, щоб отримати шукане.

Наслідок 1 з твердження 2. Якщо для якоїсь фіксованої ширини упорядкування $\tilde{h}$ не існує точного алгоритму розв'язання *задачі 1* поліноміальної складності, тоді не існує алгоритму поліноміальної складності і для будь-якого значення $h' > \tilde{h}$.

Доведення. Випливає безпосередньо з *твердження 2*, оскільки якби існував алгоритм поліноміальної складності для якогось $h' > \tilde{h}$, то він би існував і для $\tilde{h}$, що суперечить передумові наслідку. ■

Наслідок 2 з твердження 2. Якщо для деякого класу графів C, для яких додання ланцюжків до графу не виводить за межі цього класу, існує точний алгоритм розв'язання *задачі 1* поліноміальної складності для деякої фіксованої ширини упорядкування $\tilde{h}$, тоді існує поліноміальний алгоритм і для будь-якого $h' < \tilde{h}$.

Доведення. Безпосередньо випливає з *твердження 2* і того, що додання ланцюжків не виводить граф за межі класу. ■

Твердження 3. Якщо існує точний алгоритм A розв'язання *задачі 1* поліноміальної складності для деякої фіксованої ширини упорядкування $\tilde{h}$ та графів з $D_{\tilde{h}}$ (для яких існують щільні упорядкування ширини $\tilde{h}$), тоді існує поліноміальний алгоритм і для довільного графу при $h = \tilde{h}$.

Доведення. Нехай маємо деякий непорожній граф G, що має n вершин. Зрозуміло, що довжина його оптимального упорядкування l^* буде знаходитися у проміжку від 1 (якщо $n \leq h'$ та всі вершини є ізольованими) до n (граф є ланцюжком) включно.

Побудуємо новий граф G', який отримаємо з графу G шляхом додання до нього $m = k * \tilde{h} + r$ ізольованих вершин, де $r = \tilde{h} * \left\lceil \frac{n}{\tilde{h}} \right\rceil - n$, k – деякий множник. Значення цього множника не менше від 0 (якщо до G достатньо додати $r < \tilde{h}$ вершин, щоб отримати щільне упорядкування, тобто для графів, для яких існують оптимальні упорядкування, де r вільних позицій). Найбільшого значення він досягає у випадку графу-ланцюжка, коли маємо $n * (\tilde{h} - 1) = k * \tilde{h} + r = k * \tilde{h} + \tilde{h} * \left\lceil \frac{n}{\tilde{h}} \right\rceil - n \Longrightarrow k = n - \left\lceil \frac{n}{\tilde{h}} \right\rceil$. Отже, остаточно отримали, що $k \in \overline{0, n - \left\lceil \frac{n}{\tilde{h}} \right\rceil}$. Побудуємо тепер

упорядкування S графу G' при $h = \tilde{h}$ за алгоритмом A. Відомо, що воно, якщо для G' існує щільне упорядкування, буде оптимальним і може бути отримане за поліноміальний час.

Зрозуміло, що якщо $m = l^* * \tilde{h} - n$, тоді S буде щільним, а його довжина буде l^*. З одного боку вона не може бути меншою за l^*, оскільки для розміщення вершин графу G потрібно принаймні l^* місць. З іншого боку, вона не може бути більшою, аніж l^*, оскільки в оптимальному упорядкуванні G буде $l^* * \tilde{h} - n$ вільних місць, які можна зайняти ізольованими вершинами, а алгоритм A знаходить оптимальне щільне упорядкування, якщо воно існує. Тоді, якщо видалимо з S всі додані ізольовані вершини, то отримаємо оптимальне упорядкування S^* для вихідної задачі за поліноміальний час, оскільки додання та видалення ізольованих вершин можна виконати за лінійний час.

Аналогічно, доведенню *твердження 2*, якщо зможемо знаходити кількість ізольованих вершин $m = l^* * \tilde{h} - n$ або, що те саме, значення множника k, для якого досягається відповідне значення, за поліноміальний час, то зможемо використовувати алгоритм A для знаходження оптимальних упорядкувань для будь-яких графів при $h = \tilde{h}$.

Помітимо, що $m = l^* * \tilde{h} - n = k * \tilde{h} + r$ відповідає мінімальному значенню k, для якого отримане упорядкування S за алгоритмом A для графу G' при $h = \tilde{h}$ буде щільним. Дійсно, для менших значень k отримаємо нещільне упорядкування, оскільки додали недостатньо ізольованих вершин. Для більших значень k отримаємо також щільне упорядкування, оскільки матимемо кількість вершин, що кратна $\tilde{h}$.

Застосуємо бінарний пошук до розв'язання цієї задачі. Оберемо деяке значення $k_0 \in \overline{a_0, b_0} = \overline{0, n - \left\lceil \frac{n}{\tilde{h}} \right\rceil}$, що розбиває цей проміжок приблизно навпіл. Будуємо граф G' для цього значення k, за алгоритмом A знаходимо упорядкування S при $h = \tilde{h}$. Якщо S є щільним, то обираємо, аналогічно, $k_1 \in \overline{a_1, b_1} = \overline{0, k_0}$ та продовжуємо пошук на цьому проміжку. Якщо ж у S є вільні позиції, то знаходимо $k_1 \in \overline{a_1, b_1} =$

$\overline{k_0 + 1, n - \left\lceil \frac{n}{\tilde{h}} \right\rceil}$ та аналогічно продовжуємо пошук. Збіжність алгоритму випливає зі збіжності бінарного пошуку для монотонних функцій. Пошук потребує логарифмічного часу, перевірка може бути виконана за лінійний час, а отже знайдемо шукане m за поліноміальний час.

Оскільки усі кроки описаного алгоритму можна виконати за поліноміальний час, алгоритм A має поліноміальну складність за умовою, то отримали алгоритм, що є точним та має поліноміальну складність для будь-якого графу G при $h = \tilde{h}$. ■

Зауваження. Якби алгоритм A гарантував, що всі ізольовані вершини в отриманому упорядкуванні розташовуються якомога пізніше, то було б достатньо лише раз додати до графу m ізольованих вершин, що відповідає значенню $k = n - \left\lceil \frac{n}{\tilde{h}} \right\rceil$, видалити їх з отриманого упорядкування і, можливо, перемістити вперед ізольовані вершини вихідного графу, щоб отримати шукане.

Наслідок з твердження 3. Якщо існує точний алгоритм A розв'язання *задачі 1* поліноміальної складності для деякої фіксованої ширини упорядкування $\tilde{h}$ та графів з $D_{\tilde{h}}$ (для яких існують щільні упорядкування ширини $\tilde{h}$), тоді існує поліноміальний алгоритм і для довільного графу при будь-якому $h' \leq \tilde{h}$.

Доведення. Випливає безпосередньо з *тверджень 2 та 3*. ■

Отриманий наслідок дозволяє узагальнювати алгоритми, отримані для класу графів, для яких існують щільні упорядкування, на випадок всіх графів та менших h. Зокрема дозволяє застосовувати алгоритми, отримані до парних значень h, для менших непарних. Це виявляється зручним при узагальненні алгоритму, заснованого на максимальному паросполученні, якому присвячений підрозділ 2.2.

Відзначимо окремо, що міркування, використані в доведенні, можна застосувати до зведення *задачі 1* до *задачі 2* та навпаки. Справді, оптимальна ширина упорядкування h^* в задачі $P_S(G, l, h)$ є найменшим значенням ширини h, за якого довжина оптимального упорядкування l^* задачі $P_S(G, h, l)$ не перевищує l. І навпаки: оптимальна довжина упорядкування l^* задачі $P_S(G, h, l)$ є найменшим значенням довжини l, за якого ширина оптимального упорядкування h^* в задачі $P_S(G, l, h)$ не

перевищує h. Такий зв’язок можна розглядати як прояв взаємної двоїстості між цими задачами.

2.2 Узагальнення алгоритму, заснованого на максимальному паросполученні

У підрозділі 1.4 розглянули відомі точні алгоритми поліноміальної складності розв’язання задач оптимального упорядкування. Відмічалося, що алгоритми, засновані на рівневому принципі та лексикографічному помічені, можуть бути легко узагальнені на всю множину задач паралельного упорядкування, проте стають лише наближеними. В той же час алгоритм, заснований на максимальному паросполученні, важко узагальнити на випадок довільних класичних задач паралельного упорядкування через специфічність процедури обрання вершин на кожному кроці.

У підрозділі 2.1 розглянуті твердження, що дозволяють звузити множину класичних задач паралельного упорядкування, які потребують розгляду, до класу графів, для яких існують щільні упорядкування при парних значеннях ширини упорядкування. У цьому підрозділі теоретично та експериментально обґрунтовано доцільність використання до цього класу графів алгоритму, заснованого на максимальному паросполученні.

2.2.1 Про можливість знаходження алгоритмом оптимальних упорядкувань

У цьому пункті розглянуті твердження, що доводять доцільність використання алгоритму, заснованого на максимальному паросполученні, до класу графів, для яких існують щільні упорядкування для парної ширини.

У подальшому викладенні буде використане поняття «кліки неорієнтованого графу» у класичному сенсі.

Означення 2.2. Кліка неорієнтованого графу – це підмножина його вершин, в якій кожні дві вершини з’єднані ребром [98].

Твердження 4. З існування щільного упорядкування для деякого графу G для ширини упорядкування $h = \tilde{h}$ випливає існування у графі досяжності неперетинних клік розмірності $\tilde{h}$, що покривають усі вершини графу.

Доведення. Нехай маємо граф G, для якого оптимальне упорядкування S^* довжини l^* при $h = \tilde{h}$ є щільним.

Для будь-якого місця $i \in \overline{1, l^*}$ з того, що вершини з $S^*[i]$ стоять на одному місці, за означенням паралельного упорядкування випливає, що для будь-яких двох вершин $v_1 \in S^*[i], v_2 \in S^*[i]$ не існує орієнтованого шляху ані з v_1 у v_2, ані з v_2 у v_1.

Відомо, що у графі досяжності $\overline{G}$, що відповідає графу G, дві вершини v_1 та v_2 з'єднані ребром лише, коли не існує орієнтованого шляху ані з v_1 у v_2, ані з v_2 у v_1. З наведеного вище випливає, що між усіма парами вершин v_1, v_2 з $S^*[i]$ є ребра у $\overline{G}$, а отже вони утворюють у ньому кліку розмірності $\tilde{h}$.

Оскільки проведені міркування вірні для всіх місць в упорядкуванні S^*, на кожному з них знаходиться по $\tilde{h}$ вершин графу G, і упорядкування S^* містить всі вершини графу G, то з цього випливає висновок твердження. ■

Наведене твердження є необхідною умовою існування щільного упорядкування для графу. Зазначимо, що у випадку $h = 2$ вона є і достатньою, тому що у випадку, коли максимальне паросполучення, що є покриттям вершин кліками розмірності 2, покриває всі вершини графу, отримуємо за відповідним алгоритмом щільне упорядкування. Для $h > 2$ це не так. Приклад такого графу та відповідного графу при $h = 3$ досяжності можна побачити на рис. 2.1 та рис. 2.2.

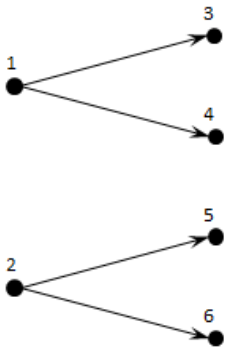

Рис. 2.1. Зображення графу з прикладу недостатності умови з твердження 4

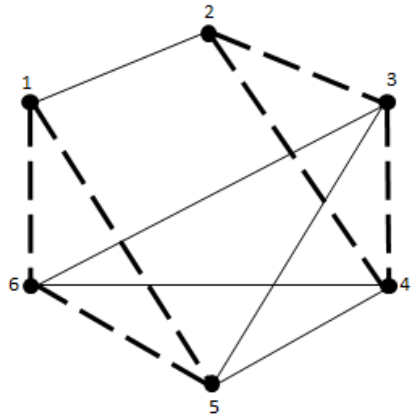

Рис. 2.2. Зображення графу досяжності з прикладу недостатності умови з твердження 4

На рис. 2.2 штриховою лінією позначені ребра графу досяжності, що входять до покриття вершин графу кліками розмірності 3, а суцільною – ребра, що не увійшли до клік. Бачимо, що покриті усі вершини, проте оптимальним покриттям є наступне:

$$S^* = \begin{Bmatrix} 1 & 3 \\ 2, & 4, 6 \\ & 5 \end{Bmatrix}, l(S^*) = 3.$$

Щільне упорядкування не може бути отримане через те, що у графі спочатку лише дві вершини без вхідних дуг.

Алгоритм, заснований на максимальному паросполученні, розглядає пари вершин, тому зручніше застосовувати його до парних значень $\tilde{h}$. Наступне твердження доводить доцільність його використання у цьому випадку.

Твердження 5. Серед максимальних паросполучень M графу досяжності $\overline{G}$, що відповідає графу G, знайдуться ті, що дають оптимальне упорядкування при застосуванні алгоритму, заснованого на максимальному паросполученні, у випадку, коли оптимальне упорядкування при парному значенні ширини $h = \tilde{h}$ є щільним.

Доведення. Нехай маємо граф G, для якого оптимальне упорядкування S^* довжини l^* при $h = \tilde{h}$ є щільним.

Побудуємо для нього граф досяжності $\overline{G}$. З *твердження 4* випливає, що існує покриття вершин графу $\overline{G}$ кліками розмірності $\tilde{h}$. Розглянемо довільну кліку з цього покриття та розіб'ємо вершини, що входять до неї, на пари (можемо це зробити, оскільки $\tilde{h}$ – парне).

Розглянемо тепер всі такі пари, отримані з усіх клік, що утворюють покриття. Ці пари утворюють максимальне паросполучення для графу $\overline{G}$, оскільки кожна вершина належить лише одній парі (пари містять вершини з однієї кліки, а кліки не перетинаються), у кожній парі вершини зв'язані (належать одній кліці), пари покривають всі вершини (пари покривають всі вершини клік, а кліки – всі вершини графу).

Якщо тепер застосуємо алгоритм, заснований на максимальному паросполученні, в якому будемо використовувати на кожному місці максимально

можливу кількість пар, де в якості максимального паросполучення візьмемо отриманий набір пар, то зможемо отримати оптимальне упорядкування, якщо розмістимо пари з однієї кліки на одному місці (вершини з однієї кліки відповідають одному місцю в S^*). ■

Помітимо, що для графу досяжності можуть існувати й максимальні паросполучення, при використанні яких не отримаємо оптимальне упорядкування. Розглянемо граф та відповідний граф досяжності при $h = 4$, зображені на рис. 2.3 та рис. 2.4.

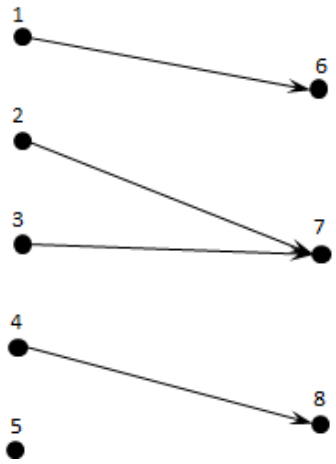

Рис. 2.3. Зображення графу з прикладу існування паросполучень, що не дають оптимальний розв'язок

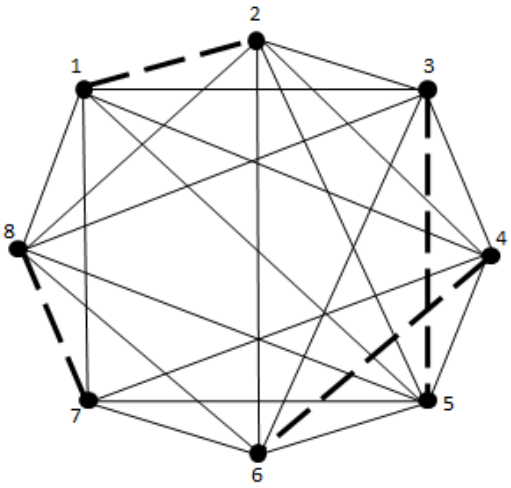

Рис. 2.4. Зображення графу досяжності з прикладу існування паросполучень, що не дають оптимальний розв'язок

На рис. 2.4 штриховою лінією позначені ребра графу досяжності, що входять до максимального паросполучення, а суцільною – ребра, що не увійшли до нього. Оптимальним упорядкуванням для цього графу є наступне:

$$S^* = \begin{Bmatrix} 1 & 5 \\ 2 & 6 \\ 3, & 7 \\ 4 & 8 \end{Bmatrix}, l(S^*) = 2.$$

Через те, що пари $(1,2)$ та $(3,5)$ складаються з вершин, що не мають вхідних дуг, та інших таких пар немає, то, відповідно до алгоритму, поставимо їх на перше місце, проте тоді не зможемо отримати щільне упорядкування, оскільки не можемо

поставити разом пари $(4,6)$ та $(7,8)$ через те, що існує шлях з вершини 4 у 8. У результаті матимемо наступне упорядкування:

$$S = \begin{Bmatrix} 1 \\ 2, 4, 7 \\ 3, 6, 8 \\ 5 \end{Bmatrix}, l(S) = 3.$$

Наведені *твердження 4 та 5* дозволяють стверджувати, що алгоритм, заснований на максимальному паросполученні, в принципі може знаходити оптимальні розв'язки у задачах, де ширина упорядкування є парною і для графів існують щільні упорядкування. З цього випливають доречність використання даного алгоритму до таких задач та необхідність розробки його модифікацій, що будуть відсікати максимальні паросполучення, які не дають оптимальні розв'язки. Дослідження потребує також можливість розробки деяких його узагальнень, які будуть спиратися на кліки більшої розмірності.

Бачимо також, що у класичному варіанті, алгоритм, заснований на максимальному паросполученні, також є лише наближеним для цього класу задач упорядкування.

Експериментальному дослідженню властивостей алгоритму, заснованого на максимальному паросполученні, та його модифікаціям присвячений пункт 2.2.2.

2.2.2 Експериментальні результати

Для експериментального визначення точності алгоритму, заснованого на максимальному паросполученні (АМП), він був реалізований відповідно до його алгоритму, наведеному у цій роботі. Для узагальнення алгоритму на випадок ширини $h > 2$ крок 5 алгоритму повторюється, доки не вичерпаються вільні позиції, чи доки можемо обрати наступну пару або одиночну вершину.

При проведенні експерименту генерувалися лише графи, для яких існують щільні упорядкування при парних значеннях h, оскільки для них знаємо точний розв'язок за побудовою. Нумерація вершин у графі є випадковою. Більш детальний огляд експериментів наведений у [4].

Умови експерименту підсумовані у таблиці 2.1.

Таблиця 2.1. Значення параметрів, використаних при тестуванні

n	h	Кількість тестів
[10,20]	{4,6,8,10}	10000
[21,40]	{4,6,8,10}	10000
[41,60]	{4,6,8,10}	10000
[60,100]	{4,6,8,10}	10000

Результати перевірки точності класичного алгоритму, заснованого на максимальному паросполученні, наведені у таблиці 2.2. Ці та подальші результати містять три параметри: кількість випадків, коли алгоритм, заснований на максимальному паросполученні, давав точний результат; середній відхил довжини отриманого упорядкування від довжини оптимального та кількість випадків, в яких довжина отриманого упорядкування вдвічі довша, аніж довжина оптимального.

Таблиця 2.2. Результати експерименту для класичного алгоритму

Класичний алгоритм		
АМП точний	Середній відхил АМП	Отримано вдвічі довше упорядкування
8688	1.160823	9
5718	1.562121	19
2564	2.165008	4
761	3.391384	0

Результати показали, що він має дуже малу точність. Так кількість випадків, коли він дає оптимальний розв’язок швидко спадає. Середній відхил отриманих розв’язків також дуже швидко зростає і значно відрізняється від одиниці вже для графів невеликої розмірності (20-40 вершин). Окрім цього, існують випадки, коли отримане упорядкування вдвічі довше, аніж оптимальне, що перевищує максимальну оцінку точності з *твердження 1*.

Можна зробити висновок, що зі збільшенням кількості вершин у графі кількість максимальних паросполучень, при використанні яких отримуємо неоптимальні розв’язки, збільшується швидше, аніж кількість тих, за яких отримуємо оптимальні.

Упорядкування вдвічі довші за оптимальні отримуємо, оскільки алгоритм може залишати порожні позиції тому, що вільні вершини зв’язані у пари, які не можуть

бути розміщені. Така поведінка алгоритму ще більше посилює вплив початкового паросполучення. Очевидним способом зменшити цей вплив є дозвіл розділяти пару, якщо вона не може бути розміщена на поточному кроці повністю, проте принаймні одна з її вершин є відкритою. Отриманий алгоритм назвемо модифікованим.

Оскільки на отримане паросполучення сильно впливає нумерація вершин у графі, можна також припустити, що якщо вдасться так їх перенумерувати, щоб з більшою ймовірністю отримувати паросполучення, що дають оптимальні упорядкування, то зможемо додатково покращити результат. Відомо, що для переважної більшості оптимальних розв'язків для графів, для яких існують щільні упорядкування, виконується рівневий принцип. Тому природно припускати, що якщо перенумеруємо вершини графу відповідно до рівневого принципу, то можемо збільшити ймовірність отримати паросполучення, що дає точний розв'язок.

Результати експерименту зі застосування модифікованого алгоритму до попередньо перенумерованих, відповідно до рівневого принципу, вершин наведені у таблиці 2.3.

Таблиця 2.3. Результати експерименту для модифікованого алгоритму з перенумерацією вершин графу

Модифікований алгоритм з перенумерацією		
АМП точний	Середній відхил АМП	Отримано вдвічі довше упорядкування
9675	1.006154	0
8913	1.022079	0
8134	1.107181	0
7491	1.229175	0

Бачимо, що отримали значне покращення показників. Кількість випадків, коли алгоритм був точний для графів із 60-100 вершинами зріс майже у 10 разів, порівняно із класичним. Середній відхил для перших трьох випадків розмірності графів менший, аніж середній відхил для першого випадку для класичного алгоритму, а для четвертого – порівняний із ним.

Повернемося до ідеї, що нумерація вершин у графі сильно впливає на результат роботи алгоритму. Можна припустити, що, якщо випадковим чином перенумерувати

вершини, то отримаємо нове паросполучення та відповідно нове результуюче упорядкування, яке може виявитись коротшим за початкове. Чим більше разів повторимо цю процедуру, тим більша ймовірність, що серед отриманих упорядкувань буде оптимальне.

Для перевірки цієї гіпотези був проведений експеримент, в якому вершини графу випадковим чином перенумеровувались, якщо модифікований алгоритм давав не точний розв’язок, та до отриманого графу повторно застосовувався алгоритм. Описані дії повторювались або доки алгоритм не знайде оптимальне упорядкування, чи доки не вичерпається максимально дозволена кількість повторів. Такий алгоритм називатимемо випадковим модифікованим алгоритмом, заснованим на максимальному паросполученні.

При генерації графів застосовували ті ж значення параметрів; максимальна кількість повторів у всіх випадках обрана рівною 10. Отримані результати наведені у таблиці 2.4.

Таблиця 2.4. Результати експерименту для випадкового модифікованого алгоритму

Випадковий модифікований алгоритм		
АМП точний	Середній відхил АМП	Отримано вдвічі довше упорядкування
9950	1	0
9834	1	0
9600	1.005	0
9145	1.037427	0

Кількість випадків, коли отриманий алгоритм був точним, перевищує 91% для всіх розмірностей графів. Для графів з кількістю вершин від 60 до 100 отримали у 12 разів кращий показник порівняно із класичним алгоритмом. Середній відхил також значно зменшився. Це підтверджує ефективність запропонованого підходу, особливо враховуючи, що максимальна кількість перенумерацій була обрана малою.

Можна припустити, що якщо будемо додатково застосовувати до графів, окрім алгоритмів, заснованих на рівневому принципі, й такий модифікований алгоритм, то зможемо ще покращити загальний результат, в першу чергу за рахунок випадків, в яких для отримання точного упорядкування потрібно порушити рівневий принцип.

Для перевірки цього припущення проведений експеримент, що порівнює точність модифікованого алгоритму, заснованого на максимальному паросполученні, із кращим з результатів алгоритмів, заснованих на лексикографічному та подвійному помічені (пріоритетність вершини визначається спочатку за номером рівня, потім – за кількістю нащадків), далі об'єднаний алгоритм (ОА). Для його проведення були використані ті ж параметри, що й для попереднього експерименту, за винятком того, що при генерації з графів видалялися транзитивні дуги.

Результати складаються з двох частин: порівняння алгоритмів між собою та порівняння точності алгоритмів за значеннями цільової функції. Порівняння алгоритмів між собою містить кількість тестових випадків, в яких за алгоритмом, заснованим на максимальному паросполученні (АМП), отримане упорядкування з меншою довжиною ($l_{\text{АМП}} < l_{\text{ОА}}$); в яких об'єднаним алгоритмом отримане коротше упорядкування ($l_{\text{АМП}} > l_{\text{ОА}}$) та в яких за обома алгоритмами отримані упорядкування однакової довжини ($l_{\text{АМП}} = l_{\text{ОА}}$). Порівняння точності алгоритмів містить кількість випадків, коли за алгоритмом, заснованим на максимальному паросполученні, отриманий точний розв'язок; коли за об'єднаним алгоритмом отриманий точний розв'язок; кількість випадків, коли один з алгоритмів був точним, а також середній відхил довжини розв'язків, отриманих за алгоритмами, від довжини оптимального розв'язку.

Результати наведені у таблицях 2.5-2.6.

Таблиця 2.5. Результати порівняння випадкового модифікованого алгоритму та об'єднаного алгоритму між собою

Випадковий модифікований алгоритм та об'єднаний алгоритм		
$l_{\text{АМП}} < l_{\text{ОА}}$	$l_{\text{АМП}} = l_{\text{ОА}}$	$l_{\text{АМП}} > l_{\text{ОА}}$
4	9957	39
59	9795	146
132	9518	350
195	9085	720

Таблиця 2.6. Результати порівняння точності для випадкового модифікованого алгоритму та об’єднаного алгоритму за значеннями цільової функції

Випадковий модифікований алгоритм та об’єднаний алгоритм				
АМП точний	ОА точний	Один з алгоритмів точний	Середній відхил АМП	Середній відхил ОА
9960	9995	9999	1	1
9841	9928	9987	1	1
9599	9816	9948	1.014963	1
9160	9674	9867	1.041667	1.009202

З результатів бачимо, що кількість випадків, коли випадковий модифікований алгоритм та об’єднаний дають упорядкування однієї довжини, становить більше 90%. Кількість випадків, коли отримані за алгоритмом упорядкування мають меншу довжину, швидко зростає із зростанням кількості вершин у графі. Бачимо також, що в усіх випадках, коли за запропонованим алгоритмом отримували коротші упорядкування, вони виявлялися оптимальними. З іншого боку, кількість випадків, коли об’єднаний алгоритм давав кращі упорядкування, проте вони не були точними, зростає зі збільшенням кількості вершин. Це пов’язано з необхідністю порушення рівневого принципу.

Кількість випадків, в яких принаймні один з алгоритмів є точним, для графів усіх розмірностей перевищує 98,6%, що значно покращує результати при використанні лише алгоритмів, заснованих на рівневому принципі. З цього можна зробити висновок, що отриманий алгоритм не тільки ефективно витримує рівневий принцип, а й порушує його, коли це необхідно.

Усі попередні результати додатково підтверджують ефективність випадкових алгоритмів, заснованих на максимальному паросполученні, а також важливість та перспективність модифікацій класичного алгоритму, як таких, що можуть порушувати рівневий принцип. До недоліків випадкового алгоритму порівняно із алгоритмом з перенумерацією, відповідно до рівневого принципу, можна віднести його не детермінованість, яка значно ускладнює аналітичне дослідження властивостей алгоритму.

2.3 Аналіз впливу автоморфізму на схеми напрямленого перебору

Відмітимо, що при використанні підходу до скорочення перебору, який спирається виключно на уточнення оцінок довжини [5], неминуче зіштовхнемося із випадком, коли повторно аналізуються ізоморфні графи. Розглянемо цей випадок детальніше на прикладі.

Приклад 2.1. Нехай маємо граф з рис. 2.5. Усі його вершини знаходяться на критичних шляхах. Застосуємо метод гілок та меж при $h = 3$, використовуючи базову оцінку знизу довжини упорядкування з підпункту 1.4.2.1.

Отримане для цієї оцінки дерево розгалужень зображено на рис. 2.6 (значення оцінки для проміжного упорядкування вказано в дужках поряд з його номером).

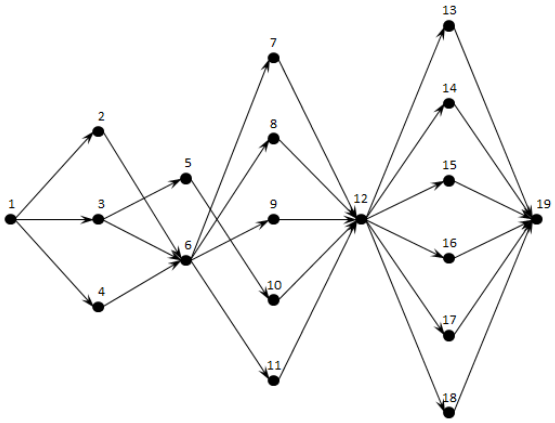

Рис. 2.5. Граф до прикладу 2.1

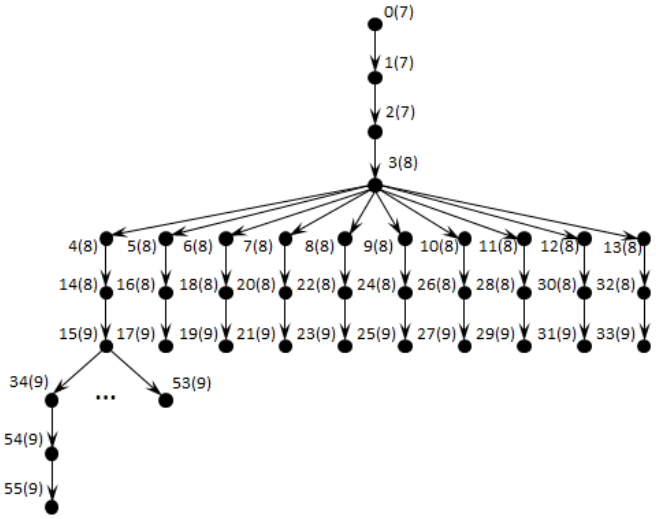

Рис. 2.6. Дерево варіантів методу гілок та меж

Наведемо проміжні упорядкування:

$$S_1 = \{1, \dots\}; S_2 = \begin{Bmatrix} 2 \\ 1, 3, \dots \\ 4 \end{Bmatrix}; S_3 = \begin{Bmatrix} 2 & 5 \\ 1, 3, & 6, \dots \\ 4 & \end{Bmatrix}; S_4 = \begin{Bmatrix} 2 & 5 & 7 \\ 1, 3, & 6, & 8 \dots \\ 4 & & 9 \end{Bmatrix}; S_5 = \begin{Bmatrix} 2 & 5 & 7 \\ 1, 3, & 6, & 8 \dots \\ 4 & & 10 \end{Bmatrix};$$

$$S_6 = \begin{Bmatrix} 2 & 5 & 7 \\ 1, 3, & 6, & 8 \dots \\ 4 & & 11 \end{Bmatrix}; S_7 = \begin{Bmatrix} 2 & 5 & 7 \\ 1, 3, & 6, & 9 \dots \\ 4 & & 10 \end{Bmatrix}; S_8 = \begin{Bmatrix} 2 & 5 & 7 \\ 1, 3, & 6, & 9 \dots \\ 4 & & 11 \end{Bmatrix}; S_9 = \begin{Bmatrix} 2 & 5 & 7 \\ 1, 3, & 6, & 10 \dots \\ 4 & & 11 \end{Bmatrix};$$

$$S_{10} = \begin{Bmatrix} 2 & 5 & 8 \\ 1, 3, & 6, & 9 \dots \\ 4 & & 10 \end{Bmatrix}; S_{11} = \begin{Bmatrix} 2 & 5 & 8 \\ 1, 3, & 6, & 9 \dots \\ 4 & & 11 \end{Bmatrix}; S_{12} = \begin{Bmatrix} 2 & 5 & 8 \\ 1, 3, & 6, & 10 \dots \\ 4 & & 11 \end{Bmatrix}; S_{13} = \begin{Bmatrix} 2 & 5 & 9 \\ 1, 3, & 6, & 10 \dots \\ 4 & & 11 \end{Bmatrix};$$

$$S_{14} = \begin{Bmatrix} 2 & 5 & 7 & 10 \\ 1, 3, & 6, & 8, & 11, \dots \\ 4 & & 9 & \end{Bmatrix}; S_{15} = \begin{Bmatrix} 2 & 5 & 7 & 10 \\ 1, 3, & 6, & 8, & 11, 12, \dots \\ 4 & & 9 & \end{Bmatrix}; S_{16} = \begin{Bmatrix} 2 & 5 & 7 & 9 \\ 1, 3, & 6, & 8, & 11, \dots \\ 4 & & 10 & \end{Bmatrix};$$

$$S_{17} = \begin{Bmatrix} 2 & 5 & 7 & 9 \\ 1, 3, & 6, & 8, & 11, 12, \dots \\ 4 & & 10 & \end{Bmatrix}; S_{18} = \begin{Bmatrix} 2 & 5 & 7 & 9 \\ 1, 3, & 6, & 8, & 10, \dots \\ 4 & & 11 & \end{Bmatrix}; S_{19} = \begin{Bmatrix} 2 & 5 & 7 & 9 \\ 1, 3, & 6, & 8, & 10, 12, \dots \\ 4 & & 11 & \end{Bmatrix};$$

$$S_{20}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\9\\10\end{matrix},\begin{matrix}8\\11\\\end{matrix},\ldots\right\};S_{21}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\9\\10\end{matrix},\begin{matrix}8\\11\\\end{matrix},12,\ldots\right\};S_{22}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\9\\11\end{matrix},\begin{matrix}8\\10\\\end{matrix},\ldots\right\};$$

$$S_{23}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\9\\11\end{matrix},\begin{matrix}8\\10\\\end{matrix},12,\ldots\right\};S_{24}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\10,\\11\end{matrix},\begin{matrix}8\\9\\\end{matrix},\ldots\right\};S_{25}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\10,\\11\end{matrix},\begin{matrix}8\\9\\\end{matrix},12,\ldots\right\};$$

$$S_{26}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}8\\9\\10\end{matrix},\begin{matrix}7\\11\\\end{matrix},\ldots\right\};S_{27}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}8\\9\\10\end{matrix},\begin{matrix}7\\11\\\end{matrix},12,\ldots\right\};S_{28}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}8\\9\\11\end{matrix},\begin{matrix}7\\10\\\end{matrix},\ldots\right\};$$

$$S_{29}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}8\\9\\11\end{matrix},\begin{matrix}7\\10\\\end{matrix},12,\ldots\right\};S_{30}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}8\\10,\\11\end{matrix},\begin{matrix}7\\9\\\end{matrix},\ldots\right\};S_{31}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}8\\10,\\11\end{matrix},\begin{matrix}7\\9\\\end{matrix},12,\ldots\right\};$$

$$S_{32}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}9\\10,\\11\end{matrix},\begin{matrix}7\\8\\\end{matrix},\ldots\right\};S_{33}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}9\\10,\\11\end{matrix},\begin{matrix}7\\8\\\end{matrix},12,\ldots\right\};S_{34}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\8,\\9\end{matrix}\begin{matrix}10\\11\\\end{matrix}\begin{matrix}13\\12,14,\\15\end{matrix},\ldots\right\};$$

$$S_{35}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\8,\\9\end{matrix}\begin{matrix}10\\11\\\end{matrix}\begin{matrix}13\\12,14,\\16\end{matrix},\ldots\right\};S_{36}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\8,\\9\end{matrix}\begin{matrix}10\\11\\\end{matrix}\begin{matrix}13\\12,14,\\17\end{matrix},\ldots\right\};S_{37}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\8,\\9\end{matrix}\begin{matrix}10\\11\\\end{matrix}\begin{matrix}13\\12,14,\\18\end{matrix},\ldots\right\};$$

$$S_{38}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\8,\\9\end{matrix}\begin{matrix}10\\11\\\end{matrix}\begin{matrix}13\\12,15,\\16\end{matrix},\ldots\right\};S_{39}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\8,\\9\end{matrix}\begin{matrix}10\\11\\\end{matrix}\begin{matrix}13\\12,15,\\17\end{matrix},\ldots\right\};S_{40}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\8,\\9\end{matrix}\begin{matrix}10\\11\\\end{matrix}\begin{matrix}13\\12,15,\\18\end{matrix},\ldots\right\};$$

$$S_{41}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\8,\\9\end{matrix}\begin{matrix}10\\11,\\\end{matrix}\begin{matrix}13\\12,16,\\17\end{matrix},\ldots\right\};S_{42}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\8,\\9\end{matrix}\begin{matrix}10\\11,\\\end{matrix}\begin{matrix}13\\12,16,\\18\end{matrix},\ldots\right\};S_{43}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\8,\\9\end{matrix}\begin{matrix}10\\11,\\\end{matrix}\begin{matrix}13\\12,17,\\18\end{matrix},\ldots\right\};$$

$$S_{44}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\8,\\9\end{matrix}\begin{matrix}10\\11,\\\end{matrix}\begin{matrix}14\\12,15,\\16\end{matrix},\ldots\right\};S_{45}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\8,\\9\end{matrix}\begin{matrix}10\\11,\\\end{matrix}\begin{matrix}14\\12,15,\\17\end{matrix},\ldots\right\};S_{46}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\8,\\9\end{matrix}\begin{matrix}10\\11,\\\end{matrix}\begin{matrix}14\\12,15,\\18\end{matrix},\ldots\right\};$$

$$S_{47}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\8,\\9\end{matrix}\begin{matrix}10\\11,\\\end{matrix}\begin{matrix}14\\12,16,\\17\end{matrix},\ldots\right\};S_{48}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\8,\\9\end{matrix}\begin{matrix}10\\11,\\\end{matrix}\begin{matrix}14\\12,16,\\18\end{matrix},\ldots\right\};S_{49}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\8,\\9\end{matrix}\begin{matrix}10\\11,\\\end{matrix}\begin{matrix}14\\12,17,\\18\end{matrix},\ldots\right\};$$

$$S_{50}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\8,\\9\end{matrix}\begin{matrix}10\\11,\\\end{matrix}\begin{matrix}15\\12,16,\\17\end{matrix},\ldots\right\};S_{51}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\8,\\9\end{matrix}\begin{matrix}10\\11,\\\end{matrix}\begin{matrix}15\\12,16,\\18\end{matrix},\ldots\right\};S_{52}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\8,\\9\end{matrix}\begin{matrix}10\\11,\\\end{matrix}\begin{matrix}15\\12,17,\\18\end{matrix},\ldots\right\};$$

$$S_{53}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\8,\\9\end{matrix}\begin{matrix}10\\11,\\\end{matrix}\begin{matrix}16\\12,17,\\18\end{matrix},\ldots\right\};S_{54}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\8,\\9\end{matrix}\begin{matrix}10\\11,\\\end{matrix}\begin{matrix}13&16\\12,14,&17,\\15&18\end{matrix},\ldots\right\};S_{55}=\left\{\begin{matrix}2\\1,3,\\4\end{matrix}\begin{matrix}5\\6,\\\end{matrix}\begin{matrix}7\\8,\\9\end{matrix}\begin{matrix}10\\11,\\\end{matrix}\begin{matrix}13&16\\12,14,&17,\\15&18\end{matrix},19\right\}.$$

Отже, при використанні цієї оцінки в методі гілок та меж знадобилося 55 розгалужень, причому розгалуження 1-15 є необхідними, тобто вони завжди будуть наявні у дереві незалежно від оцінки, оскільки отримані у випадках, коли усі вершини одного фіксованого рівня у дереві мали однакові оцінки. Аналогічно, розгалуження 34-55 також будуть у кожному дереві. Усі інші розгалуження пов’язані з вибором вершини дерева варіантів, що розглядається.

Розглянемо детальніше розгалуження 4-13. Легко побачити, що всім їм відповідають ізоморфні підграфи, тоді для кожного з них довжини оптимальних упорядкувань є рівними, а отже всі гілки, що йдуть з вершин 4-13 будуть однаковими, і має сенс розглядати лише одну з них. Аналогічна ситуація відбувається з

розгалуженнями 34-53. Отже, при додатковій перевірці графів на ізоморфність, вдалося б знайти оптимальне упорядкування за 10 розгалужень.

Варто відмітити також, що розгалужень 16-33 вдалося б також уникнути при застосуванні більш точнішої оцінки, тобто знадобилося б 37 розгалужень замість 55. Навіть маючи точну оцінку, яка завжди дорівнює оптимальній довжині, цю кількість не вдалося б зменшити. А оскільки задача належить до класу NP-важких, то очікується, що таку оцінку побудувати не можна, а отже для будь-якої оцінки у загальному випадку у дереві розгалужень будуть виникати розгалуження, які відповідають ізоморфним графам, та для яких оцінка буде давати однакові значення. Так, якби оцінка рівна 9 вперше була отримана у розгалуженні 54, то у дереві було б додаткових 379 розгалужень. Важливо відмітити, що кількість можливих розгалужень швидко зростає з кількістю відкритих вершин.

З цього випливає важливість дослідження гілок дерев варіантів на ізоморфність відповідних підграфів, що дасть змогу не лише значно зменшити кількість вершин, що підлягають аналізу, а отже значно його пришвидшити, але й також скоротити витрати ресурсів пам’яті.

Перед тим як перейти до розгляду вказаної проблеми, наведемо відомі поняття «ізоморфізму», «автоморфізму» та «інваріанту» орієнтованих графів [40].

Означення 2.3. Два орієнтовані графи G та H є ізоморфними, якщо між множинами їх вершин існує взаємно однозначна відповідність, що зберігає суміжність вершин та орієнтацію дуг. Позначається $G \cong H$ або $G = H$.

Означення 2.4. Автоморфізмом орієнтованого графу G називається ізоморфізм G на себе, або автоморфізм є підстановкою множини вершин V, що зберігає суміжність та орієнтацію дуг. Дві вершини i та j, що переходять одна в одну під дією автоморфізму $f: V \to V$ $(i \in V, j \in V,\ f(i) = j)$ будемо називати взаємозамінними та позначати $i \sim j$.

Означення 2.5. Інваріантом графу G називається деяка характеристика графу, найчастіше числова, пов’язана із графом G, яка зберігається для будь-якого графу, що є ізоморфним G. Прикладами інваріантів є кількість вершин чи дуг у графі.

Відзначимо, що наразі невідомо чи належать задачі перевірки графів на ізоморфність та визначення взаємозамінних вершин до класу P або NP-повних, тому вони віднесені до класу GI. Варто зазначити також, що для практично важливих класів графів існують алгоритми, що швидко визначають ізоморфізм та знаходять взаємозамінні вершини для графів великої розмірності [99-102].

Наведена проблема є тематично близькою до задачі генерації деякого класу графів без ізоморфізму [103-107], що тісно пов'язана з визначенням взаємозамінних вершин. Далі наведені твердження, що можуть скоротити кількість розгалужень, завдяки врахуванню взаємозамінності відкритих вершин.

Відмітимо також, що можливий підхід, який заснований на перевірці ізоморфізму за допомогою інваріантів. Справді, відомо, що для деяких інваріантів ймовірність, що два випадкові графи будуть мати однакові значення цих інваріантів та не будуть при цьому ізоморфними, майже нульова [108]. Проте у випадку, що розглядається, при розгалуженні підграфи утворюються з графу шляхом видалення деяких відкритих вершин, тому більша частина структури підграфів буде залишатися сталою, і можна очікувати, що й більшість інваріантів для підграфів будуть співпадати.

У подальшому вважається, що якимось чином вдалося визначити, які вершини у графі є взаємозамінними. Перевага у роботі віддається тому, як можна скористатися цією інформацією, а не як її отримати.

Введемо допоміжне означення.

Означення 2.6. Два набори вершин графу α та β називатимемо еквівалентними, якщо існує бієктивне відображення $f: \alpha \to \beta$ таке, що проектує вершини α у відповідні їм взаємозамінні вершини з β, тобто $\forall i \in \alpha: f(i) \in \beta \wedge i \sim f(i)$.

При розгляді цієї проблеми природно виникають два питання:

- чи можна при видаленні *еквівалентних* наборів отримати не *ізоморфні* підграфи;
- чи можна при видаленні *неекві валентних* наборів отримати *ізоморфні підграфи*.

Відповіді на обидва питання є позитивними, проілюструємо це на прикладах.

Приклад 2.2. Розглянемо граф G з рис. 2.7. Взаємозамінними відкритими вершинами у ньому є вершини 1 та 4, а також вершини 2 та 3. Нехай $h = 2$, отже маємо обрати 2 вершини. З вершин 1-4 можна утворити 4 попарно еквівалентних набори: $(1,2), (1,3), (2,4), (3,4)$. Граф G_1 утворений з графу G видаленням набору $(3,4)$, а граф G_2 – набору $(2,4)$.

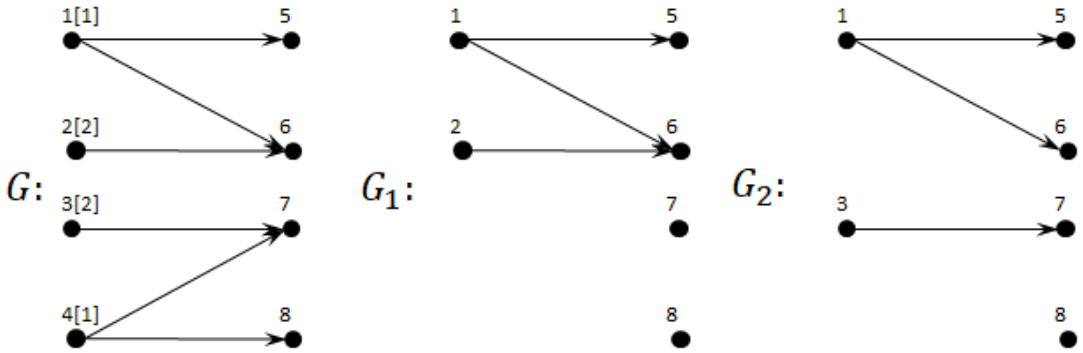

Рис. 2.7. Граф та його підграфи з прикладу 2.2

Легко побачити, що графи G_1 та G_2 не ізоморфні: вони мають різну кількість ізольованих вершин. Також помітимо, що граф утворений видаленням набору $(1,2)$ буде ізоморфним графу G_1, а видаленням набору $(1,3)$ – графу G_2.

З іншого боку, цікавим є факт, що після видалення вершини 4, що наявна в обох наборах, вершини 2 та 3 перестають бути взаємозамінними, тобто отримали неізоморфні графи через те, що видалили не взаємозамінні вершини. В той же час у випадку наборів $(1,2)$ та $(3,4)$ після видалення, наприклад, вершин 1 та 4, вершини 2 та 3 залишаються взаємозамінними та навпаки.

Приклад 2.3. Розглянемо граф G з рис. 2.8. Взаємозамінними відкритими вершинами у ньому є вершини 1 та 2. Нехай $h = 2$, маємо обрати для видалення 2 вершини. Розглянемо два не еквівалентних набори $(1,2)$ та $(2,3)$. Граф G_1 утворений з графу G видаленням набору $(2,3)$, а граф G_2 – видаленням набору $(1,2)$.

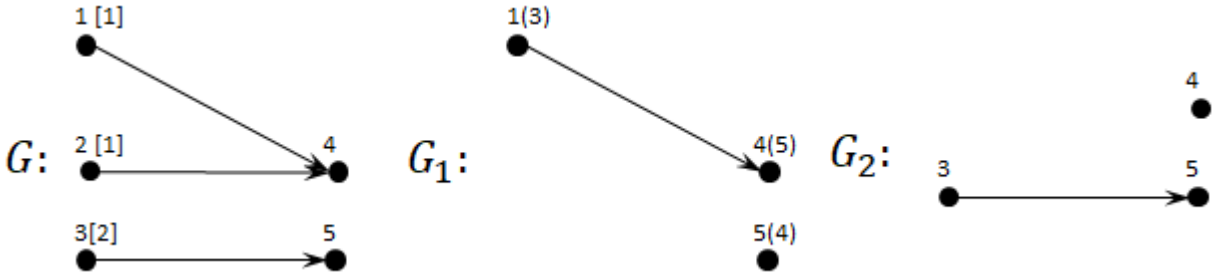

Рис. 2.8. Граф та його підграфи з прикладу 2.3

Графи G_1 та G_2 є ізоморфними (ізоморфізм вказаний на зображенні графу G_1 у дужках). Також будуть ізоморфними графи, утворені видаленням еквівалентних наборів $(1,3)$ та $(2,3)$.

Відмітимо також, що аналогічно до прикладу 2.2, після видалення вершини 2, що є спільною для наборів, вершини 1 та 3 стають взаємозамінними, тобто отримали ізоморфні графи через те, що видаляємо лише взаємозамінні вершини.

З розглянутих прикладів бачимо, що визначення взаємозамінних вершин лише у початковому графі та побудови еквівалентних наборів вершин недостатньо для визначення ізоморфності підграфів, які утворюються при розгалуженні. З іншого боку, видно, що важливим є визначення взаємозамінних вершин після видалення кожної вершини з графу. На основі цього можна запропонувати наступний алгоритм генерації неізоморфних підграфів.

Алгоритм 1. Обрання наборів вершин для розгалуження

Вхідні дані: граф G, ширина упорядкування h.
Вихідні дані: множина наборів вершин N.

1: $procedure\ generate(G, h_{max}, V_{out}, P, N)$
2: $\quad if\ length(P) + length(V_{out}) < h_{max}\ then$
3: $\quad\quad exit;$
4: $\quad end\ if$
5:
6: $\quad S \leftarrow$ {попарно не взаємозамінні вершини у G з множини V_{out}};
7: $\quad V'_{out} \leftarrow V_{out};$
8: $\quad for\ v\ in\ S$
9: $\quad\quad if\ length(P) + 1 = h_{max}\ then$
10: $\quad\quad\quad N \leftarrow N \cup (P \cup v);$
11: $\quad\quad else$
12: $\quad\quad\quad generate(G \backslash v, h_{max}, V'_{out}, P \cup v, N);$
13: $\quad\quad\quad V'_{out} \leftarrow V'_{out} \setminus$ {вершини з V_{out} взаємозаємні з v};
14: $\quad\quad end\ if$
15: $\quad end\ for$

16: $end\ procedure$
17:
18: $N \leftarrow \emptyset; P \leftarrow \emptyset; V_{out} \leftarrow$ {відкриті вершини з G};
19: $generate(G, h_{max} = \min\{h, length(V_{out})\}, V_{out}, P, N)$;

Для обґрунтування алгоритму потрібно довести два факти: що множина N містить набори, які дають всі можливі неізоморфні підграфи, і множина N не містить наборів, що дають ізоморфні підграфи.

Перший факт напряму випливає з наступної теореми.

Теорема 1. Якщо у графі G вершини i та j є взаємозамінними, то графи G_1 та G_2, утворені видаленням вершин i та j з G відповідно, будуть ізоморфними.

Доведення. Розглянемо автоморфізм f, який переводить вершину i у вершину j. Побудуємо дві матриці суміжності: в першій рядки і стовпці відповідають вершинам у порядку $1,2,\dots,n$ (n – кількість вершин у графі), в другій – у порядку $f(1), f(2), \dots, f(n)$.

За означенням автоморфізму ці матриці суміжності будуть співпадати. Більш того, оскільки $f(i) = j$, i-ті рядок і стовпець у другій матриці відповідають вершині j, а отже матриці, утворені з початкових видаленням цих рядків і стовпчиків, також співпадають. З іншого боку, отримані матриці є матрицями суміжності для графів G_1 та G_2, з чого випливає, що вони ізоморфні за означенням. ■

Справді, з теореми випливає, що отримати неізоморфні графи, видаляючи взаємозамінні вершини, неможливо. А отже, коли у відповідності до алгоритму беремо лише одну вершину з кожної множини попарно взаємозамінних вершин, то не втрачаємо жодного з наборів відкритих вершин, що дають різні неізоморфні графи.

Перед дослідженням другого факту для обґрунтування алгоритму відмітимо, що важливою є умова, що видаляються саме відкриті вершини.

Приклад 2.4. Розглянемо граф G з рис. 2.9. Графи G_1, G_2 та G_3 утворені з графу G шляхом видалення з нього вершин 3, 2 та 1 відповідно.

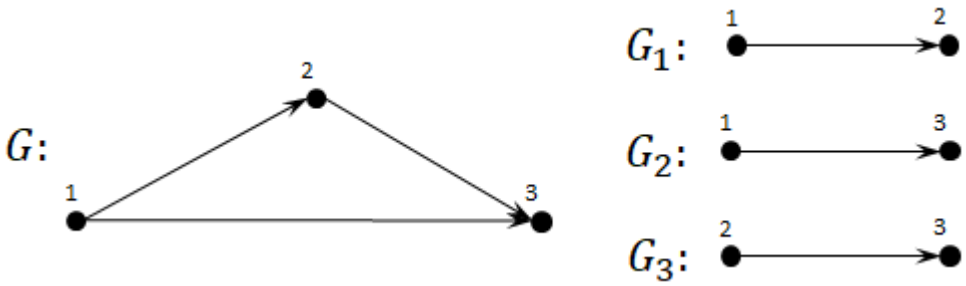

Рис. 2.9. Граф та його підграфи з прикладу 2.4

Бачимо, що жодні дві вершини у графі не є взаємозамінними, проте у всіх випадках отримуємо ізоморфні графи.

Для того, щоб довести, що жодні два набори з N не можуть привести до ізоморфних графів, необхідно перевірити наступне твердження.

Твердження 6. Для будь-яких двох наборів відкритих вершин α та β графу G з того, що графи $G_1 = G\backslash\alpha$ та $G_2 = G\backslash\beta$ є ізоморфними, випливає існування таких послідовностей вершин $\alpha' = (\alpha_1, \dots, \alpha_m), \alpha_i \in \alpha, \alpha_i \neq \alpha_j$ та $\beta' = (\beta_1, \dots, \beta_m), \beta_i \in \beta, \beta_i \neq \beta_j$, що α_1 та β_1 є взаємозамінними та для всіх $k = \overline{1,(m-1)}$ для графів $G\backslash(\alpha_1, \dots, \alpha_k)$ та $G\backslash(\beta_1, \dots, \beta_k)$ існує ізоморфізм, що переводить вершину α_{k+1} у вершину β_{k+1}.

Це твердження є хибним, що випливає з наступного прикладу.

Приклад 2.5. Розглянемо граф G з рис. 2.10. Нехай $h = 2$, маємо обрати для видалення 2 вершини. Граф G_1 утворений з графу G видаленням вершин 2 та 3, граф G_2 – вершин 1 та 4.

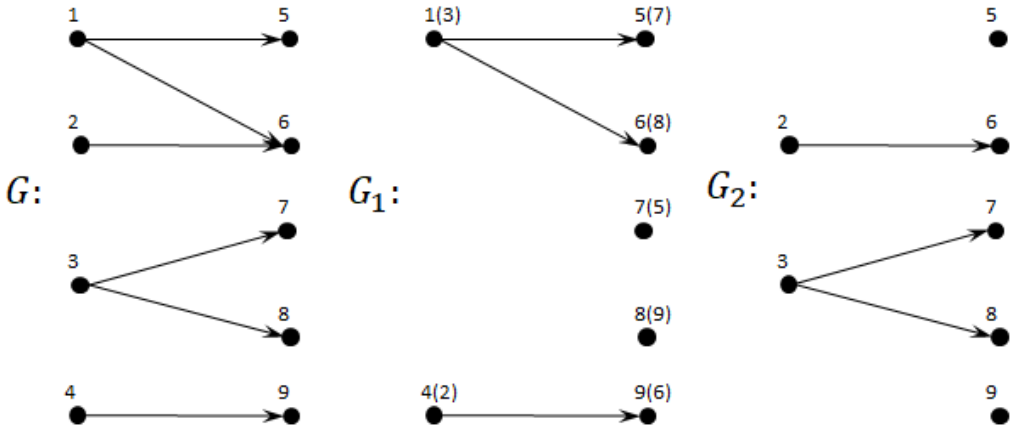

Рис. 2.10. Граф та його підграфи з прикладу 2.5.

Графи G_1 та G_2 є ізоморфними (відповідність між нумерацією вершин вказана на зображенні графу G_1 у дужках), проте серед вершин 1-4 немає взаємозамінних, а отже множина N може містити два набори, видалення яких приводять до ізоморфних графів. Помітимо проте, що при одночасному видаленні з графу вершин 1 та 3, вершини 2 та 4 стають взаємозамінними та навпаки.

Також варто перевірити чи є умова з *теореми 1* не лише необхідною, але й достатньою, тобто слабку версію попереднього *твердження 6*.

Твердження 7. Якщо графи G_1 та G_2, утворені видаленням деяких відкритих вершин i та j з G відповідно, є ізоморфними, то вершини i та j є взаємозамінними у графі G.

Це твердження також не є вірним. Проілюструємо це наступним прикладом.

Приклад 2.6. Розглянемо граф G з рис. 2.11. Граф G_1 утворений з графу G видаленням вершин 2, граф G_2 – вершин 1.

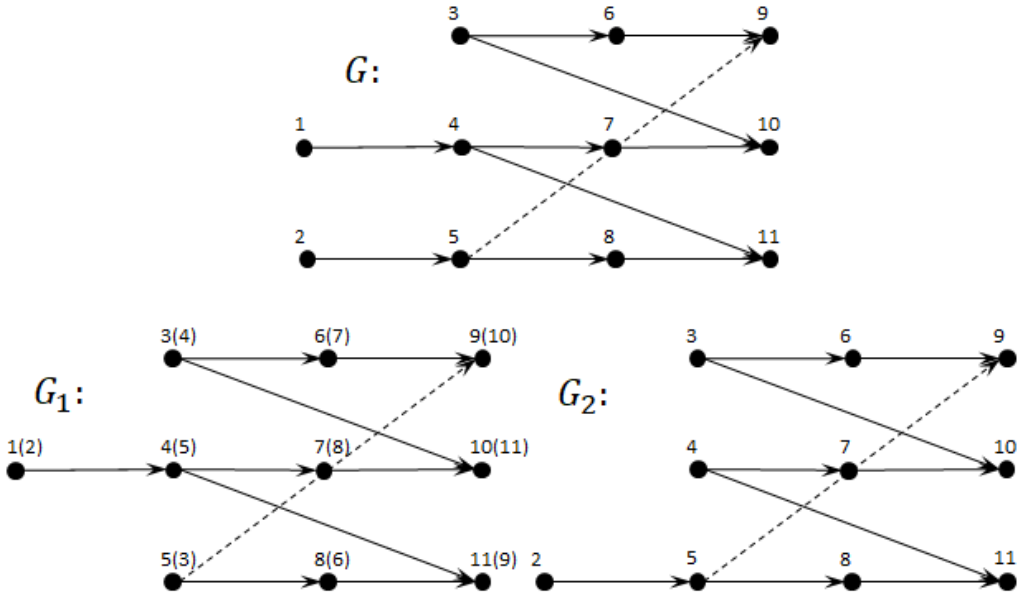

Рис. 2.11. Граф та його підграфи з прикладу 2.6

Графи G_1 та G_2 є ізоморфними (ізоморфізм вказаний на зображенні графу G_1 у дужках), проте вершини 1 і 2 не є взаємозамінними у графі G. Це легко побачити, якщо розглянути вершину-стік 11: відстань від вершини 1 до 11 дорівнює двом, а від вершини 2 – трьом; з іншого боку, вершиною-стоком на відстані два від вершини 2 є вершина 9, а вершиною-стоком на відстані три від 1 – вершина 10, а отже при

автоморфізмі вершина 11 мала б відобразитись і у вершину 9, і у вершину 10, що неможливо.

Отже, запропонований алгоритм дозволяє скоротити кількість розгалужень, що підлягають перевірці у методі гілок та меж при його застосуванні до задачі паралельного упорядкування, але не дозволяє повністю позбавитися від ізоморфних підграфів.

2.3.1 Використання інваріантів графів для скорочення напрямленого перебору

У попередньому підрозділі був представлений алгоритм, що дозволяє скоротити перебір у методі гілок та меж шляхом визначення відкритих взаємозамінних вершин у графі. Доведено, що наведений алгоритм не втрачає розгалуження, які відповідають неізоморфним підграфам, проте знаходження таких вершин у загальному випадку є трудомісткою задачею.

Іншим підходом до вирішення цієї проблеми є наближене визначення ізоморфності підграфів за допомогою потужних інваріантів. Одним з таких інваріантів є розмітка графу, отримана за алгоритмом уточнення кольорів [109].

Алгоритм уточнення кольорів

Визначимо послідовність розфарбувань вершин c^t наступним чином:

1. Нехай c^0 – початкове розфарбування. Якщо вершини графу не мають поміток, що несуть додаткову інформацію, то у початковому розфарбуванні покладаємо кольори $c^0(v)$ всіх вершин $v \in V$ однаковими. Інакше колір вершини визначається відповідною поміткою.

2. Колір вершини у наступному розфарбуванні визначається за формулою: $c^{t+1}(v) = \left(c^t(v), \left\{\{c^t(w) | w \text{ є суміжною до } v\}\right\}\right)$. Тобто новий колір вершини визначається впорядкованою парою значень: її кольором у поточному розфарбуванні та мультимножиною кольорів вершин, суміжних із нею.

Алгоритм продовжує уточнювати поточне розфарбування. Після деякої кількості кроків $t < n$ воно стабілізується, тобто кількість кольорів у c^t перестає змінюватися при збільшенні t. Таке розфарбування називається стабільним.

Доведено, що ймовірність того, що два випадкові графи мають однакову розмітку за цим алгоритмом та не є при цьому ізоморфними, є близькою до нуля [108]. У роботі [109] також показано, що правильність визначення ізоморфізму двох графів тісно пов'язана із двогранними графами.

Означення 2.7. Дводольний граф G, в якому будь-які дві вершини, що належать до однієї долі, мають рівні степені (напівстепені), називається двогранним.

Графи з цього класу при однаковій кількості вершин у долях та кількості вхідних та вихідних дуг для відповідних вершин мають однакову розмітку, проте не обов'язково є ізоморфними. Виявилось, що у цьому випадку довжини оптимальних упорядкувань також можуть бути різними.

Так для графів з рис. 2.12 ці довжини при $h = 4$ дорівнюють 4 та 3 відповідно.

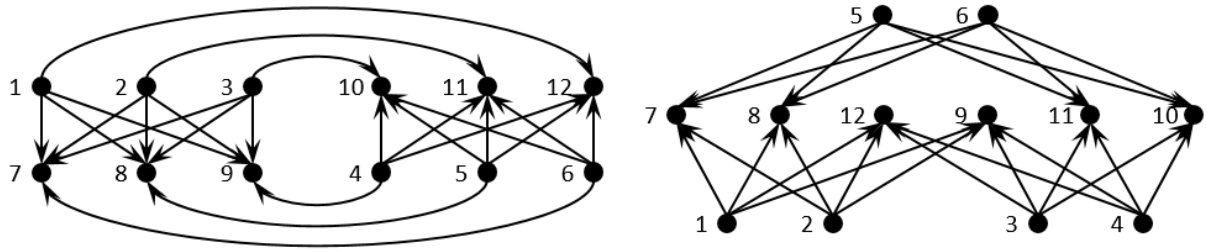

Рис. 2.12. Двогранні графи з різними довжинами оптимальних упорядкувань при h = 4

Графи не є ізоморфними, оскільки жодна пара відкритих вершин у першому графі не має чотири спільні суміжні вершини, на відміну, наприклад, від вершин 1 та 2 другого графу. При цьому обидва графи є дводольними, всі відкриті вершини належать до першої долі та мають напівстепінь виходу рівний чотирьом, всі вершини другої долі мають напівстепінь входу рівний чотирьом, а отже ці графи є двогранними. Відповідні оптимальні упорядкування при $h = 4$:

$$S_1 = \begin{Bmatrix} 1 & 5 & 7 \\ 2 & 6, & 8 \\ 3, & 9 & 10, 12 \\ 4 & & 11 \end{Bmatrix}, S_2 = \begin{Bmatrix} 1 & 5 & 7 \\ 2 & 6 & 8 \\ 3, & 9 & , 10 \\ 4 & 12 & 11 \end{Bmatrix}.$$

Ці графи можна об'єднати шляхом додавання вершин 2, 3, 5 та 6 разом з вихідними дугами з одного графу в інший. Отже, можемо втратити розгалуження, що

відповідають графам з різною довжиною упорядкування, проте при цьому може бути знайдене і шукане упорядкування.

Тому актуальним є визначення оцінки точності для цього підходу.

2.3.2 Автоморфізм у паралельно-послідовних графах

Подальший розгляд цієї теми присвячено пошуку таких класів графів, для яких виконувалися б *твердження 6, 7* з підрозділу 2.3. Одним з перспективних підкласів виявилися паралельно-послідовні графи.

2.3.2.1 Поняття паралельно-послідовних графів

Клас паралельно-послідовних графів визначається рекурсивно наступним чином [110]:

– Орграф $G = (\{i\}, \emptyset)$, який складається з єдиної вершини, належить цьому класу.

– Якщо графи $G_1 = (V_1, U_1)$ та $G_2 = (V_2, U_2)$, де $V_1 \cap V_2 = \emptyset$, належать цьому класу, тоді

$$G = G_1 S G_2 = (V_1 \cup V_2, U_1 \cup U_2 \cup I_1 \times O_2),$$

де I_1 – множина стоків графу G_1, O_2 – множина джерел графу G_2, також буде належати даному класу. В цьому випадку кажуть, що граф G є послідовною композицією графів G_1 та G_2.

– Якщо графи $G_1 = (V_1, U_1)$ та $G_2 = (V_2, U_2)$, де $V_1 \cap V_2 = \emptyset$, належать цьому класу, тоді граф

$$G = G_1 P G_2 = (V_1 \cup V_2, U_1 \cup U_2)$$

також буде належати даному класу, і він є паралельною композицією графів G_1 та G_2.

– Тільки ті орграфи, що можуть бути отримані шляхом скінченої кількості застосувань попередніх двох побудов, належать до класу паралельно-послідовних.

Ключовою властивістю цих графів є те, що якщо множини суміжних вершин до деяких двох вершин перетинаються, то вони співпадають.

Графи цього класу можна природним чином подати у вигляді бінарного дерева, в якому листи відповідають окремим вершинам графу, а внутрішні вузли визначають тип композиції, що була застосована при побудові до графів, які відповідають

піддеревам з коренями у дочірніх вузлах. Таке бінарне дерево називають бінарним деревом декомпозиції. Якщо у бінарному дереві декомпозиції зв'язані групи однакових типів композиції стиснути в одну вершину з відповідним об'єднанням дочірніх вузлів, то отримане дерево називають канонічним деревом декомпозиції.

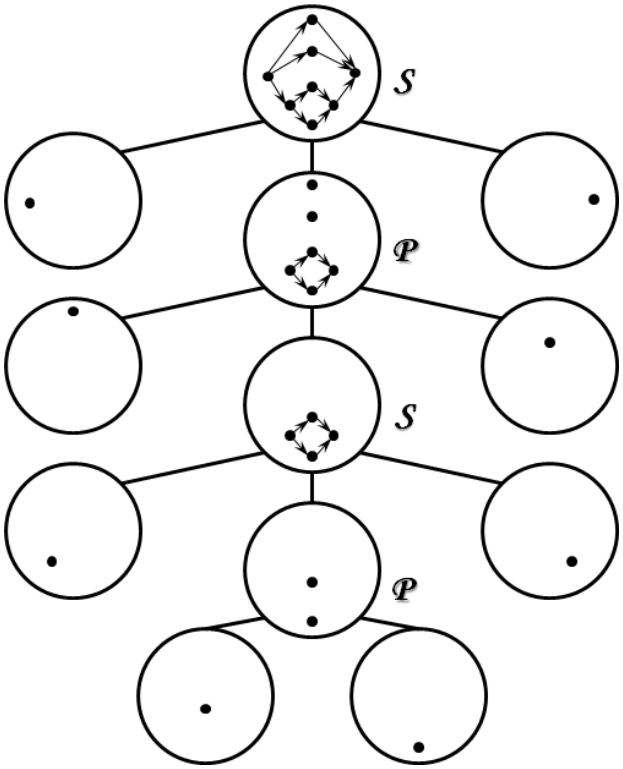

Рис. 2.13. Приклад канонічного дерева декомпозиції

Застосування дещо модифікованого відомого алгоритму перевірки ізоморфізму двох кореневих дерев [111,112], який враховує різні типи вузлів (порядок нащадків при формуванні інваріанту піддерева у послідовних вузлах типу S має значення, а в паралельних вузлах типу P – ні), дозволяє визначати множини взаємозамінних вершин для таких графів за лінійний час.

2.3.2.2 Використання специфіки паралельно-послідовних графів при побудові паралельних упорядкувань

Для паралельно-послідовних графів *твердження 7* виявилися істинним, що доводиться у наступних лемах та *теоремі 2*.

Лема 1. Орієнтовні ациклічні графи G_1 та G_2 ізоморфні тоді і тільки тоді, коли графи G_1SG та G_2SG є ізоморфними, де G – довільний ациклічний орграф.

Доведення. Необхідність. Нехай графи G_1 та G_2 ізоморфні, f – відповідний ізоморфізм, G – деякий граф.

Побудуємо дві матриці суміжності M_1 та M_2: в першій матриці рядки і стовпці відповідають вершинам графу G_1 у порядку $1,2,\dots,n$ (n – кількість вершин у графах G_1 та G_2), в другій – графу G_2 у порядку $f(1), f(2), \dots, f(n)$. За означенням ізоморфізму ці матриці збігаються. Нехай M – матриця суміжності для графу G.

Розглянемо тепер блочні матриці $M_1' = \begin{pmatrix} M_1 & D_1 \\ 0 & M \end{pmatrix}$ та $M_2' = \begin{pmatrix} M_2 & D_2 \\ 0 & M \end{pmatrix}$, де D_1 та D_2 – матриці, в яких стоять одиниці на перетині рядків, що у графах G_1 та G_2 відповідають стокам, та стовпців, що у графі G відповідають джерелам. Матриці D_1 та D_2 збігаються, оскільки їх значення визначається лише наборами рядків і стовпців, а стоки ізоморфізм переводить у стоки, за властивістю, і матриця M є однаковою у M_1' та M_2', а отже і стовпці-джерела графу G. З цього випливає, що матриці співпадають за побудовою, а, з іншого боку, є матрицями суміжності графів G_1SG та G_2SG відповідно, тому вони є ізоморфними за означенням.

Достатність. Нехай графи G_1, G_2 та G – деякі графи, графи G_1SG та G_2SG ізоморфні, f – відповідний ізоморфізм.

Побудуємо матрицю суміжності графу G_1SG у формі $\begin{pmatrix} M_1 & D_1 \\ 0 & M \end{pmatrix}$, де блок M_1 відповідає вершинам графу G_1, блок M – вершинам графу G, D_1 – матриця, в якій стоять одиниці на перетині рядків, що у графі G_1 відповідають стокам, та стовпців, що у графі G відповідають джерелам. Це зможемо зробити, оскільки між вершинами графів G_1 та G дуги є лише між всіма стоками G_1 та джерелами G за означенням послідовної композиції. Тоді, якщо побудуємо матрицю суміжності для графу G_2SG з рядками та стовпцями у порядку $f(1), f(2), \dots, f(m)$ (m – кількість вершин у графі G_1SG), вона буде збігатися з матрицею суміжності для графу G_1SG, за означенням ізоморфізму.

Серед вершин $f(1), f(2), \dots, f(n)$ (n – кількість вершин у графі G_1) не може бути вершин графу G, оскільки ізоморфізм зберігає всі властивості графів та сума відстаней від будь-якої вершини графу G_1 до всіх стоків графу G строго більша за

аналогічну суму для будь-якої вершини графу G, за властивістю послідовної композиції. Шляхом видалення з матриць суміжності рядків і стовпців $n+1, n+2, \dots, m$, що в обох графах відповідають вершинам графу G, отримаємо матриці M_1, що є матрицями суміжності графів G_1 та G_2, а отже вони є ізоморфними за означенням. ■

Наслідок з леми 1. Орієнтовні ациклічні графи G_1 та G_2 ізоморфні тоді і тільки тоді, коли графи GSG_1 та GSG_2 є ізоморфними, де G – довільний ациклічний орграф.

Доведення. Аналогічно доведенню *леми 1*. ■

Лема 2. Орієнтовні ациклічні графи G_1 та G_2 ізоморфні тоді і тільки тоді, коли графи G_1PG та G_2PG є ізоморфними, де G – довільний ациклічний орграф.

Доведення. Необхідність. Нехай графи G_1 та G_2 ізоморфні, f – відповідний ізоморфізм, G – деякий граф.

Побудуємо дві матриці суміжності M_1 та M_2: в першій матриці рядки і стовпці відповідають вершинам графу G_1 у порядку $1,2, \dots, n$ (n – кількість вершин у графах G_1 та G_2), в другій – графу G_2 у порядку $f(1), f(2), \dots, f(n)$. За означенням ізоморфізму ці матриці збігаються. Нехай M – матриця суміжності для графу G.

Розглянемо тепер блочні матриці $M_1' = \begin{pmatrix} M_1 & 0 \\ 0 & M \end{pmatrix}$ та $M_2' = \begin{pmatrix} M_2 & 0 \\ 0 & M \end{pmatrix}$. Вони співпадають за побудовою, а, з іншого боку, є матрицями суміжності графів G_1PG та G_2PG відповідно, з чого випливає, що вони ізоморфні за означенням.

Достатність. Нехай графи G_1, G_2 та G – деякі графи, графи G_1PG та G_2PG ізоморфні, f – відповідний ізоморфізм.

Побудуємо матрицю суміжності графу G_1PG у формі $\begin{pmatrix} M_1 & 0 \\ 0 & M \end{pmatrix}$, де блок M_1 відповідає вершинам графу G_1, блок M – вершинам графу G. Завжди зможемо це зробити, оскільки між вершинами графів G_1 та G немає дуг за означенням паралельної композиції. Тоді, якщо побудуємо матрицю суміжності для графу G_2PG з рядками та стовпцями у порядку $f(1), f(2), \dots, f(m)$ (m – кількість вершин у графі G_1PG), вона буде збігатися з матрицею суміжності для графу G_1PG, за означенням ізоморфізму.

Якщо серед вершин $f(1), f(2), \dots, f(n)$ (n – кількість вершин у графі G_1) немає вершин графу G, тоді шляхом видалення з матриць суміжності тих рядків і стовпців, що в обох графах відповідають вершинам G, отримаємо матриці M_1, які є матрицями суміжності графів G_1 та G_2, тому вони є ізоморфними за означенням.

Якщо серед вершин $f(1), f(2), \dots, f(n)$ знайдуться вершини графу G, тоді переставимо рядки і стовпці, які відповідають вершинам $f(1), f(2), \dots, f(n)$ та $1,2, \dots, n$, у матрицях суміжності так, щоб у матриці суміжності для графу G_2PG спочатку йшли вершини, що відповідають графу G_2, а потім G. Позначимо новий порядок вершин у першій матриці $i_1, i_2, \dots, i_k, i_{k+1}, \dots, i_n$, де k – кількість тих вершин з $f(1), f(2), \dots, f(n)$, що належать графу G_2. Помітимо, що при цьому зміниться лише матриця M_1, оскільки сусідні блоки нульові, позначимо змінену матрицю M_2. Вона також матиме блочно-діагональний вигляд, оскільки між вершинами графів G_2 та G немає дуг за означенням паралельної композиції. Тоді вершини $i_{k+1}, \dots, i_n$ у графі G_1 та вершини $f(i_{k+1}), \dots, f(i_n)$ у графі G мають дуги лише між собою (всі інші значення у відповідних рядках та стовпцях дорівнюють нулю), а відповідні підграфи є ізоморфними (мають однакові матриці суміжності після видалення рядків і стовпців, що відповідають іншим вершинам).

Побудуємо відображення g графу G_1PG на себе, яке для вершини $i_{k+1}, \dots, i_n$ співпадає з f, для вершин $f(i_{k+1}), \dots, f(i_n)$ з f^{-1}, а для інших вершин є тотожнім. Таке відображення буде автоморфізмом графу G_1PG, що випливає з попереднього абзацу. Розглянемо композицію $f_1 = f \circ g$: вона є ізоморфізмом (за властивістю композиції ізоморфізму та автоморфізму), залишає вершини $f(i_{k+1}), \dots, f(i_n)$ на місці, оскільки $f\left(f^{-1}\left(f(i_l)\right)\right) = f(i_l), l = \overline{k+1, n}$, відображає вершини $i_{k+1}, \dots, i_n$ у вершини $f\left(f(i_{k+1})\right), f\left(f(i_{k+2})\right), \dots, f\left(f(i_n)\right)$ та не змінює відображення решти вершин (зокрема $i_1, i_2, \dots, i_k$ відображаються у ті ж вершини графу G_2).

Якщо тепер серед $f_1(1), f_1(2), \dots, f_1(n)$ немає вершин графу G, то твердження доведено з попередніх міркувань, інакше зможемо повторити усі наведені міркування та побудувати новий ізоморфізм. Тоді, оскільки граф є скінченним та кожен

наступний ізоморфізм додає нові вершини, які він залишає на місці, та не впливає на відображення вершин, які залишали на місці попередні ізоморфізми, за побудовою, зрештою всі вершини G_1 будуть проектуватися на вершини G_2. ■

Наслідок 1 з лем 1 та 2. Орієнтовні ациклічні графи G_1 та G_2 ізоморфні тоді і тільки тоді, коли графи, утворені деякою послідовністю паралельних та послідовних комбінацій цих графів з відповідними графами, є ізоморфними.

Доведення. Випливає з послідовного використання попередніх *лем 1, 2* та *наслідку з леми 1*. ■

Наслідок 2 з лем 1 та 2. Дві вершини є взаємозамінними у графі G тоді і тільки тоді, коли вони є взаємозамінними у графі, утвореному деякою послідовністю паралельних та послідовних комбінацій цього графу з деякими графами.

Доведення. Випливає безпосередньо з *наслідку 1 з лем 1 та 2*. ■

Теорема 2. Якщо для деяких відкритих вершин i та j паралельно-послідовного орграфу G, графи $G\backslash i$ та $G\backslash j$ є ізоморфними, то вершини i та j є взаємозамінними у графі G.

Доведення. Нехай T_c – канонічне дерево декомпозиції графу G. Розглянемо шляхи від вершин i та j до кореня T_c. Першою вершиною, що належить обом цим шляхам має бути вершина p, що відповідає паралельній композиції. Справді, якщо б p відповідала послідовній композиції, то з цього випливало б, що існує орієнтований шлях або з вершини i у вершину j, або у зворотному напрямку, за властивістю послідовної композиції, що в свою чергу суперечить тому, що обидві ці вершини є відкритими у G.

Розглянемо тепер граф G', що відповідає піддереву T_c з коренем p. Цей граф є паралельно-послідовним, оскільки є підграфом паралельно-послідовного графу G, та не є зв'язним, оскільки корінь p його дерева декомпозиції відповідає паралельній композиції. Вершини i та j належать графу G' за побудовою. З *наслідку 1 з лем 1 та 2* та ізоморфності графів $G\backslash i$ й $G\backslash j$ випливає, що графи $G'\backslash i$ та $G'\backslash j$ також є ізоморфними.

Нехай тепер $G'_1, G'_2, \dots, G'_k$ - система максимальних (за включенням) неорієнтовано зв'язних підграфів графу G' (набір послідовно-паралельних графів, що утворюють G' за допомогою паралельної композиції). Без втрати загальності можемо вважати, що вершина $i \in G'_1$, а вершина $j \in G'_2$, інакше p не була б першою загальною вершиною на шляхах до кореня T_c.

Розглянемо тепер графи $G'\backslash i$ та $G'\backslash j$. Граф $G'\backslash i$ утворено з підграфів $\mathrm{G}'_1\backslash i, \mathrm{G}'_2, \dots, \mathrm{G}'_k$, а граф $G'\backslash j$ – з підграфів $\mathrm{G}'_1, \mathrm{G}'_2\backslash j, \dots, \mathrm{G}'_k$. З ізоморфізму цих графів випливає, що знайдеться таке бієктивне відображення f, яке підграфу $G'\backslash i$ ставить у відповідність підграф $G'\backslash j$, ізоморфний йому. Відзначимо, що підграфи $\mathrm{G}'_1\backslash i$ та G'_1 і $G'_2\backslash j$ та G'_2 не ізоморфні, оскільки містять різну кількість вершин, підграфи $G'_q, q = \overline{3,k}$ містяться в обох графах.

Покажемо, що на основі f можна побудувати відображення, що проектує $G'_1\backslash i$ на $G'_2\backslash j$ та G'_1 на G'_2. Розглянемо $f(G'_1) = G'_x$, нехай $x \neq 2$ та $f(G'_x) = G'_y$, тоді з транзитивності відношення ізоморфізму випливає, що підграфи G'_1 та G'_y - ізоморфні, а отже зможемо побудувати нове відображення, що проектує G'_1 на G'_y, а G'_x на G'_x (залишає на місці). Повторюючи аналогічну побудову, прийдемо до відображення, що проектує G'_1 на G'_2, оскільки G'_1 завжди буде проектуватися на якийсь новий підграф (попередні будуть проектуватися самі на себе) та G'_2 - єдиний підграф, який міститься лише в одному з графів та може бути ізоморфним з G'_1. Аналогічні міркування можемо застосувати до підграфу $G'_1\backslash i$, а отже шукане відображення існує.

Отже, підграфи G'_1 та G'_2 і $G'_1\backslash i$ та $G'_2\backslash j$ є ізоморфними. Нехай g ізоморфізм з G'_1 у G'_2 і $g(i) = l$. Тоді, якщо l співпадає з вершиною j, вершини i та j є взаємозамінними у G' за означенням (можна утворити наступний автоморфізм: g переводить G'_1 у G'_2 та i у j, g^{-1} – G'_2 у G'_1, тотожне відображення – G'_q у $G'_q, q = \overline{3,k}$), а отже, за *наслідком 2 з лем 1 та 2*, й у графі G, і твердження теореми є доведеним.

Якщо ж $l \neq j$, тоді маємо послідовно-паралельний граф G'_2, в якому є дві відкриті вершини l та j і графи $G'_2\backslash j$ та $G'_2\backslash l$ є ізоморфними, оскільки $g(i) = l$ та графи $G'_1\backslash i$ та $G'_2\backslash j$ є ізоморфними, а отже отримали початкове твердження для графу меншої

розмірності. Тоді можемо всі міркування доведення застосувати до нього знову. Граф є скінченним, тому врешті-решт залишиться граф з однією вершиною, а отже будуть виконані умови з попереднього абзацу. ■

Доведена теорема підтверджує перспективність використання алгоритму генерації неізоморфних підграфів для паралельно-послідовних графів.

Виявилось, що для такого класу графів *твердження 6* також не виконується. Розглянемо приклад графу з рис. 2.14. Графи G_1та G_2 утворені з графу G видаленням вершин 1,2 та 3,5 відповідно. Ці графи також є ізоморфними, відповідність вершин наведена у дужках біля поміток вершин графу G_2. Отже, маємо два графи, що утворені шляхом видалення пари відкритих вершин з вихідного графу, та є ізоморфними. При цьому неважко перевірити, що серед пар вершин (1,3), (1,5), (2,3), (2,5) немає взаємозамінних.

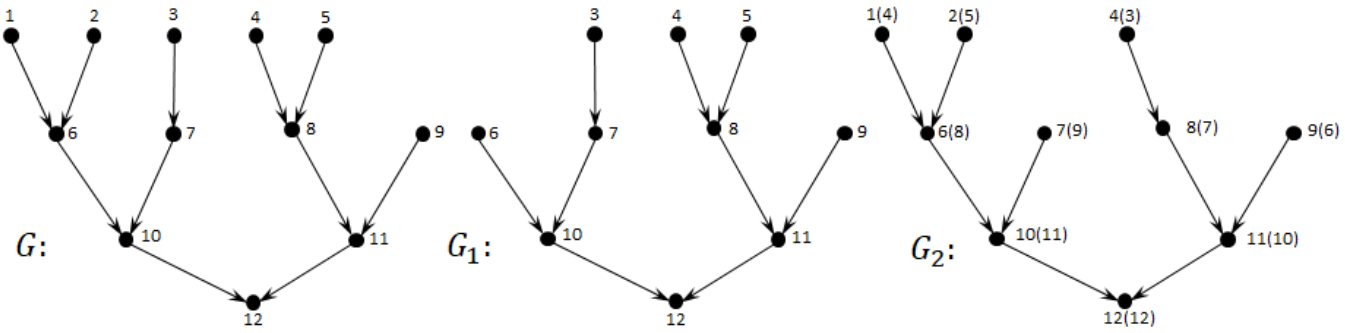

Рис. 2.14. Контрприклад для випадку видалення двох вершин з бінарного дерева

Оскільки графи у наведеному прикладі є кореневими бінарними деревами, які є найпростішими нетривіальними графами, то можна очікувати, що це твердження є хибним для усіх важливих класів графів. Тому у подальших дослідженнях варто зосередитись на кількісному визначенні, який виграш дає такий варіант скорочення перебору.

2.4 Аналіз графів з щільним упорядкуванням

У підрозділі 2.1 показано, що будь-яка класична задача паралельного упорядкування може бути зведена до задачі, в якій шукане упорядкування є щільним.

Очевидною перевагою такої задачі є те, що довжина оптимального упорядкування відома і потрібно лише його побудувати.

Щільність упорядкування накладає ряд додаткових обмежень на проміжні упорядкування, що будуються при використанні методу гілок та меж, тому їх врахування може дозволити значно скоротити кількість гілок, що підлягають розгляду.

Відзначимо спершу, що через те, що довжина упорядкування відома, можемо визначити місця, які в упорядкуванні може займати кожна з вершин. Розглянемо узагальнену схему визначення крайнього лівого та правого місця, які в упорядкуванні може займати вершина.

Для визначення цих місць в задачі $P_S(G, l, h)$ використовуються спеціальні упорядкування $\underline{S}$ та $\overline{S}$. Для кожної вершини v_i визначаються місця $\underline{\mu_i}$ та $\overline{\mu_i}$, які вона займає в цих упорядкуваннях відповідно. Тоді проміжок допустимих місць, що може займати ця вершина, визначається наступним чином: $\left[\underline{\mu_i}; \overline{\mu_i} + l - \underline{l}\right]$, де $\underline{l}$ – довжина критичного шляху у графі G.

Відмітимо, що цей алгоритм неможливо застосувати до задачі $P_S(G, h, l)$ в загальному випадку, оскільки довжина упорядкування є невідомою величиною. Помітимо також, що в наведених міркуваннях не враховується інформація про допустиму ширину упорядкування h, бо для $P_S(G, l, h)$ вона є шуканою величиною. Проте для задачі $P_S(G, h, l)$ з щільним упорядкуванням обидві величини є відомими, тому зможемо більш точно визначити межі можливого розташування вершин.

Значення $\underline{\mu_i}$ та $\underline{l} - \overline{\mu_i} + 1$ можна розглядати як елементарні оцінки знизу довжини упорядкування для підграфів $\underline{G_i}$ та $\bar{G}_i$, можливо порожніх, вершини яких мають шляхи до вершини v_i та з цієї вершини відповідно. Дійсно, $\underline{\mu_i}$ та $\underline{l} - \overline{\mu_i} + 1$ на одиницю більші за довжини критичних шляхів у цих графах (необхідність додавати одиницю пов'язана з відсутністю вершини v_i у графах). Враховуючи, що всі вершини графів $\underline{G_i}$ та $\bar{G}_i$ мають бути розташованими в упорядкуванні ліворуч та праворуч від v_i відповідно, можемо замість довжини критичного шляху використати будь-яку

іншу відому оцінку для упорядкування, оскільки це є теоретично мінімально можливою кількістю місць, на яких можуть бути розташовані всі вершини графу. Таким чином отримаємо деякі уточнені значення $\underline{\mu_i^h}$ та $\overline{\mu_i^h}$, які можемо використовувати аналогічним чином.

Відмітимо також, що такі ж міркування можна застосовувати й при обчисленні оцінок, які використовують спеціальні упорядкування $\underline{S}$ та $\overline{S}$. Таким чином зможемо побудувати упорядкування $\underline{S^h}$ та $\overline{S^h}$, в яких кожна вершина v_i графу займає крайнє ліве та крайнє праве місце у відповідності з деякою оцінкою довжини для графів $\underline{G_i}$ та $\bar{G}_i$. Ті оцінки, що використовують упорядкування $\underline{S^h}$ та $\overline{S^h}$ замість $\underline{S}$ та $\overline{S}$ будуть уточненими оцінками довжини знизу.

Зауважимо, що упорядкування $\underline{S^h}$ та $\overline{S^h}$ мають дві ключові відмінності від упорядкувань $\underline{S}$ та $\overline{S}$: по-перше, вони можуть мати порожні місця, по-друге, їх довжини можуть бути різними. Так, наприклад, для графу $K_{1,4}$, $h = 3$ та базової оцінки довжини упорядкування $\overline{S^h}$ має порожнє друге місце, його довжина дорівнює 3, в той же час довжина упорядкування $\underline{S^h}$ дорівнює 2. З цієї причини при визначенні крайнього правого допустимого місця в упорядкуванні замість довжини критичного шляху у графі потрібно використовувати довжину упорядкування $\overline{S^h}$.

Усі наведені міркування підсумовано у наступному алгоритмі.

Алгоритм визначення поміток $\underline{\mu_i^h}$

1. Вершини без вхідних дуг отримують помітки $\underline{\mu_i^h} = 1$.

2. Розглянемо множину вершин $\underline{V}$, які ще не мають поміток, а усі їх попередники вже помічені.

3. Для кожної вершини $v_i \in \underline{V}$ побудуємо спеціальне упорядкування $\underline{S_i^h}$, в якому попередники v_j вершини v_i будуть розташовані відповідно на місцях $\underline{\mu_j^h}$.

4. Вершина v_i отримує помітку $\underline{\mu_i^h} = 1 + f\left(\underline{S_i^h}\right)$, де $f(\cdot)$ – деяка відома оцінка знизу довжини упорядкування; або, більш загально, $\underline{\mu_i^h} = 1 + f\left(\underline{G_i}\right)$.

5. Якщо у графі залишилися вершини без поміток повертаємося на 2, інакше кінець.

Аналогічні помітки $\overline{\mu_i^h}$ можна отримати, якщо рухатись від вершин, що не мають вихідних дуг.

Перейдемо тепер до розгляду обмежень, що накладає вимога щільності на проміжні упорядкування. Тож нехай на деякому етапі побудови дерева розгалужень вже заповненими в упорядкуванні є перші k місць та нерозташованими залишаються вершини підграфу $\tilde{G}$. Будемо розглядати обмеження в порядку складності їх перевірки.

Очевидно, якщо у графі $\tilde{G}$ менше ніж h відкритих вершин, то побудувати щільне упорядкування вже не вдасться, тому поточну гілку можна відсікти.

Помітимо, що в процесі розташування вершин підграфи $\bar{G}_i$ залишаються незмінними, а отже значення поміток $\overline{\mu_i^h}$ можна порахувати один раз для початкового графу G, враховуючи у подальшому, що їх значення потрібно розглядати відносно вихідного графу, а не графу $\tilde{G}\left(\tilde{V}, \tilde{U}\right)$ (вони будуть відрізнятися на значення різниці між довжинами упорядкувань $\overline{S^h}$ для цих двох графів). Для зручності подальшого викладення будемо використовувати зведені помітки крайнього правого місця, яке вершина може займати в упорядкуванні: $\overline{\lambda_i^h} = \overline{\mu_i^h} + l - \overline{l^h}$, де $\overline{l^h}$ – довжина упорядкування $\overline{S^h}$ для початкового графу G.

Розглянемо тепер множини вершин $\overline{V_j^h} = \left\{v_i \in \tilde{V}: \overline{\lambda_i^h} \le j\right\}, j = \overline{k+1, l}$, тобто тих вершин, які мають бути розташовані у шуканому упорядкуванні до місця j включно. З іншого боку, кількість вершин, що можна розташувати на місця з $k+1$ по j дорівнює $h * (j-k)$. Зрозуміло, що у випадку, коли потужність відповідної множини перевищує цю кількість, тобто $\left|\overline{V_j^h}\right| > h * (j-k)$, побудувати щільне

упорядкування також вже не вдасться і подальші розгалуження можна не проводити. Відзначимо, що додатково можна було б перевіряти, що всі множини $\overline{V_j^h}, j = \overline{1,k}$ є порожніми, проте достатньо перевірити лише множину $\overline{V_k^h}$, за побудовою. Ця перевірка вписується у загальну схему, оскільки маємо $h * (k - k) = 0$. Маємо для цього випадку $\overline{\lambda_i^h} \leq k \Longrightarrow l - k \leq \overline{l^h} - \overline{\mu_i^h} < \overline{l^h} - \overline{\mu_i^h} + 1$, тобто довжина критичного шляху у $\tilde{G}$ буде перевищувати кількість місць, доступних для розташування.

Зауважимо, що тут і далі у цьому підрозділі, довжина критичного шляху розуміється у дещо узагальненому сенсі як мінімальна кількість місць, необхідна для розташування всіх вершин графу відповідно до деякого критерію.

Для перевірки наступних обмежень потрібно обчислити помітки $\underline{\mu_i^h}$ для вершин графу $\tilde{G}$, які на відміну від $\overline{\mu_i^h}$ постійно змінюються внаслідок розташування вершин. Також будемо використовувати зведені помітки крайнього лівого допустимого місця: $\underline{\lambda_i^h} = \underline{\mu_i^h} + k$.

Порівняємо тепер для кожної вершини помітки $\underline{\lambda_i^h}$ та $\overline{\lambda_i^h}$. Якщо знайдеться така вершина v_i, для якої $\underline{\lambda_i^h} > \overline{\lambda_i^h}$, то щільне упорядкування побудувати також не вдасться, оскільки для розміщення всіх нащадків та попередників цієї вершини знадобиться більше ніж $l - k$ місць: $\underline{\lambda_i^h} > \overline{\lambda_i^h} \Longrightarrow \underline{\mu_i^h} + \left(\overline{l^h} - \overline{\mu_i^h}\right) > l - k$. Перейдемо тепер до вершин, для яких ці помітки співпадають: $V_j^h = \left\{v_i \in \tilde{V}: \underline{\lambda_i^h} = \overline{\lambda_i^h} = j\right\}, j = \overline{k+1, l}$. Всі вершини множин V_j^h можуть бути розташовані лише на місці j, тоді, якщо хоча б одна з них містить більше ніж h елементів, то також не отримаємо щільне упорядкування. Отже, отримали необхідну умову: $\left|V_j^h\right| \leq h$.

Залишилося розглянути вершини, для яких $\underline{\lambda_i^h} < \overline{\lambda_i^h}$. Побудуємо множини $V_{j_1,j_2}^h = \left\{v_i \in \tilde{V}: j_1 \leq \underline{\lambda_i^h}, \overline{\lambda_i^h} \leq j_2\right\}, k+1 \leq j_1 \leq j_2 \leq l$, всі вершини цих множин можуть бути розміщенні лише на місцях з відрізку $[j_1; j_2]$. З іншого боку, цей діапазон вміщує $h * (j_2 - j_1 + 1)$ вершин, а отже у випадку, коли потужність якоїсь з цих

множин перевищує наведене значення, також не вдасться побудувати щільне упорядкування, і має виконуватись: $\left|V_{j_1,j_2}^h\right| \le h * (j_2 - j_1 + 1)$.

Підсумовуючи розглянуті випадки бачимо, що неможливо отримати щільне упорядкування через те, що для розміщення вершин графу $\tilde{G}$ необхідно більше ніж $(l-k)$ місць. А отже, якби вдалося побудувати оцінку знизу довжини, яка б враховувала всі зазначені характеристики у структурі графу, то не потрібно було б перевіряти кожну умову окремо.

Відмітимо, що з виконання останньої необхідної умови існування щільного упорядкування випливає виконання майже всіх попередніх простіших умов. По-перше, легко побачити, що $V_j^h = V_{j,j}^h, j = \overline{k+1,l}$ та $h * (j - j + 1) = h$, а отже умова для вершин з рівними зведеними помітками виконується. По-друге, якщо $\left|\overline{V_k^h}\right| = 0$, тоді $\overline{V_j^h} = V_{k+1,j}^h, j = \overline{k+1,l}$ і $h * (j - (k+1) + 1) = h * (j - k)$, тобто виконується умова для вершин, права зведена помітка яких не перевищує j. По-третє, розглянемо множину $V_{k+2,l}^h$: вона не містить відкритих вершин, оскільки всі відкриті вершини мають помітки $\underline{\lambda_i^h} = k+1$. Тоді, якщо їх кількість менша за h, то матимемо $\left|V_{k+2,l}^h\right| > h * (l-k) - h = h * (l - (k+2) + 1)$, а отже відповідна умова буде порушена. Останні дві умови, пов'язані з $\overline{V_k^h}$ та вершинами, для яких $\underline{\lambda_i^h} > \overline{\lambda_i^h}$, можуть бути замінені перевіркою довжини критичного шляху у $\tilde{G}$, як показано вище.

Відзначимо також, що майже в усіх попередніх необхідних умовах перевірялося обмеження потужності місць, а не можливість їх заповнення. Так, наприклад, можна розглядати множини вершин, які можуть бути розташовані на заданому проміжку: $\dot{V}_{j_1,j_2}^h = \left\{v_i \in \tilde{V}: [j_1; j_2] \cap \left[\underline{\lambda_i^h}; \overline{\lambda_i^h}\right] \ne \emptyset\right\}, k+1 \le j_1 \le j_2 \le l$. Очевидно, що для щільного заповнення проміжку місць необхідно, щоб потужності всіх цих множин перевищували $h * (j_2 - j_1 + 1)$. Розглянемо тепер множини V_{k+1,j_1-1}^h та $V_{j_2+1,l}^h$. За побудовою, ці множини, якщо вони непорожні, містять вершини, які можна розташувати виключно на проміжках $[k+1; j_1 - 1]$ та $[j_2 + 1; l]$ відповідно, а отже

вони не містять спільних елементів. Більш того, перетин $\dot{V}^h_{j_1,j_2}$ з цими множинами також є порожнім, оскільки проміжки не перетинаються, і також можна побачити, що кожна вершина належить принаймні одній з цих множин. Тоді у випадку недостатньої кількості вершин для розміщення $\left|\dot{V}^h_{j_1,j_2}\right| < h * (j_2 - j_1 + 1)$ отримаємо, що $\left|V^h_{k+1,j_1-1} \cup V^h_{j_2+1,l}\right| = \left|V^h_{k+1,j_1-1}\right| + \left|V^h_{j_2+1,l}\right| > h * (l-k) - h * (j_2 - j_1 + 1) = h * (l - k - j_2 + j_1 - 1) = h * (l - (j_2 + 1) + 1) + h * \big((j_1 - 1) - (k+1) + 1\big)$, а тоді, за принципом Діріхле, принаймні одна з необхідних умов для множин V^h_{k+1,j_1-1} та $V^h_{j_2+1,l}$ буде порушена. А отже, цю необхідну умову також можна не перевіряти окремо.

Насправді, обидві зазначені необхідні умови можна об’єднати в одну. Для цього проведемо спочатку перетворення над умовою для діапазонів місць, аби звести її до більш зручного вигляду. Розділимо обидві частини формули на h та скористаємося тим, що для $\forall a, c \in \mathbb{Z}, b \in \mathbb{N}$ виконується $a \le b * c \Leftrightarrow \frac{a}{b} \le c \Leftrightarrow \left\lceil \frac{a}{b} \right\rceil \le \lceil c \rceil = c$, оскільки $\lceil \cdot \rceil$ є неспадною функцією, тоді отримаємо $\left\lceil \frac{\left|V^h_{j_1,j_2}\right|}{h} \right\rceil \le j_2 - j_1 + 1$ для $k+1 \le j_1 \le j_2 \le l$. Перенесемо тепер ліворуч праву частину нерівності та додамо l до обох частин, тоді для тих саме значень j_1, j_2 матимемо $(j_1 - 1) + \left\lceil \frac{\left|V^h_{j_1,j_2}\right|}{h} \right\rceil + (l - j_2) \le l$. Оскільки остання нерівність має виконуватись для всіх значень $k+1 \le j_1 \le j_2 \le l$, то вона є еквівалентною наступній:

$$\max_{k+1 \le j_1 \le j_2 \le l} \left\{ (j_1 - 1) + \left\lceil \frac{\left|V^h_{j_1,j_2}\right|}{h} \right\rceil + (l - j_2) \right\} \le l.$$

Розглянемо детальніше доданки, що утворюють цей вираз. Другий доданок відповідає мінімальній кількості місць, необхідних для розташування вершин, що можуть розташовуватись виключно на місцях з діапазону $[j_1; j_2]$. Перший та третій доданок – мінімальній довжині критичних шляхів зліва та справа для цих вершин відповідно.

Розглянемо тепер випадок, коли довжина критичного шляху перевищує $(l - k)$. У цьому випадку знайдуться такі значення $j_1 > j_2$, для яких множина $V^h_{j_1,j_2}$ не буде

порожньою. Тоді для цих вершин маємо $\left\lceil \frac{\left|V_{j_1,j_2}^h\right|}{h} \right\rceil \geq 1$, і тоді $(j_1 - 1) + \left\lceil \frac{\left|V_{j_1,j_2}^h\right|}{h} \right\rceil + (l - j_2) \geq l + (j_1 - j_2) > l$, що вписується у загальну перевірку. А отже зможемо об'єднати цю умову з попередньою, замінивши допустиму область на множину пар (j_1, j_2), для яких множина V_{j_1,j_2}^h не є порожньою. Це не зменшує загальності, оскільки у випадку $j_1 \leq j_2$ та $V_{j_1,j_2}^h = \emptyset$ матимемо $(j_1 - 1) + \left\lceil \frac{\left|V_{j_1,j_2}^h\right|}{h} \right\rceil + (l - j_2) = l - 1 - (j_2 - j_1) < l$, тобто умова буде виконана.

Відмітимо окремо, що для двох пар значень (j_1, j_2) та (j_1', j_2') таких, що $j_1' \leq j_1, j_2 \leq j_2', V_{j_1,j_2}^h = V_{j_1',j_2'}^h$, маємо $(j_1' - 1) + \left\lceil \frac{\left|V_{j_1',j_2'}^h\right|}{h} \right\rceil + (l - j_2') = (j_1 - 1) + \left\lceil \frac{\left|V_{j_1,j_2}^h\right|}{h} \right\rceil + (l - j_2) - (j_1 - j_1') - (j_2' - j_2) \leq (j_1 - 1) + \left\lceil \frac{\left|V_{j_1,j_2}^h\right|}{h} \right\rceil + (l - j_2)$, а отже будь-яке розширення діапазонів не призводить до порушення умови, і має сенс розглядати лише найкоротші з них.

З іншого боку, у випадку $j_1 = k + 1, j_2 = l$ множина V_{j_1,j_2}^h міститиме всі вершини, за побудовою позначок $\underline{\lambda_i^h}$ та $\overline{\lambda_i^h}$, тобто $\left|V_{j_1,j_2}^h\right| = h * (l - k)$, оскільки шукане упорядкування є щільним. Тоді отримаємо $(j_1 - 1) + \left\lceil \frac{\left|V_{j_1,j_2}^h\right|}{h} \right\rceil + (l - j_2) = (k + 1 - 1) + \left\lceil \frac{h*(l-k)}{h} \right\rceil + (l - l) = l$, а отже шуканий максимум обмежень зверху та знизу значенням довжини упорядкування, і необхідна умова існування щільного упорядкування приймає остаточний вигляд:

$$l = \max_{(a,b):V_{a,b}^h \neq \emptyset} \left\{ (a - 1) + \left\lceil \frac{\left|V_{a,b}^h\right|}{h} \right\rceil + (l - b) \right\}.$$

Розглянемо тепер як зсуви значень $\underline{\lambda_i^h}, \overline{\lambda_i^h}$ на константи $\underline{\Delta}$ та $\overline{\Delta}$ відповідно впливають на отриманий вираз. Зафіксуємо деяку довільну пару значень (a, b): $V_{a,b}^h \neq \emptyset$. Розглянемо тепер пару $\left(a + \underline{\Delta}, b + \overline{\Delta}\right)$, зрозуміло, що в рамках зсунутих позначень

множина вершин $V^h_{a+\underline{\Delta},b+\overline{\Delta}}$ співпаде з множиною $V^h_{a,b}$ у вихідних позначеннях, а отже другий доданок виразу не зміниться. Для третього доданку маємо $\left(l+\overline{\Delta}\right)-\left(b+\overline{\Delta}\right)=l-b$, а отже зсув також не впливає і на його значення. Для першого доданку – $\left(a+\underline{\Delta}-1\right)=(a-1)+\underline{\Delta}$, тобто підсумкове значення правої частини зміниться на $\underline{\Delta}$.

Відзначені спостереження дозволяють перейти від розгляду поміток $\underline{\lambda_i^h},\overline{\lambda_i^h}$ до $\underline{\mu_i^h},\overline{\mu_i^h}$. При цьому допустима множина прийме вигляд $X=\left\{(a,b)\colon 1\le a\le \underline{l^h},1\le b\le \overline{l^h},V^h_{a,b}\neq\emptyset\right\}$, де $\underline{l^h}$, $\overline{l^h}$ – довжини упорядкувань $\underline{S^h}$ та $\overline{S^h}$ для графу $\tilde{G}$, тут і далі $V^h_{a,b}$ будуються, використовуючи значення $\underline{\mu_i^h},\overline{\mu_i^h}$ замість $\underline{\lambda_i^h},\overline{\lambda_i^h}$ без зміни позначень. Тоді умова прийме вигляд:

$$l=k+\max_{(a,b)\in X}\left\{(a-1)+\left\lceil\frac{\left|V^h_{a,b}\right|}{h}\right\rceil+\left(\overline{l^h}-b\right)\right\}.$$

Така форма є більш звичною для методу гілок та меж, оскільки тепер всі елементи, які приймають участь в пошуку максимуму пов’язані лише з проміжним графом $\tilde{G}$. Введена допустима множина також є більш зручною для опрацювання випадку $\underline{l^h}\neq\overline{l^h}$.

У загальному випадку, отриманий максимум буде оцінкою знизу довжини упорядкування для проміжного графу $\tilde{G}$, що випливає з наведеного вище аналізу доданків, з яких він складається. Можна також переконатися, що він є узагальненням вже відомих оцінок з підпункту 1.4.2.1 (якщо використаємо помітки $\underline{\mu_i}$ та $\overline{\mu_i}$, при $a=1$ отримаємо уточнену оцінку, а при $b=\underline{l}$ – останній аргумент максимуму покращеної оцінки). Це дозволяє очікувати, що використання такої оцінки знизу також дозволить скоротити кількість розгалужень у методі гілок та меж у загальному випадку.

Перейдемо тепер до питання, як можна ефективно обчислювати значення цього максимуму за відомих значень $\left(\underline{\mu_i^h},\overline{\mu_i^h}\right),i=\overline{1,n}$.

При наївному переборі усіх пар (a, b) та побудові множин $V_{a,b}^h$, маємо для кожної пари проходити по масиву поміток та перевіряти, чи задовольняють вони умові потрапляння вершини у множину $V_{a,b}^h$. Після чого, якщо ця множина виявилася непорожньою, обчислювати поточне значення аргументу максимуму. Складність описаного алгоритму складає $O\left(\underline{l^h} * \overline{l^h} * n\right)$.

Використовуючи властивості множин $V_{a,b}^h$, цей показник можна покращити.

Скористаємося тим, що маємо перевірити всі проміжки $[a; b]$ по черзі. Зафіксуємо деякі значення a та b. Легко побачити, що $V_{a,b+1}^h = V_{a,b}^h \cup \left\{v_i \in \tilde{V}: \overline{\mu_i^h} = b + 1, \underline{\mu_i^h} \geq a\right\}$ та $V_{a+1,b}^h = V_{a,b}^h \setminus \left\{v_i \in \tilde{V}: \overline{\mu_i^h} \leq b, \underline{\mu_i^h} = a\right\}$.

Обчислимо спочатку числовий масив C_h, де на кожному місці $k = 1, \dots, \overline{l^h}$ буде знаходитись кількість вершин, для яких $\overline{\mu_i^h} = k$, тоді $\left|V_{1,b}^h\right| = \sum_{k=1}^{b} C_h[k]$. Це дає змогу обчислити значення виразу для всіх значень b за лінійний час.

Для переходу до $a = 2$ маємо виключити з розгляду всі вершини, для яких $\underline{\mu_i^h} = 1$. Для цього достатньо у масиві C_h зменшити відповідні значення на кількість вершин $v_i: \overline{\mu_i^h} = k, \underline{\mu_i^h} = 1$. Після цього матимемо $\left|V_{2,b}^h\right| = \sum_{k=1}^{b} C_h[k]$. Продовжуючи таким чином знайдемо шуканий максимум.

Помітимо також, що якщо попередньо відсортуємо масив позначок за зростанням $\underline{\mu_i^h}$, то кожне значення масиву потрібно буде використати лише 1 раз і складність отриманого алгоритму буде $O\left(\underline{l^h} * \overline{l^h} + n\right)$.

Псевдокод запропонованого алгоритму наведений нижче.

Алгоритм 2. Обчислення оцінки для довільного графу

Вхідні дані: ширина упорядкування h,

лексикографічно відсортований масив пар $L = \left\{\left(\underline{\mu_i^h}, \overline{\mu_i^h}\right), i = \overline{1, n}\right\}$.

Вихідні дані: значення оцінки знизу $\tilde{l}$.

```
1:  procedure enh_lower_estimate(L, h)
2:      μ̄_min ≔ min_i {μ̄_i^h}; μ̄_max ≔ max_i {μ̄_i^h};
3:      for k ≔ μ̄_min to μ̄_max
4:          C_h[k] ≔ 0;
5:      end for
6:      for i ≔ 1 to n
7:          C_h[μ̄_i^h] ≔ C_h[μ̄_i^h] + 1;
8:      end for
9:
10:     l̃ ≔ 0; j ≔ 1;
11:     while j ≤ n
12:         cnt ≔ 0;
13:         for k ≔ μ̄_min to μ̄_max
14:             cnt ≔ cnt + C_h[k];
15:             if cnt > 0 then
16:                 l̃ ≔ max{l̃, μ_j^h − 1 + ⌈cnt/h⌉ + μ̄_max − k};
17:             end if
18:         end for
19:         while j ≤ n and (j = 1 or μ_{j−1}^h = μ_j^h)
20:             C_h[μ̄_j^h] ≔ C_h[μ̄_j^h] − 1;
21:             j ≔ j + 1;
22:         end while
23:     end while
24:
25:     return l̃;
26: end procedure
```

У випадку, коли упорядкування $\underline{S^h}$ та $\overline{S^h}$ співпадають (всі вершини знаходяться на критичних шляхах), обчислення може бути пришвидшене до лінійної складності $O(\underline{l^h})$, оскільки $\underline{\mu_i^h} = \overline{\mu_i^h}, i = \overline{1,n}$ і кількість вершин у $V_{a,b}^h$ можна обчислити, підсумувавши кількості вершин на місцях з a по b в упорядкуванні $\underline{S^h}$ ($\overline{S^h}$).

Цільова функція приймає вигляд $\max\limits_{1\le a\le b\le \underline{l^h}} \left\{(a-1) + \left\lceil \frac{1}{h}\sum_{i=a}^{b} \left|\underline{S^h}[i]\right| \right\rceil + \left(\overline{l^h} - b\right)\right\}$.

Помітимо, що елемент $\overline{l^h}$ є сталим, а $l = (b - a + 1)$ дорівнює кількості місць, що приймає участь у підсумуванні, тоді пошук максимуму зводиться до наступної задачі:

$$\max_{1\le a\le b\le \underline{l^h}} \left\{ \left\lceil \frac{1}{h}\sum_{i=a}^{b} \left|\underline{S^h}[i]\right| \right\rceil - l \right\} = \max_{1\le a\le b\le \underline{l^h}} \left\lceil \frac{1}{h}\sum_{i=a}^{b} \left|\underline{S^h}[i]\right| - l \right\rceil = \max_{1\le a\le b\le \underline{l^h}} \left\lceil \frac{1}{h}\left(\sum_{i=a}^{b} \left|\underline{S^h}[i]\right| - l * h\right) \right\rceil =$$

$$= \max_{1\le a\le b\le \underline{l^h}} \left\lceil \frac{1}{h}\sum_{i=a}^{b}\left(\left|\underline{S^h}[i]\right| - h\right)\right\rceil = \left\lceil \frac{1}{h}\max_{1\le a\le b\le \underline{l^h}}\sum_{i=a}^{b}\left(\left|\underline{S^h}[i]\right| - h\right)\right\rceil.$$

Тобто отримали задачу пошуку безперервної послідовності з елементами $\left|\underline{S^h}[i]\right| - h$, яка має максимальну суму. Така задача ефективно розв'язується за допомогою алгоритму Кадана [113], складність якого і складає $O\left(\underline{l^h}\right)$.

Псевдокод цього алгоритму наведено нижче.

Алгоритм 3. Обчислення оцінки для графу, всі вершини якого знаходяться на критичних шляхах

Вхідні дані: ширина упорядкування h,

масив потужностей місць упорядкування $\underline{S_i^h}$: $C = \left\{\left|\underline{S_i^h}\right|, i = \overline{1, \underline{l^h}}\right\}$.

Вихідні дані: значення оцінки знизу $\tilde{l}$.

```
1:  procedure enh_lower_estimate_cp(C, h)
2:      cnt := 0; l̃ := l^h; l := 0;
3:      for i := 1 to l^h
4:          cnt := cnt + C[i];
5:          l := l + 1;
6:          if cnt ≥ l * h then
7:              l̃ := max(l̃, ⌈cnt/h⌉ + l^h − l);
8:          else
9:              cnt := 0; l := 0;
10:         end if
11:     end for
12:
13:     return l̃;
14: end procedure
```

Зазначимо окремо, що аналогічні процедури можуть бути використані для обчислення уточненої оцінки знизу для задачі $P_S(G, l, h)$. Дійсно, використовуючи помітки $\underline{\mu_i}$ та $\overline{\mu_i}$ для визначення множин $V_{a,b}^h$, по аналогії з оцінкою з підпункту 1.4.2.1, отримаємо наступний вираз:

$$h^* \ge \max_{1\le a\le b\le \underline{l}} \left\lceil \frac{1}{(b-a+1) + \left(l - \underline{l}\right)} \left|V_{a,b}^h\right| \right\rceil.$$

Повернемося до розгляду отриманої оцінки знизу довжини упорядкування. Вона так само може бути використана для обчислення поміток $\underline{\mu_i^h}$ та $\overline{\mu_i^h}$. При цьому,

наприклад під час обчислення поміток $\underline{\mu_i^h}$, маємо попередньо визначати помітки $\overline{\mu_i^h}$ для графів, що складаються з вершин попередніх рівнів (в якості поміток $\underline{\mu_i^h}$ цих вершин можемо використати вже обчислені на попередніх етапах значення), а для цього потрібно знайти помітки $\underline{\mu_i^h}$ для їх підграфів, і так далі. Таким чином, отримуємо рекурсивну серію викликів, вихід з якої відбувається, коли всі вершини підграфу отримають відповідні помітки. Через це складність обчислення оцінки для вихідного графу може перетворитися на експоненційну.

Розглянемо серію таких викликів на прикладі графів-ланцюжків. Зрозуміло, що в цьому випадку матимемо найбільшу кількість рекурсивних викликів, оскільки на кожному рівні знаходиться лише одна вершина.

Приклад послідовності обчислень проміжних поміток при обчислені поміток $\underline{\mu_i^h}$ для ланцюжка з 5 вершин зображено на рис. 2.15. Кожен стовпчик відображає виклики при визначенні помітки $\underline{\mu_i^h}$ для вершини відповідного рівня вихідного графу. У першому рядку знаходиться поточна вершина та її попередні рівні, в наступних – підграфи, для яких виконуються проміжні етапи обчислення. Колами позначені вершини, для визначення поміток яких робиться рекурсивний виклик. Номер вершини, для якої на поточному етапі було визначено помітку, позначено стрілочкою згори. Напрям стрілки вказує на те, яка помітка була обчислена: вправо – помітка $\underline{\mu_i^h}$, вліво – помітка $\overline{\mu_i^h}$. Вихід з рекурсії відбувається на етапі, коли над номером вершини, яка її запустила, з’явилась стрілочка.

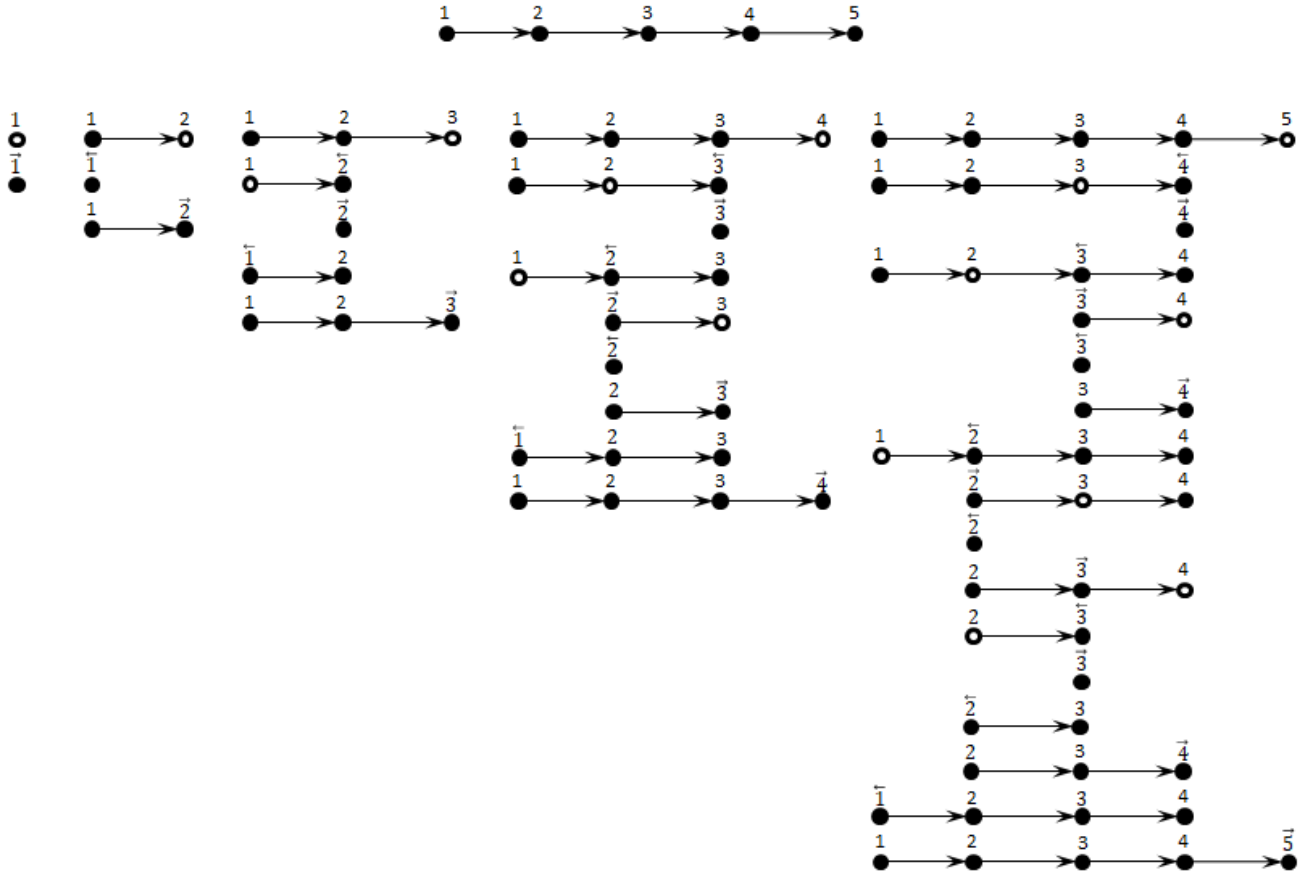

Рис. 2.15. Послідовність рекурсивного визначення поміток для ланцюжка

Бачимо, що були розглянуті усі комбінації послідовних вершин (рівнів) та кількість кроків у кожному стовпчику є наступним ступенем двійки. Отже, загальна їх кількість для визначення однієї помітки вершини рівня k складає $2^k - 1$.

В справедливості цієї залежності легко переконатися, оскільки для обчислення помітки $\underline{\mu_k^h}$ потрібно попередньо визначити помітки $\underline{\mu_i^h}$ та $\overline{\mu_i^h}$ всіх попередніх рівнів та обчислити значення оцінки знизу. Тоді, за методом математичної індукції матимемо: $(2^{k-1} - 1) + (2^{k-1} - 1) + 1 = 2^k - 1$. І загальна кількість кроків для визначення оцінки довжини для вихідного графу складає $2^{k+1} - 1$.

Отже, в цьому випадку отримаємо оцінку, обчислення якої має експоненційну складність. У загальному випадку можна очікувати, що кількість проміжних кроків складатиме $O(2^{\underline{l}})$, де $\underline{l}$ – довжина критичного шляху у звичайному сенсі, оскільки визначення поміток ведеться за рівнями графу. При цьому значно скоротити цей

показник не вдасться, оскільки, як видно з рис. 2.15, повтори проміжних графів майже не зустрічаються, тому нема сенсу запам’ятовувати помітки для підграфів.

З іншого боку, можемо обмежити глибину рекурсії та, починаючи з деякого рівня вкладеності, почати використовувати оцінки знизу довжини упорядкування, для обчислення яких потрібні лише значення поміток $\underline{\mu_i^h}$ або $\overline{\mu_i^h}$. Такий підхід дозволяє зберегти поліноміальну складність обчислення та одночасно балансувати між швидкістю обрахунку та точністю оцінки. Таким чином, отримали послідовність оцінок довжини, кожна з яких, гіпотетично, є точнішою за попередні.

Відкритими залишаються питання точності значення, отриманого оцінкою з необмеженою глибиною рекурсії, у загальному випадку, та існування відомих спеціальних класів графів, для яких оцінка є точною.

Відзначимо також, що на основі отриманої необхідної умови існування щільного упорядкування може бути побудовано наближений алгоритм, заснований на методі гілок та меж. Його ідея полягає в обмеженні глибини пошуку у дереві варіантів, шляхом обрання на кожному кроці одного чи декількох найбільш перспективних напрямків розгалуження. Нехай маємо спочатку дерево варіантів, глибина якого при побудові була обмежена деяким фіксованим значенням d. Оскільки від початку знаємо довжину шуканого упорядкування, то можемо одразу виключити з розгляду всі гілки, для яких отримана оцінка довжини перевищує ці значення. Розглянемо тепер вузли дерева варіантів на глибині 1 та множини усіх їх нащадків на глибині d. Оберемо фіксовану кількість вузлів першого рівня, множини нащадків яких задовольняють деякому критерію перспективності, та повторимо всі наведені міркування для них, розглядаючи їх в якості коренів нових дерев варіантів.

До критеріїв перспективності можна віднести, наприклад, кількість нащадків та кількість відкритих вершин у них, з врахуванням того, на яких рівнях у відповідних підграфах вони знаходяться, чи без. Відмітимо, що довжини проміжних упорядкувань всіх нащадків будуть однаковими, тому розглядати їх нема сенсу.

2.5 Висновки до розділу

В розділі отримано ряд нових результатів для класичної задачі оптимального упорядкування, які пов’язані з модифікацією та покращенням як точних, так і наближених алгоритмів розв’язання.

Доведені твердження, що дозволяють звести задачу оптимального упорядкування вершин довільних графів при заданій ширині до задач із графами, для яких існують щільні упорядкування, та з парною шириною. За допомогою аналогічних міркувань продемонстровано взаємозв’язок між *задачами 1 та 2*.

Для алгоритму, заснованого на максимальному паросполученні, теоретично обґрунтовано, що його доцільно застосовувати для вирішення задач з щільними упорядкуваннями при парній ширині упорядкування. Розроблені декілька модифікацій класичного алгоритму, пов’язані з можливістю розбиття пар та перенумерацією вершин як за рівневим принципом, так і випадковим чином.

Експериментально встановлено, що ці модифікації значно збільшують його точність при застосуванні до цього класу задач, порівняно з класичним варіантом. Також проведено порівняння з відомими алгоритмами, заснованими на рівневому принципі (алгоритми, засновані на лексикографічному та подвійному помічені), яке показало, що запропонований алгоритм має співставні з ними показники ефективності. Окрім цього, при спільному застосуванні всіх трьох алгоритмів, вдалося отримати значення точності вище за 98.5% для графів розмірністю до 100 вершин.

Запропоновано новий підхід до скорочення перебору у методі гілок та меж, який полягає у виключенні гілок, що відповідають ізоморфним підграфам. Продемонстровано, що цей підхід якісно відрізняється від скорочення перебору за рахунок уточнення оцінок знизу довжини упорядкування та доповнює його. Розглянуто ряд тверджень, пов’язаних із взаємозв’язком між властивостями вершин, що видаляються з графу, та ізоморфністю відповідних їм підграфів.

Розроблений алгоритм, який використовує інформацію про наявність у графі автоморфізму для скорочення кількость розгалужень, які підлягають перевірці у

методі гілок та меж. Теоретично обґрунтовано, що цей алгоритм не втрачає розгалуження, що відповідають неізоморфним графам, проте не дозволяє повністю позбавитися від ізоморфних підграфів.

Твердження, пов’язані з цим алгоритмом, окремо розглянуті для класу послідовно-паралельних графів, оскільки цей клас є практично значущим, та для нього проблема великої кількості ізоморфних проміжних підграфів стоїть особливо гостро. Доведена необхідна та достатня умова існування автоморфізму для таких графів та показано, що запропонований алгоритм для цього класу графів також не здатен повністю позбутися гілок, що відповідають ізоморфним підграфам.

Окремо розглянуто наближений варіант цього підходу, який спирається на використання потужних інваріантів графу замість визначення наявності автоморфізму. Проілюстровано, що можемо втратити розгалуження, які відповідають неізоморфним графам з різною довжиною упорядкування. Проте і в цьому випадку може бути знайдений шуканий розв’язок.

Сформульовано та досліджено низку необхідних умов існування щільного упорядкування, пов’язаних з обмеженістю потужності місць та можливістю їх заповнення. Ці умови вдалося об’єднати в одну та запропонувати ефективні алгоритми її перевірки в загальному випадку та для випадку, коли всі вершини графу знаходяться на критичних шляхах.

Окрім цього, запропоновані узагальнення спеціальних упорядкувань $\underline{S}$ та $\overline{S}$, які враховують значення ширини h, використання яких дозволяє уточнити відомі оцінки знизу довжини упорядкування. Також на основі необхідної умови існування щільного упорядкування отримана нова оцінка, що є узагальненням інших відомих оцінок довжини, та нова оцінка для задачі з невідомою шириною упорядкування.

Комбінування узагальнених спеціальних упорядкувань та запропонованої оцінки дозволяє отримати послідовність оцінок знизу довжини упорядкування, кожна наступна з яких, теоретично, є точнішою за попередні. Показано, що складність обчислення таких оцінок у загальному випадку може бути експоненційною.

Запропоновано наближений алгоритм розв’язання задачі для щільних графів, що базується на методі гілок та меж з обмеженою глибиною пошуку.

Основні результати розділу опубліковані у [1,3,4,6-8].

Розділ 3. РЕЗУЛЬТАТИ ДЛЯ УЗАГАЛЬНЕНИХ ЗАДАЧ ПАРАЛЕЛЬНОГО УПОРЯДКУВАННЯ

3.1 Зв'язок задачі зі змінною шириною упорядкування та класичної задачі

Доведемо твердження, що встановлює зв'язок між *задачею 1а* зі змінною шириною упорядкування та класичною *задачею 1*.

Твердження 8. Якщо існує точний алгоритм A поліноміальної складності для деякої фіксованої ширини упорядкування $\tilde{h}$, тоді існує поліноміальний алгоритм і для *задачі 1а*, в якій всі $h_i < \tilde{h}, i = \overline{1,n}$.

Доведення. Нехай маємо деякий непорожній граф G, що має n вершин, та деякі фіксовані значення $h_i < \tilde{h}, i = \overline{1,n}$. Зрозуміло, що довжина його оптимального упорядкування l^* буде знаходитися у проміжку від 1 (якщо $n \leq h_1$ та всі вершини є ізольованими) до n (граф є ланцюжком) включно.

Побудуємо новий граф G', який отримаємо з графу G шляхом додання до нього спеціального графу H_l, всі вершини якого знаходяться на критичних шляхах, з деякою кількістю рівнів $l \in \overline{1,n}$. Окрім цього, граф H_l має на кожному рівні $(\tilde{h} - h_i), i = \overline{1,l}$ вершин відповідно, та будь-який його власний підграф, утворений з вершин двох сусідніх рівнів, є повним дводольним графом. Побудуємо тепер упорядкування S графу G' при $h = \tilde{h}$ за алгоритмом A. Відомо, що воно буде оптимальним і може бути отримане за поліноміальний час.

Зрозуміло, що якщо $l = l^*$, тоді довжина S теж буде l^*. З одного боку вона не може бути більшою за l^*, оскільки маємо $(\tilde{h} - h_i)$ вершин відповідних рівнів, які займають по $(\tilde{h} - h_i)$ позицій на кожному з l^* місць в упорядкуванні, а вершини, що належать графу G, можна розмістити на l^* місцях з місткостями $h_i, i = \overline{1,l^*}$. А з іншого не може бути меншою, аніж l^*, оскільки рівні можемо розміщувати лише у лінію, а їх кількість l^*. Отже, якщо видалимо з S усі вершини, що належать графу H_l, то отримаємо оптимальне упорядкування S^* для вихідної задачі за поліноміальний час, оскільки додання та видалення вершин цього графу можна виконати за лінійний час.

Отже, якщо зможемо знаходити кількість рівнів $l = l^*$ у графі H_l, яка у загальному випадку є невідомою, за поліноміальний час, то зможемо використовувати алгоритм A для знаходження оптимальних упорядкувань для *задачі 1a* з $h_i < \tilde{h}, i = \overline{1,n}$. Це можна зробити за допомогою бінарного пошуку.

Помітимо, що $l = l^*$ відповідає мінімальному значенню l, для якого в отриманому упорядкуванні S за алгоритмом A для графу G' при $h = \tilde{h}$ на всіх місцях буде знаходитись по $(\tilde{h} - h_i)$ вершин з відповідних рівнів. Дійсно, для $l > l^*$ довжина S буде співпадати з l, оскільки потрібно розмістити l рівнів графу H_l, а вершини графу G можна розмістити й на меншій кількості місць. Для $l < l^*$ у S будуть місця, де не буде $(\tilde{h} - h_i)$ вершин графу H_l, інакше вершини граф G можна було б розмістити на меншу кількість місць, аніж l^* при $h = h_i$.

Застосуємо бінарний пошук до розв'язання цієї задачі. Оберемо деяке значення $l_0 \in \overline{a_0, b_0} = \overline{1,n}$, що розбиває цей проміжок приблизно навпіл. Будуємо граф G', для нього за алгоритмом A знаходимо оптимальне упорядкування S при $h = \tilde{h}$. Якщо у S на кожному місці стоїть $(\tilde{h} - h_i)$ вершин графу H_l, то обираємо, аналогічно, $l_1 \in \overline{a_1, b_1} = \overline{1, l_0}$ та продовжуємо пошук на цьому проміжку. Якщо ж у S є місця, на яких стоїть менше ніж $(\tilde{h} - h_i)$ вершин графу H_l, то знаходимо $l_1 \in \overline{a_1, b_1} = \overline{l_0 + 1, n}$ та аналогічно продовжуємо пошук. Збіжність алгоритму випливає зі збіжності бінарного пошуку для монотонних функцій. Пошук потребує логарифмічного часу, перевірка може бути виконана за лінійний час, а отже знайдемо шукане $l = l^*$ за поліноміальний час.

Оскільки усі кроки описаного алгоритму можна виконати за поліноміальний час, алгоритм A має поліноміальну складність за умовою, то таким чином отримали алгоритм, що є точним та має поліноміальну складність для *задачі 1a* з $h_i < \tilde{h}, i = \overline{1,n}$. ■

Наслідок 1 з твердження 8. Якщо існує точний алгоритм A поліноміальної складності для паралельно-послідовних графів та деякої фіксованої ширини

упорядкування $\tilde{h}$, тоді існує поліноміальний алгоритм і для *задачі 1а*, в якій всі $h_i < \tilde{h}, i = \overline{1,n}$, для графів з цього класу.

Доведення. Безпосередньо випливає з *твердження 8* і того, що граф H_l є паралельно-послідовним за побудовою. ■

Важливо відмітити, що застосування аналогічних міркувань дозволяє використовувати оцінки знизу довжини упорядкування для *задачі 1* до *задачі 1а* у методі гілок та меж.

Для цього необхідно спочатку обчислити значення $\tilde{l}$ базової оцінки для *задачі 1а* для цього графу (див. підпункт 1.4.2.1). Після чого обрати деяке значення $\tilde{h} > h_i, i = \overline{1,\tilde{l}}$, додати до вихідного графу спеціальний граф $H_{\tilde{l}}$ та застосувати деяку оцінку знизу для *задачі 1* при $h = \tilde{h}$ до отриманого графу. Якщо отримане значення $\widetilde{l_1} > \tilde{l}$, то й для розташування вершин початкового графу потрібно принаймні $\widetilde{l_1}$ місць, оскільки додання графу $H_{\tilde{l}}$ не впливає на довжину оптимального упорядкування. Описану процедуру потрібно повторювати доти, доки нове отримане значення не співпаде з попереднім.

3.2 Використання рівневого принципу в узагальнених задачах паралельного упорядкування

Оскільки для узагальненої задачі паралельного упорядкування зі змінною шириною (*задача 1а*) не знайдено точних поліноміальних алгоритмів з постійним показником степеня навіть для найпростіших випадків, зокрема дерев, було вирішено розглянути цю задачу для вхідних (кореневих) бінарних дерев, як найпростіших нетривіальних представників дерев. Тому підрозділ присвячений саме цьому класу графів, спробам віднайти якісь залежності й властивості, що дозволять розробити точний поліноміальний алгоритм чи, принаймні, покращити існуючі наближені, та провести обчислювальний експеримент для практичного підтвердження результатів. Виділивши фундаментальні властивості задач для бінарних дерев, у перспективі, є можливість узагальнити їх на увесь клас дерев.

3.2.1 Аналіз випадків порушення оптимальності алгоритму, заснованого на рівневому принципі

При використанні алгоритму, заснованого на рівневому принципі, для розв'язання *задачі 1а*, він буде наближеним навіть для дерев, більш того він залишається наближеним і для бінарних дерев.

Отриманий розв'язок може бути неоптимальним з двох причин:

1) через неоднозначність вибору вершин з однаковими мітками;

2) через те, що у оптимальному упорядкуванні вершини, які мають менші мітки, стоять раніше аніж вершини з більшими.

Розглянемо приклади порушення оптимальності алгоритму, заснованого на рівневому принципі, для бінарних дерев з обох причин.

Приклад 3.1. Застосуємо рівневий принцип для побудови $P_S(G, h_i, l)$ для графу з рис. 3.1 та $h_i = 2,3,1,1,1, \dots$.

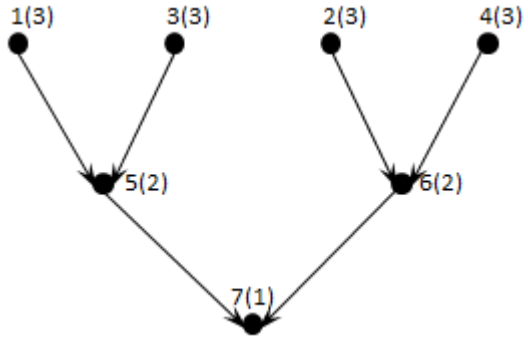

Рис. 3.1. Зображення графу з прикладу 3.1

Маємо 4 вершини, які мають мітку рівну 3, це вершини 1, 2, 3 та 4. На перше місце можна поставити $h_1 = 2$ вершин, які, згідно з алгоритмом, ми обираємо довільно. Виберемо, наприклад, вершини 1 та 2, при такому виборі не відкривається жодна вершина. Тоді на друге місце можемо поставити лише 2 вершини з $h_2 = 3$ можливих, а саме 3 і 4. Це відкриє вершини 5 та 6. Оскільки, $h_3 = h_4 = h_5 = 1$, то решту вершин можемо послідовно поставити лише по одній на кожне місце. Отже, отримаємо наступне упорядкування:

$$S = \left\{\begin{matrix}1 & 3\\ 2, & 4\end{matrix}, 5,6,7\right\}.$$

Отримане упорядкування має довжину $l = 5$. Проте, якщо замість вершин 1 та 2 на першому кроці обрати вершини 1 та 3, то це відкриє вершину 5, і тоді можна буде поставити на 2 місце вже 3, а не 2 вершини. Матимемо наступне:

$$S^* = \left\{\begin{matrix}1 & 2\\ 3, & 4, 6,7\\ & 5\end{matrix}\right\}.$$

Таке упорядкування вже буде оптимальним, і його довжина $l^* = 4$ (меншу довжину отримати неможливо, оскільки перші три місця вміщують не більше 6 вершин, а у графі їх 7). Бачимо, що отримали неоптимальний розв'язок через довільність вибору вершин з однаковими мітками.

Приклад 3.2. Застосуємо рівневий принцип для побудови $P_S(G, h_i, l)$ для графу з рис. 3.2 та $h_i = 2,2,4,2,1,1,1,1, \dots$.

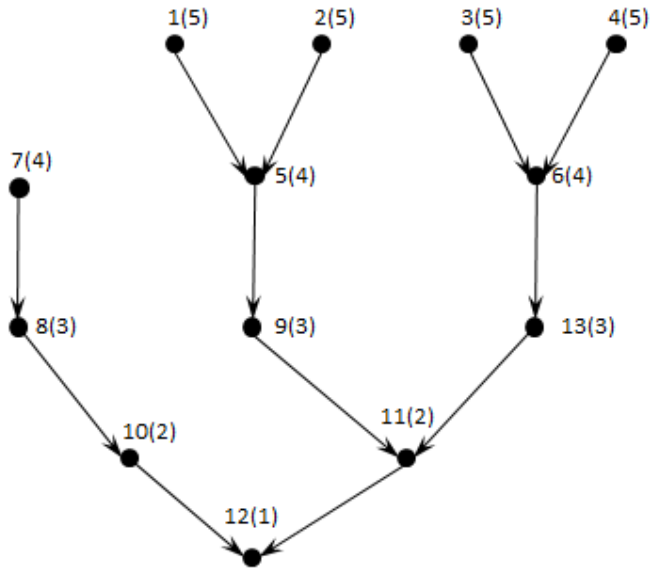

Рис. 3.2. Зображення графу з прикладу 3.2

У цьому випадку, згідно з рівневим принципом, перші два місця потрібно заповнити вершинами 1-4 (по дві на кожне), що відкриє вершини 5 та 6. Потім на 3 місце зможемо поставити лише 3 вершини з $h_3 = 4$: 5, 6 та 7 вершини. Після чого на 4 та 5 місцях розташовуємо вершини 8, 9 та 13. Вершини, що залишилися (10, 11 та 12) розташовуємо послідовно на місцях з 6 по 8. Врешті решт, отримаємо:

$$S = \begin{Bmatrix} 1 & 3 & 5 & 8 \\ 2, & 4, & 6, & 9, \\ & & 7 & \end{Bmatrix} 13,10,11,12\Big\}.$$

Довжина такого упорядкування $l = 8$. Оптимальне ж упорядкування отримаємо у випадку, якщо на перше місце поставимо 1 та 7 вершину, а на друге – 2 і 8, тоді третє місце зможемо заповнити повністю вершинами 3, 4, 5 та 10, потім відкритими залишаться вершини 6 і 9, які й поставимо на 4 місце, а вершини 13, 11 та 12 займуть місця з 5 по 7. Отже, маємо наступне:

$$S^* = \begin{Bmatrix} & & 3 & \\ 1 & 2 & 4 & 6 \\ 7, & 8, & 5, & 9, \\ & & 10 & \end{Bmatrix} 13,11,12\Big\}.$$

Довжина такого упорядкування $l^* = 7$ буде оптимальною. У цьому випадку причиною неоптимальності було те, що вершини 7 та 8, які мають мітки 4 стоять у оптимальному упорядкуванні раніше, аніж вершини 3 та 4, які мають мітки рівні 5.

З прикладів 3.1 та 3.2 бачимо, що алгоритм, заснований на рівневому принципі, перестає бути точним навіть для бінарних дерев. І якщо довільність вибору можна було б скорегувати шляхом переходу до якогось ізоморфного графу, для якого отримане упорядкування було б оптимальним, то друга причина вказує на неспроможність методу як такого, що може потенційно точно розв'язувати *задачу 1а*. Отже, розв'язання цієї задачі, хоча б для бінарних дерев, потребує розробки нових підходів та алгоритмів, які будуть більш детально враховувати структуру графу.

Тим не менш можна довести наступне твердження.

Для зручності математичного викладення введемо наступні функції.

Функція $f : V \to V \cup \{\varnothing\}$ така, що $f(v_i) = v_j \Leftrightarrow (i, j) \in U$, $f(v_i) = \varnothing \Leftrightarrow \forall j : (i, j) \notin U$. Для кореневого дерева ця функція є однозначно визначеною. Будемо позначати $f^p(\cdot) = \underbrace{f(f(\ldots f(\cdot)))}_{p}$.

Функція $g: V \to 1,..., \bar{l}$ така, що $g(v_i) = j \Leftrightarrow v_i \in \bar{S}[j]$, де $\bar{S}$ – спеціальне упорядкування, в якому кожна вершина займає крайнє праве допустиме місце в упорядкуванні, довжина якого $\bar{l}$ дорівнює довжині критичного шляху у графі.

Твердження 9. Для довільного кореневого дерева у випадку $h_i \in \{1,2\}$ алгоритм, заснований на рівневому принципі, дає оптимальний розв'язок.

Доведення. Згідно із алгоритмом, заснованим на рівневому принципі, будемо розміщувати вершини дерева так, щоб вершини наступних рівнів розміщувалися не раніше вершин з попередніх рівнів.

Нехай розміщуємо вершини рівня l графу G та намагаємося заповнити ними поточне вільне місце k упорядкування S.

Можливі наступні найпростіші ситуації: місткість $h_k = 1$ на місце k ставимо вершину $v_i : g(v_i) = l$ *(I)*, місткість $h_k = 2$ на місце k ставимо вершини $v_i, v_j : g(v_i) = g(v_j) = l$ *(IIa)*, місткість $h_k = 2$ на місце k ставимо вершини $v_i, v_j : g(v_i) = l, g(v_j) = l + 1$ *(IIb)*.

Помітимо, що якщо при виконанні алгоритму відбуваються лише розглянуті вище випадки, то поточне упорядкування є щільним, а всі вершини рівня $l+1$ є відкритими (не мають вхідних дуг).

Можливим є також випадок, коли місткість $h_k = 2$, проте існує єдина вершина $v_i : g(v_i) = l$, а серед вершин рівня $l+1$ нема відкритих *(IIc)*.

Припустимо, що при іншому порядку вибору вершин цього рівня прийшли до випадку *IIb* (інші випадки неможливі, оскільки кількість вершин на рівні є сталою). Тоді відкритою вершиною рівня $l+1$ буде $v_j = f(v_i)$, і знайдеться вершина $v_s : g(v_s) = l, f(v_s) \neq v_j$. Але, оскільки $g(v_i) = g(v_s) = l$, тоді існує єдина вершина $v_r = f(v_s) : g(v_r) = l + 1$, яка при попередньому розміщенні була закритою. Звідси випливає, що $v_r = f(v_i)$, а це неможливо, оскільки функція f є однозначною. Отримали суперечність, порядок розташування вершин рівня у цьому випадку не важливий.

Випадок *IIc* включає два варіанти: існує відкрита вершина $v_j : g(v_j) > l+1$, тоді ставимо її разом з v_i на місце k *(IIc1)*, множину вершин, обраних до розташування таким чином, позначимо $\tilde{V}$; інших відкритих вершин немає, тоді ставимо на k лише вершину v_i (*IIc2*). Тоді у випадку *IIc* усі вже розташовані вершини, окрім, можливо, вершин з $\tilde{V}$, є предками $v_j = f(v_i)$. Припустимо супротивне: серед вже розміщених вершин, які не належать $\tilde{V}$, є ті, які не є предками v_j. Припустимо, що це вершина $v_s : g(v_s) \le l$, тоді існують вершини $v_{r_p} = f^{\,p}(v_s), p = 1,..., q,\ \ q = l - g(v_s) + 1$, які не є предками v_j, а отже $v_{r_q} : g(v_{r_q}) = l+1, v_{r_q} \ne v_j$. Оскільки вершина $v_s \notin \tilde{V}$, то й $v_{r_p} \notin \tilde{V}$ (нащадки є на кожному рівні, розміщуємо рівні по черзі), а отже v_{r_q} не може бути вже розміщеною, оскільки розглядаємо рівень l, але тоді вона є відкритою, оскільки всі її предки вже розміщені (лежать на попередніх рівнях, а v_i не є предком і остання на рівні). Звідки випливає, що можемо розмістити її разом із v_i, що суперечить тому, що знаходимось у випадку *IIc*. Таким чином отримали суперечність, всі вже розташовані вершини, що не належать $\tilde{V}$, є предками v_j.

З доведеного випливає, що доки розміщуємо вершини за *I, IIa, IIb*, отримуємо щільне упорядкування. Коли зустрінеться випадок *IIc*, то порядок розташування вершин на попередніх кроках не вплине на те, що матимемо випадок *IIc* на поточному кроці. Окрім цього, вершина наступного рівня, єдина не розміщена раніше у цьому випадку, не може стояти в упорядкуванні раніше, оскільки на усіх попередніх місцях стоять її предки. З цього випливає, що отримане упорядкування буде оптимальним.

У випадку, коли $v_i : f(v_i) = \varnothing$ розміщуємо корінь дерева на місце k і завершуємо алгоритм. ■

Зауваження. Випадку *IIc2* відповідає підграф кореневого дерева, утворений графом-ланцюжком.

Повернемося до прикладів 3.1 та 3.2 порушення оптимальності алгоритму, заснованого на рівневому принципі.

У прикладі 3.1 на оптимальність вплинув порядок вибору вершин, при якому не відкрилося жодної нової. Через це на друге місце змогли поставити лише 2 вершини, а це автоматично збільшило мінімальну можливу довжину на 1. В той час у оптимальному упорядкуванні змогли заповнити усі місця повністю через те, що відкрили нову вершину після заповнення першого місця. Проте, як зазначалось раніше, цього б не відбулося, якщо вершини були б пронумеровані дещо інакше. Так наприклад, помінявши позначки другої та третьої вершини місцями та керуючись тим же принципом вибору, ми отримаємо оптимальне упорядкування.

З цього прикладу можна зробити висновок, що шуканий точний алгоритм має бути інваріантний відносно ізоморфних графів, що не виконується для алгоритму, заснованого на рівневому принципі. Хоча необхідність цієї властивості можна послабити. Наприклад, можна ввести уніфікований спосіб нумерації вершин дерева. Один з таких способів: послідовно нумерувати вершини від нижнього рівня до верхнього, а в межах рівня так, щоб дуги з вершин із найменшими позначками йшли у вершину з найменшою позначкою на попередньому рівні, наступні за малістю позначки відповідали вершинам, з яких дуги йдуть у вершину з переднайменшою позначкою на попередньому рівні і так далі. За такої перенумерації вершин графу класи еквівалентності графів значно збільшаться, проте ізоморфні графи все ще можуть належати різним класам еквівалентності. Можна додавати більше умов на порядок нумерації, наприклад, впорядковувати гілки за кількістю вершин у них. Це розширить класи еквівалентності графів і послабить необхідність інваріантності точного алгоритму відносно ізоморфних графів.

У прикладі 3.2 оптимальність порушується через те, що, незважаючи на розміщення відкритих вершини на перше та друге місце, відкриття нових вершин не дозволяє заповнити третє місце повністю. При цьому, якщо обирати вершини з нижчих рівнів можна досягти того, що третє місце буде заповнене повністю, що і бачимо у оптимальному упорядкуванні. Ця проблема набагато серйозніша, аніж попередня, оскільки, як вже зазначалося, показує, що алгоритм, заснований на рівневому принципі, не працює взагалі. Як би не змінювали позначки вершин у графі,

оптимальне упорядкування за цим методом знайти не можна. Можна припускати, що точний алгоритм має базуватися на рівневому принципі, проте на якійсь вдосконаленій його формі, яка більш детально враховує структуру графу.

З обох прикладів також прослідковується, що алгоритм має забезпечувати якомога більше відкритих вершин перед заповненням кожного місця. Цей принцип є загальним, взагалі кажучи, для усіх задач паралельного упорядкування. Дійсно, у оптимальному упорядкуванні максимально заповнені усі місця, а отже і на кожному кроці алгоритму тим краще, чим більше відкритих вершин у графі на даний момент. Підсумовуючи, можна припускати, що чим більше відкритих вершин на кожному кроці гарантує алгоритм, тим ближчим отримане упорядкування буде до оптимального.

Також обидва приклади демонструють, що усі місця необхідно максимально заповнювати, тобто немає сенсу залишати деякі відкриті вершини, якщо якісь з них ще можна поставити на поточне місце. Аналогічно до попереднього, цей принцип вірний для усіх задач паралельного упорядкування, оскільки чим більше вершин вже розташовано, тим менше залишається і тим швидше можна відкрити інші.

З боку практичної зручності, бажано, щоб алгоритм був online-алгоритмом, тобто вибір вільних вершин, які будуть поставлені на поточне місце не залежав від місткостей наступних місць. Така вимога обумовлюється наприклад тим, що заздалегідь невідомо скільки процесорів будуть вільними у подальші моменти часу. Якби точний алгоритм мав зазначену властивість, то можна було б підлаштовуватись під зміну навантаження на процесори, при цьому не втрачаючи оптимальності підсумкового упорядкування. Така властивість гарно поєднується з вимогою до максимізації кількості вільних вершин на кожному кроці, наприклад, у випадку, коли алгоритм є жадібним.

Помітимо також, що важливою особливістю усіх кореневих дерев є те, що, якщо вершині v_i передує m_i вершин, а вершині v_j – m_j вершин, і ці вершини не з’єднані орієнтованим шляхом, тоді для того, щоб відкрити вершини v_i та v_j необхідно

попередньо розмістити $m_i + m_j$ вершин, що випливає з того, що з кожної вершини виходить не більше однієї дуги.

3.2.2 Новий алгоритм знаходження наближених розв'язків узагальненої задачі паралельного упорядкування. Приклади застосування.

Використовуючи міркування, наведені у попередньому пункті, було розроблено алгоритм з наступними властивостями:

- жадібно обирає вільні вершини, які ставляться на поточне місце, так щоб максимізувати кількість вільних вершин на наступному кроці;
- враховує інформацію про те, на якому рівні знаходиться вершина, тобто спирається на рівневий принцип;
- враховує більш детальну інформацію про структуру графу, а саме до яких вершин йдуть дуги з вільних вершин графу та кількість вершин, які необхідно розташувати, щоб відкрити дану вершину;
- вибір вільних вершин для розташування на поточному місці не залежить від місткості наступних місць в упорядкуванні, тобто є online-алгоритмом.

Запропонований алгоритм

Етап 1. Попередня розстановка міток (перша мітка відповідає кількості вершин, які потрібно попередньо поставити аби відкрити дану, друга – рівень вершини у розвернутому $\bar{S}$).

1) Будується упорядкування $\bar{S}$ для графу G. Всім вершинам v_i присвоюється друга мітка рівна $\bar{l} - \mu_i + 1$, де $\bar{l}$ – довжина упорядкування $\bar{S}$, μ_i – номер місця, на якому розташована вершина v_i в упорядкуванні $\bar{S}$.

2) Вершини, що не мають вхідних дуг, отримують першу мітку рівну нулю.

3) Вершини, у які входять дуги лише з вже помічених вершин, отримують першу мітку рівну сумі міток вершин, які їм передують, плюс кількість таких вершин.

4) Повторюємо пункт 3), доки усім вершинам не буде присвоєна перша мітка.

Етап 2. Розташування вершин.

Нехай на кожному кроці i відомі величина h_i та поточний граф G.

1) Обираємо вершини $v_j\left(j = \overline{1,k}\right)$, у які входять дуги лише з відкритих вершин.

2) Сортуємо обрані вершини спочатку у порядку неспадання першої мітки, а потім незростання другої. Отримуємо список вершин $\tilde{V} = \{\tilde{v}_j\}, j = \overline{1,k}$.

3) Послідовно переглядаємо вершини з $\tilde{V}$ та розташовуємо усі відповідні їм вершини-попередники на місце i, доки це можливо.

4) Якщо i місце заповнене не повністю та у $\tilde{V}$ залишилися вершини, то заповнюємо це місце відкритими вершинами, що передують першій вершині з $\tilde{V}$.

5) Якщо i місце заповнене не повністю, у $\tilde{V}$ не залишилося вершин, але ще є відкриті вершини, які не є ізольованими, тоді обираємо вершини $\tilde{\tilde{V}} = \{\tilde{\tilde{v}}_r\}, r = \overline{1,m}$, які безпосередньо слідують за ними. Розташовуємо відкриті вершини, що передують вершинам з $\tilde{\tilde{V}}$ аналогічно до пунктів 3) та 4).

6) Якщо i місце заповнене не повністю, у списках $\tilde{V}$ та $\tilde{\tilde{V}}$ не залишилося вершин, але в графі ще є ізольовані вершини, тоді ставимо їх на це місце у будь-якому порядку доки воно не заповниться, чи ізольовані вершини не закінчаться.

7) На цьому кроці або i місце повністю заповнене, або у поточному графі G не залишилося вільних вершин. Усі розташовані на i місці вершини видаляються з G разом з вихідними дугами, якщо вони є, $G := G, i := i + 1$.

8) Якщо граф G – порожній, то *кінець*, інакше переходимо на *Етап 3*.

Етап 3. Поновлення міток.

1) Друга мітка не змінюється.

2) Для кожної вершини, яка була видалена з графу, перші мітки всіх вершин, що знаходяться на шляху від видаленої вершини до кореня дерева (вершини, знайдені пошуком у глибину від видаленої вершини) зменшуємо на одиницю.

3) Перехід на *Етап 2*.

Приклади застосування

Повернемося до прикладів 3.1 та 3.2 з пункту 3.2.1 та застосуємо до них запропонований алгоритм.

Приклад 3.3. Застосуємо запропонований алгоритм для побудови $P_S(G, h_i, l)$ для графу з рис. 3.1 та $h_i = 2,3,1,1,1, \dots$.

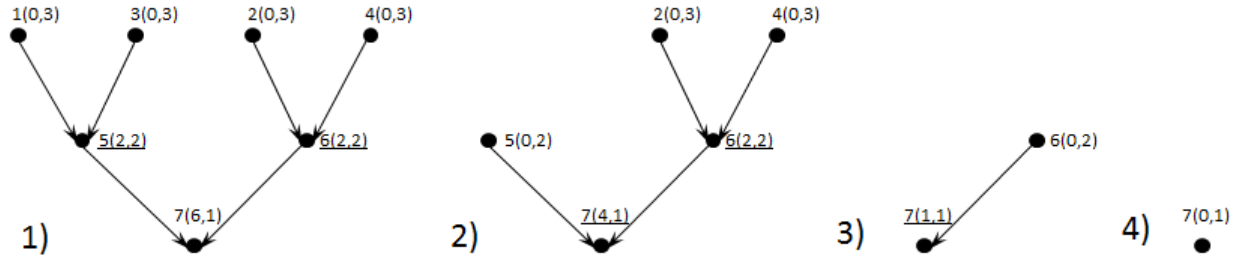

Рис. 3.3. Застосування запропонованого алгоритму до графу з рис. 3.1

На першій частині рисунку 3.3 зазначений початковий граф із розставленими мітками. До $\tilde{V}$ потрапили вершини 5 та 6 (вони підкреслені на рис. 3.3 1)), вони мають однакові першу та другу мітку, тому їх порядок у $\tilde{V}$ не важливий. Для визначеності припустимо, що спочатку стоїть вершина 5, а потім 6. Згідно з алгоритмом, на перше місце поставимо вершини 1 та 3, оскільки вони передують вершині 5. Перше місце заповнене повністю, тому видаляємо усі обрані вершини та поновлюємо позначки. Отримаємо граф з рис. 3.3 2).

Тепер до $\tilde{V}$ потрапляє лише шоста вершина, тому ставимо вершини 2 та 4 на друге місце в упорядкуванні. Друге місце заповнене не повністю, до $\tilde{\tilde{V}}$ потрапляє лише вершина 7, їй передує лише одна вільна вершина – 5, тому додаємо її на друге місце в упорядкуванні. Оскільки друге місце заповнене повністю та вільних вершин у графі не залишилось, то видаляємо вершини 2, 4 та 5 з графу і поновлюємо позначки. Отримаємо граф з рис. 3.3 3).

У $\tilde{V}$ буде лише вершина 7, тому ставимо шосту вершину на третє місце і видаляємо її з графу, поновлюючи при цьому позначки. Отримаємо граф з рис. 3.3 4).

У випадку графу з рис. 3.3 4) $\tilde{V}$ та $\tilde{\tilde{V}}$ будуть порожніми, проте граф містить ізольовану вершину, яку і ставимо на четверте місце. Неважко побачити, що отримане упорядкування збігається з оптимальним, отриманим у попередньому пункті 3.2.1.

Приклад 3.4. Застосуємо запропонований алгоритм для побудови $P_S(G, h_i, l)$ для графу з рис. 3.2 та $h_i = 2,2,4,2,1,1,1,1, \dots$.

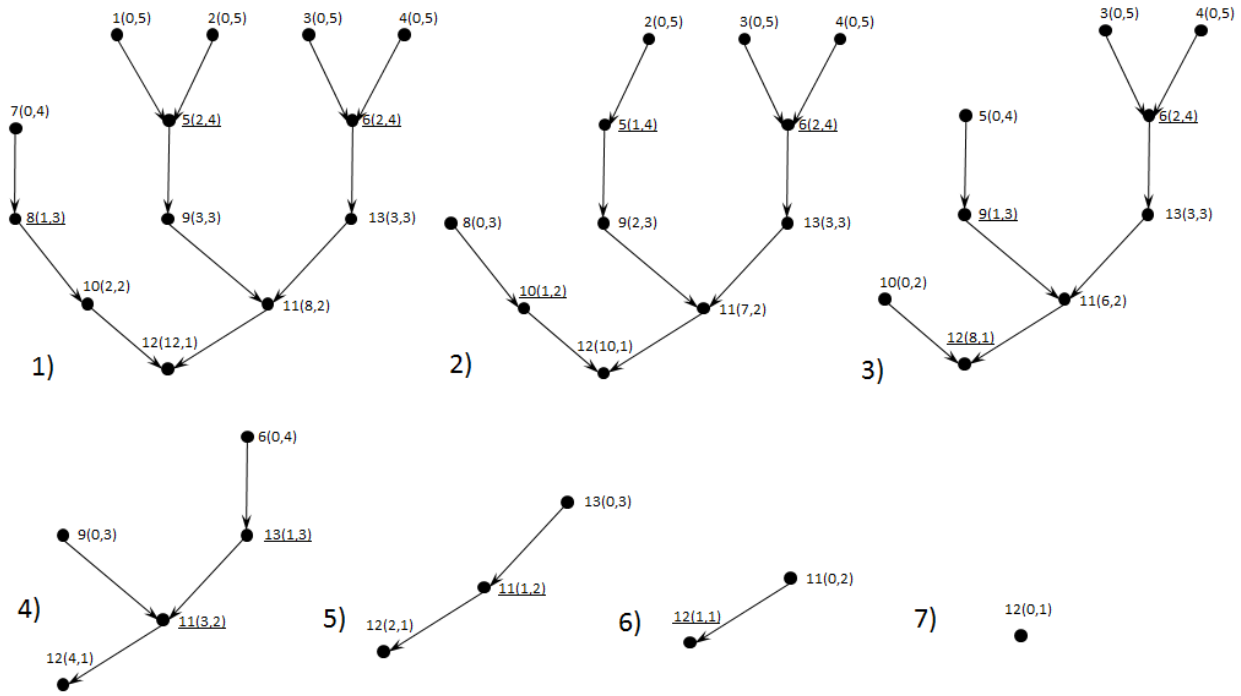

Рис. 3.4. Застосування запропонованого алгоритму до графу з рис. 3.2

На першій частині рисунку 3.4 зазначений початковий граф із розставленими мітками. Отримаємо $\tilde{V} = [8,5,6]$, тому ставимо на перше місце вершини 7 та 1. Перше місце заповнене повністю, тому видаляємо обрані вершини та поновлюємо позначки. Отримаємо граф з рис. 3.4 2).

Тепер $\tilde{V} = [5,10,6]$, отже на друге місце, згідно з алгоритмом, маємо поставити вершини 2 та 8. Таким чином, друге місце заповнене повністю. Після їх видалення та поновлення позначок матимемо граф з рис. 3.4 3).

Маємо $\tilde{V} = [9,6]$, тому ставимо на третє місце вершини 3, 4 та 5. Оскільки місткість $h_3 = 4$, то можемо розмістити ще вершини. До $\tilde{\tilde{V}}$ увійде лише вершина 12, їй передує лише 10 вершина, тому ставимо її також на третє місце. Тепер у графі немає вільних вершин, третє місце заповнене повністю, тому видаляємо вершини 3, 4, 5 та 10 з графу, поновлюючи позначки. Отримаємо граф з рис. 3.4 4).

Для поточного графу $\tilde{V} = [13]$, тому ставимо шосту вершину на четверте місце, і, оскільки на це місце можна поставити 2 вершини, тоді розглядаємо $\tilde{\tilde{V}} = [11]$. Вершині 11 передує лише вільна вершина 9, тому ставимо її на четверте місце, тим

самим заповнюючи його. Видаляючи тепер поставлені вершини, поновлюючи при цьому позначки, приходимо до графу з рис. 3.4 5).

Отримаємо $\tilde{V} = [11]$, тому заповнюємо п’яте місце повністю вершиною 13, після видалення якої та оновлення позначок отримаємо граф з рис. 3.4 6).

Тепер $\tilde{V} = [12]$, тому на шосте місце ставимо вершину 11. І аналогічно до попереднього пункту, отримуємо граф з рис. 3.4 7).

Граф з рис. 3.4 7) має лише ізольовану вершину 12, тому ставимо її на сьоме місце. Оскільки після видалення її отримаємо порожній граф, то на цьому виконання алгоритму завершується. Бачимо, що отримане за запропонованим алгоритмом упорядкування співпадає з шуканим оптимальним.

Отже, запропонований алгоритм успішно вирішує приклади, на яких порушується оптимальність алгоритму, заснованого на рівневому принципі. Це вже свідчить про те, що якщо запропонований алгоритм не буде точним у всіх випадках, коли алгоритм, заснований на рівневому принципі є точним, то, принаймні, вдасться розширити підклас задач, для яких існують точні поліноміальні алгоритми зі сталим показником степеня. Окрім цього, запропонований алгоритм реалізує всі основні виокремлені принципи з пункту 3.2.1, що дозволить при практичному тестуванні визначити їх особливості при використанні для пошуку розв’язку задачі паралельного упорядкування зі змінною шириною.

3.2.3 Порівняння запропонованого алгоритму та алгоритму, заснованого на рівневому принципі. Обчислювальні експерименти.

Розпочнемо дослідження запропонованого алгоритму з теоретичних міркувань, пов’язаних з принципами, на яких він був побудований.

Зрозуміло, що важливим фактором для його застосування є особливість дерев, зазначена у останньому абзаці пункту 3.2.1. Дійсно, якщо спробувати застосувати цей алгоритм до довільного графу (що можливо, оскільки під час кроків алгоритму не використовується, що граф є саме деревом), то жадібність перестає працювати, оскільки з вершин може виходити більше однієї дуги. Дійсно, це призводить до того, що якщо між двома вершинами немає орієнтованого шляху, то звідси не випливає,

що серед вершин, які їм передують, нема однакових. А якщо так, тоді для довільних графів потрібно перебирати всі можливі комбінації вершин, що безпосередньо слідують за відкритими, щоб знайти такі, які потребують найменшу кількість вершин для відкриття. При цьому відповідна схема породжуватиме алгоритм не обов'язково поліноміальної складності. Отже, якщо запропонований алгоритм і є точним для вхідних дерев, то можна напевно сказати, що точним для довільних графів він не буде.

Припустимо, що запропонований алгоритм буде точним для будь-яких дерев. Тоді одразу виникає питання, як з цього випливає, що алгоритм, заснований на рівневому принципі, є точним для орієнтованих дерев для *задачі 1*. Хоча запропонований алгоритм базується на дещо узагальненому рівневому принципі, проте із доведенням оптимальності алгоритму, заснованого на рівневому принципі, виникають труднощі. Це викликає підозри, що запропонований алгоритм навіть не завжди є точним, коли точним є алгоритм, заснований на рівневому принципі. До того ж, алгоритм побудовано так, щоб на кожному кроці максимізувати кількість відкритих вершин. Проте залишається питання, чи випливає з цього, що на кожному кроці буде досягнуто максимум кількості відкритих вершин за будь-яких значень місткості на наступних кроках. Великі сумніви викликає і можливість оптимального алгоритму бути online взагалі, оскільки ця властивість підказана практичною зручністю, а не аналізом конкретних прикладів. Ці складнощі теоретичної перевірки підштовхують до необхідності практичної перевірки.

Для практичної перевірки будемо порівнювати алгоритм, заснований на рівневому принципі, та запропонований алгоритм. Така пара обрана через схожість алгоритмів та однакові алгоритмічні складності (алгоритмічно найскладнішими частинами у обох алгоритмах є сортування).

Вже показано на прикладах у пункті 3.2.1, що існують приклади бінарних дерев, для яких алгоритм, заснований на рівневому принципі, не може отримати оптимальний розв'язок, а запропонований – може. Тепер експериментально перевіримо наскільки часто зустрічаються такі дерева, та чи існують графи, для яких

запропонований алгоритм не знаходить оптимальний розв'язок, а алгоритм, заснований на рівневому принципі,– так.

Для експериментального порівняння обраних алгоритмів, вони були програмно реалізовані, відповідно до їх алгоритмів, наведених у цій роботі, та протестовані, щоб впевнитись у правильності їх реалізації. Також був реалізований метод гілок та меж для того, щоб була змога порівнювати упорядкування, отримані досліджуваними алгоритмами, із точним, оптимальним.

Тестування проводилося у двох режимах: режимі порівняння алгоритмів лише між собою (без використання методу гілок та меж) та режимі порівняння точності алгоритмів (з використанням методу гілок та меж для знаходження оптимальних значень).

У режимі порівняння алгоритмів будемо генерувати бінарні дерева (вершини, яких були перенумеровані чи ні) з кількістю вершин від 10 до 100, причому цей проміжок розділимо на 4: [10, 20], [21, 40], [41, 60] та [61, 100]. Висота дерева обиралась довільно з проміжку $[\lfloor log_2 B_i \rfloor + 1, 10]$, де B_i – максимальна кількість вершин на проміжку $i, i = \overline{1,4}$. Для кожного проміжку місткості місць у паралельному упорядкуванні обираються довільно або так, щоб виконувалися умова неспадання чи незростання місткостей, зі значеннями з проміжку [1, 10]. Такий вибір множини значень місткості обумовлений тим, що на середніх обчислювальних машинах рідко буває більше десяти ядер. Для кожного проміжку було проведено по 10000 експериментів.

Усі попередні умови підсумовано у вигляді таблиці 3.1.

Таблиця 3.1. Значення параметрів, використаних при тестуванні у режимі порівняння алгоритмів

n	Висота	h_i	Кількість тестів
[10,20]	[5,10]	[1,10]	10000
[21,40]	[6,10]	[1,10]	10000
[41,60]	[6,10]	[1,10]	10000
[61,100]	[7,10]	[1,10]	10000

У режимі порівняння точності алгоритмів через обчислювальну складність методу гілок та меж, при використанні базової оцінки та маленьких значень місткостей, значення параметрів тестування були дещо зменшені. Так генерувалися випадкові бінарні дерева (вершини, яких були перенумеровані чи ні) з кількістю вершин з проміжків: [10, 20], [21, 30], [31, 40] та [41, 50]. Висота дерев та місткості обиралася за тим же принципом. Для кожного проміжку кількість експериментів було скорочено до 3000. Виконання алгоритму методу гілок та меж зупиняється, якщо кількість вершин у дереві розгалуження перевищує 1000. Таке значення обране через те, що таке обмеження помітно не впливає на відсоток графів, для яких за цих умов не встигаємо отримати оптимальне упорядкування, проте дуже впливає на час виконання обчислювального експерименту.

Усі попередні умови підсумовано у вигляді таблиці 3.2.

Таблиця 3.2. Значення параметрів, використаних при тестуванні у режимі порівняння точності алгоритмів

n	Висота	h_i	Кількість тестів	Максимальна кількість вершин у дереві розгалуження
[10,20]	[5,10]	[1,10]	3000	1000
[21,30]	[5,10]	[1,10]	3000	1000
[31,40]	[6,10]	[1,10]	3000	1000
[41,50]	[6,10]	[1,10]	3000	1000

Результати у режимі порівняння алгоритмів наведені у таблицях 3.3–3.6. Результати містять кількість тестових випадків, в яких алгоритмом, заснованим на рівневому принципі (АРП), отримано упорядкування з меншою довжиною ($l_{\text{АРП}} < l_{\text{АМ}}$), в яких запропонованим алгоритмом (ЗА) отримано упорядкування меншої довжини ($l_{\text{АРП}} > l_{\text{ЗА}}$) та в яких обома алгоритмами отримані упорядкування рівної довжини ($l_{\text{АРП}} = l_{\text{ЗА}}$). Окрім цього, таблиці містять середнє значення різниці між довжинами, отриманими за алгоритмами, у випадках, коли вони не рівні між собою.

Таблиця 3.3. Результати тестування для випадкових бінарних дерев та місткостей місць у паралельному упорядкуванні

Випадкові бінарні дерева та місткості				
$l_{\text{АРП}} < l_{\text{ЗА}}$	$l_{\text{АРП}} = l_{\text{ЗА}}$	$l_{\text{АРП}} > l_{\text{ЗА}}$	Середній відхил АРП	Середній відхил ЗА
531	9319	150	1	1.003766
1608	7927	465	1.004301	1.04602
1339	8128	533	1.005629	1.032114
409	8940	651	1.006144	1.002445

Таблиця 3.4. Результати тестування для випадкових бінарних дерев та незростаючих місткостей місць у паралельному упорядкуванні

Випадкові бінарні дерева та незростаючі місткості				
$l_{\text{АРП}} < l_{\text{ЗА}}$	$l_{\text{АРП}} = l_{\text{ЗА}}$	$l_{\text{АРП}} > l_{\text{ЗА}}$	Середній відхил АРП	Середній відхил ЗА
4	9979	17	1	1
11	9942	47	1.042553	1
0	10000	0	—	—
0	10000	0	—	—

Таблиця 3.5. Результати тестування для випадкових бінарних дерев та неспадаючих місткостей місць у паралельному упорядкуванні

Випадкові бінарні дерева та неспадаючі місткості				
$l_{\text{АРП}} < l_{\text{ЗА}}$	$l_{\text{АРП}} = l_{\text{ЗА}}$	$l_{\text{АРП}} > l_{\text{ЗА}}$	Середній відхил АРП	Середній відхил ЗА
439	9561	0	—	1.006834
1727	8273	0	—	1.020266
2895	7105	0	—	1.032124
2115	7885	0	—	1.011348

Таблиця 3.6. Результати тестування для перенумерованих бінарних дерев та випадкових місткостей місць у паралельному упорядкуванні

Перенумеровані бінарні дерева та випадкові місткості				
$l_{\text{АРП}} < l_{\text{ЗА}}$	$l_{\text{АРП}} = l_{\text{ЗА}}$	$l_{\text{АРП}} > l_{\text{ЗА}}$	Середній відхил АРП	Середній відхил ЗА
520	9381	99	1	1.017308
1754	7995	251	1	1.042189
1365	8446	189	1	1.029304
445	9431	124	1	1.002247

З результатів тестування вже бачимо, що запропонований алгоритм не може повністю замінити алгоритм, заснований на рівневому принципі. І хоча запропонований алгоритм і поступається алгоритму, заснованому на рівневому принципі, він потенційно розширює клас графів, для яких можна знайти оптимальний розв’язок *задачі 1а* для вхідних бінарних дерев за поліноміальний час зі сталим

показником степеня. Для цього потрібно використовувати обидва алгоритми, а потім у якості відповіді обирати найкраще з отриманих за цими алгоритми упорядкувань. Далі такий алгоритм будемо називати об’єднаним алгоритмом.

Результати у режимі порівняння точності алгоритмів наведені у таблицях 3.7–3.10. Результати містять кількість тестових випадків, в яких алгоритмом, заснованим на рівневому принципі, отримано точний розв’язок, в яких запропонованим алгоритмом отримано точний розв’язок та в яких об’єднаним алгоритмом (ОА) отримано точний розв’язок. Окрім цього, таблиці містять середнє значення різниці довжини, отриманої за об’єднаним алгоритмом та оптимальним значенням, а також кількість випадків, у яких метод гілок та меж з видаленням розгалужень, які відповідають ізоморфним підграфам, не встиг знайти розв’язок.

Таблиця 3.7. Результати тестування точності для випадкових бінарних дерев та місткостей місць у паралельному упорядкуванні

Випадкові бінарні дерева та місткості				
АРП точний	ЗА точний	ОА точний	Середній відхил ОА	Незавершених випадків
2936	2822	2982	1	2
2710	2477	2827	1	99
2358	1912	2485	1	449
2345	2056	2481	1	516

Таблиця 3.8. Результати тестування для випадкових бінарних дерев та незростаючих місткостей місць у паралельному упорядкуванні

Випадкові бінарні дерева та незростаючі місткості				
АРП точний	ЗА точний	ОА точний	Середній відхил ОА	Незавершених випадків
2984	2985	3000	-	0
2959	2976	2995	1	0
2967	2981	2993	1	1
2992	2995	2999	-	1

Таблиця 3.9. Результати тестування для випадкових бінарних дерев та неспадаючих місткостей місць у паралельному упорядкуванні

Випадкові бінарні дерева та неспадаючі місткості				
АРП точний	ЗА точний	ОА точний	Середній відхил ОА	Незавершених випадків
3000	2858	3000	-	0
3000	2621	3000	-	0
3000	2188	3000	-	0
3000	2082	3000	-	0

Таблиця 3.10. Результати тестування для перенумерованих бінарних дерев та випадкових місткостей місць у паралельному упорядкуванні

Перенумеровані бінарні дерева та випадкові місткості				
АРП точний	ЗА точний	ОА точний	Середній відхил ОА	Незавершених випадків
2950	2810	2981	1	1
2779	2407	2840	1	124
2460	1946	2520	1	460
2517	2067	2562	1	437

З результатів тестування можна помітити наступне:

1) У переважній більшості випадків (від 70 до 100%) упорядкування, отримані за досліджуваними алгоритмами мають однакову довжину, що дозволяє стверджувати про спроможність запропонованого алгоритму отримати точні розв’язки як задач, які не можна розв’язати за допомогою алгоритму, заснованого на рівневому принципі, так і задач, які ним розв’язуються точно.

2) Зі збільшенням кількості вершин у випадковому бінарному дереві кількість випадків, у яких запропонований алгоритм знаходить коротше упорядкування, зростає, а аналогічний показник для алгоритму, заснованого на рівневому принципі, досягає максимуму та спадає. Це свідчить про те, що запропонований алгоритм починає краще працювати для дерев, що мають більше вершин, а отже і складнішу структуру.

3) Якщо отримані за цими алгоритмами упорядкування мають різну довжину, то різниця між ними буде більшою за одиницю лише у рідкісних випадках. Це свідчить про те, що хоча алгоритм, заснований на рівневому принципі, і має взагалом кращі

показники, проте ці алгоритми є дуже близькими за ефективністю. Приклад дерева, для якого запропонований поступаються на 2 місця можна побачити на рис. 3.5.

4) При перенумерації вершин бінарних дерев ефективність рівневого принципу значно зростає. Це свідчить про дієвість такого способу покращити властивість алгоритму бути інваріантним відносно ізоморфних графів.

5) Об’єднаний алгоритм дозволяє істотно покращити показники точності алгоритму, заснованого на рівневому принципі. Його результати у всіх розглянутих випадках не відрізняються від точних більше, аніж на одиницю.

6) У випадках, коли місткості незростаючі, краще працює запропонований алгоритм, знаходячи оптимальний розв’язок у 99% випадків. Випадки, коли об’єднаний алгоритм є неоптимальним зустрічаються дуже рідко, один з них наведений на рис. 3.6.

7) У випадку, коли місткості неспадні, не було знайдено жодного випадку, коли б алгоритм, заснований на рівневому принципі був би неточним. Існує можливість, що для таких задач він буде оптимальним завжди, що потребує додаткового дослідження.

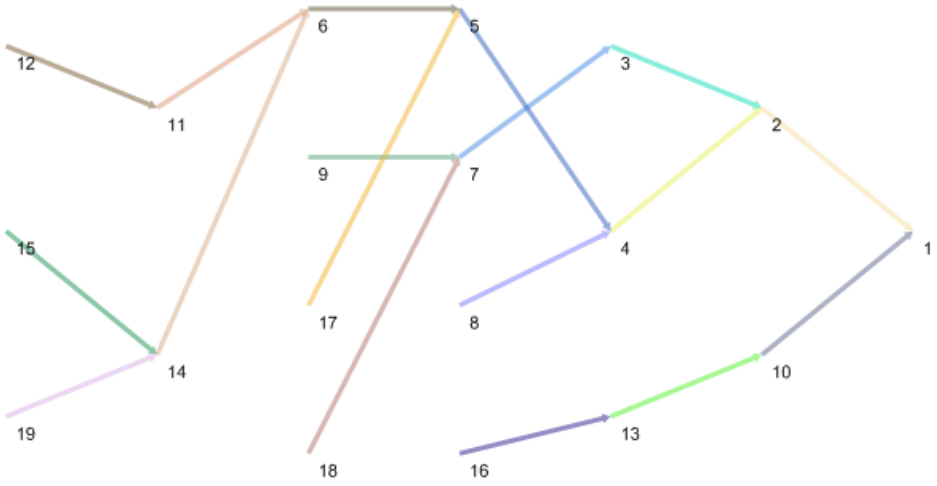

Рис. 3.5. Граф, для якого запропонований алгоритм поступається на два місця при місткостях $h_i = 8,1,1,1,10,8,3,3,4,8, \ldots$

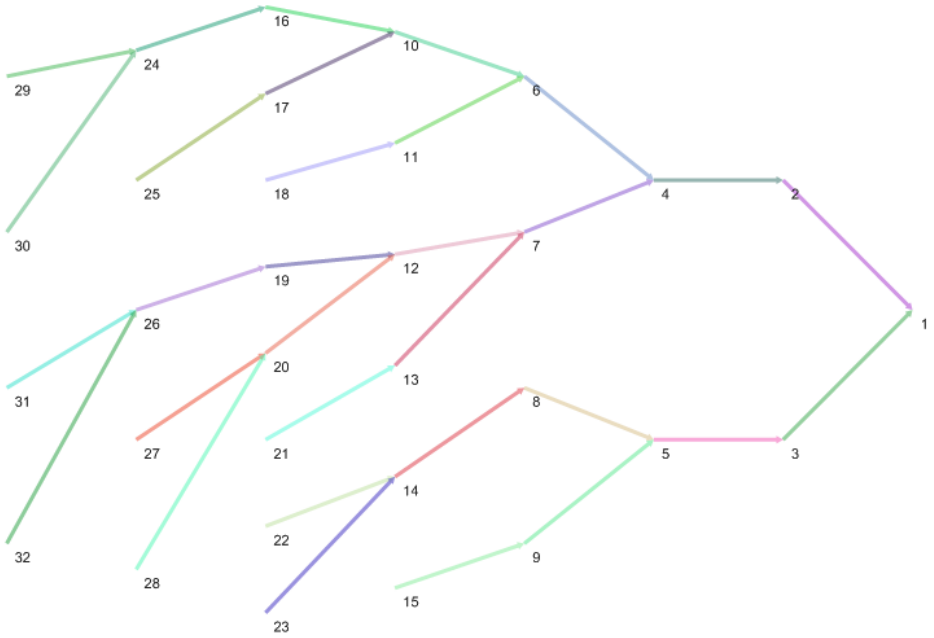

Рис. 3.6. Граф, для якого об’єднаний алгоритм дає неоптимальний розв’язок при незростаючих місткостях $h_i = 10,9,6,4,2,1,1, \ldots$

За допомогою наведеного обчислювального експерименту була підтверджена правильність теоретичних сумнівів, що виникали стосовно запропонованого алгоритму. Також бачимо, що незважаючи на це, запропонований алгоритм дуже добре себе показав під час експерименту, що також підтверджує вірність виділених принципів, яких має дотримуватися точний алгоритм.

Окрім цього, запропонований алгоритм має зручні для практичного застосування властивості і суттєво не поступається у значенні цільової функції для отриманих розв’язків як алгоритму, заснованому на рівневому принципі, так і точному розв’язку. Це каже про можливість та доцільність його використання для розподілу завдань між процесорами, коли їх кількість постійно зменшується.

Важливо відмітити, що у випадках, коли запропонований алгоритм дає неточний розв’язок, можна побачити, що потрібно враховувати структуру дерев на два, три і далі рівнів униз, що наштовхує на думки, що точний алгоритм має враховувати місткості усіх місць упорядкування.

Важливою також є проблема неінваріантності наближених алгоритмів відносно ізоморфних графів. Приклад графу, для якого кожен з алгоритмів та об’єднаний

алгоритм можуть давати оптимальні та неоптимальні упорядкування, залежно від нумерації його вершин, зображено на рис. 3.7. З експериментів бачимо, що навіть проста перенумерація вершин, застосована у цій роботі, значно покращує показники обох алгоритмів, але особливо алгоритму, заснованого на рівневому принципі. Можна припускати, що використовуючи більш складні методи перенумерації вершин можна ще покращити показники розглянутих алгоритмів.

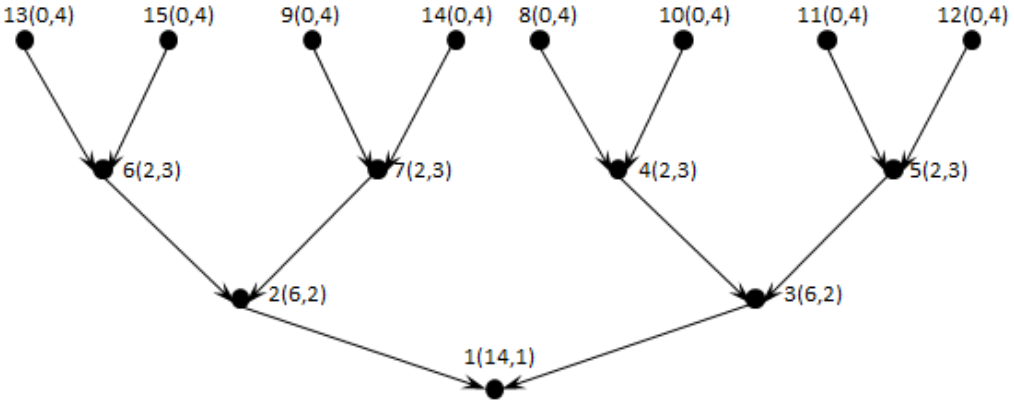

Рис. 3.7. Граф, для якого об’єднаний алгоритм дає неоптимальний розв’язок при місткостях $h_i = 4,7,7,1,5,7,6, \ldots$

Повернемося до питання можливості побудови точного online-алгоритму у загальному випадку. Виявилося, що для цього класу задач точні алгоритми не можуть мати наведену властивість навіть для випадку вихідних бінарних дерев (лісів).

Розглянемо приклад графу з рис. 3.8.

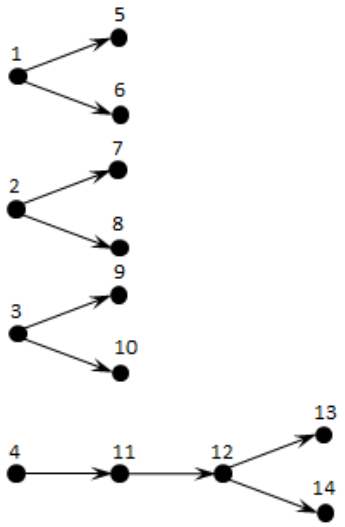

Рис. 3.8. Граф, для якого не можна побудувати online-алгоритм

При $h_i = 3,7,1,1,2,\ldots$ оптимальним розв'язком буде наступне упорядкування:

$$S_1^* = \left\{ \begin{matrix} 1 & 4 & 8 \\ 2, & 5 & 9 \\ 3 & 6 & 10 \\ & 7 & \end{matrix}, 11,12, \begin{matrix} 13 \\ 14 \end{matrix} \right\}.$$

В той же час, для $h_i = 3,2,1,8,\ldots$ матимемо:

$$S_2^* = \left\{ \begin{matrix} 1 & & & 5 & 9 \\ 2, & 3 & 12, & 6 & 10 \\ 4 & 11 & & 7 & 13 \\ & & & 8 & 14 \end{matrix} \right\}.$$

Не важко переконатися в тому, що неможливо обрати три вершини на перше місце так, щоб отримати оптимальне упорядкування в обох випадках. З цього випливає, що алгоритм має враховувати й місткості h_i наступних місць.

З іншого боку, враховуючи міркування з підрозділу 3.1, можемо пов'язати це зі змінами у структурі графу H_l для цих задач. А отже, при визначенні пріоритетності вершин тим чи іншим алгоритмом мають враховуватися не лише їх попередники та нащадки, як це відбувається в алгоритмах, заснованих на рівневому принципі.

3.3 Узагальнення задач упорядкування з урахуванням неповного завантаження

Для подальшої формалізації задач, що виникають з необхідності враховувати додаткові обмеження та умови, наведемо деякі відомі означення і нагадаємо постановки задач [41].

Задача 4 (з призначенням). По заданим графу G, значенню ширини h та вектору призначень $f_i \in \{1, \ldots, h\}$ $(i = \overline{1,n})$, який визначає виконавця, що має виконати відповідну роботу, побудувати паралельне упорядкування мінімальної довжини.

Відзначимо, що наведена задача належить до класу NP-складних при ширині упорядкування $h = 2$, навіть для графів-ланцюжків [114].

Розглянемо також деякі інші модифікації задач.

Означення 3.1. Тривалістю виконання роботи p_i називається натуральне число, що визначає кількість місць в упорядкуванні, на яких має стояти відповідна вершина.

Означення 3.2. Задачі, в яких вершини з тривалістю виконання більше одиниці можуть бути розташовані лише на суміжних місцях, називаються задачами без переривань. Якщо ця умова не є обов'язковою, тоді маємо задачу із можливістю переривання.

Відзначимо, що у випадку задачі із можливістю переривання, вона може бути зведена до еквівалентної задачі, в якій тривалості виконання всіх робіт дорівнюють одиниці, шляхом заміни вершин на ланцюжки з довжинами, рівними тривалостям виконання відповідних робіт. У подальшому будемо розглядати лише такі задачі.

Перейдемо тепер до дослідження нового класу задач упорядкування.

Нехай для деякої підмножини виконавців визначені заплановані вихідні, періоди профілактики, тощо, під час яких вони не можуть виконувати роботи, тобто на відповідних місцях в упорядкуваннях обов'язково мають бути пропуски. Зрозуміло, що у випадку, коли роботи не призначені конкретному виконавцю, отримаємо узагальнену задачу. Тому доцільним є розглядати лише випадок задачі із призначенням.

Задача 5 (з вихідними). По заданим графу G, значенню ширини h, вектору призначень $f_i \in \{1, \dots, h\}$ $(i = \overline{1,n})$ та множинам вихідних $W^j, j = \overline{1,h}$, які визначають місця, на яких не можуть бути вершини, що відповідають i-му виконавцю, побудувати паралельне упорядкування мінімальної довжини.

Розглянемо різницю між *задачами 1а, 4* та *5* на наступному прикладі.

Приклад 3.5. Нехай маємо граф G, зображений на рис. 3.9, $p_1 = 2, p_i = 1 (i = \overline{2,7})$.

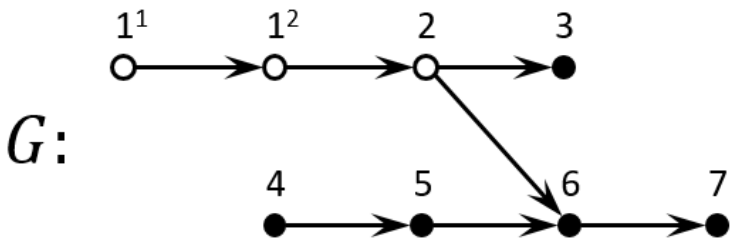

Рис. 3.9. Граф для прикладу демонстрації відмінностей між задачами

Для *задачі 1а* при $h_i = \{2,1,2,1,2\}$ отримаємо наступне оптимальне упорядкування:

$$S_1 = \left\{ \begin{matrix} 1^1 \\ 4 \end{matrix}, 1^2, \begin{matrix} 2 \\ 5 \end{matrix}, 6, \begin{matrix} 3 \\ 7 \end{matrix} \right\}, l(S_1) = 5.$$

Для *задачі 4* при $h = 2$ та $f_i = \{1,1,2,2,2,2,2\}$ (на рис. 3.9 роботи призначені для першого виконавця зображені колами, для другого – кругами) оптимальним буде наступне упорядкування (квадратами із крапками позначені вимушено порожні позиції):

$$S_2 = \left\{ \begin{matrix} 1^1, & 1^2 & 2 & \boxdot & \boxdot & \boxdot \\ 4, & \boxdot, & 5, & 6, & 3, & 7 \end{matrix} \right\}, l(S_2) = 6.$$

Розглянемо тепер *задачу 5* при $h = 2, W^1 = \{2\}, W^2 = \{4\}$ (відповідають $h_i = 1$ з прикладу до *задачі 1а*) та вектору призначень f_i, аналогічним попередній задачі, отримаємо наступне оптимальне упорядкування (квадратами з хрестиками додатково позначені вихідні):

$$S_3 = \left\{ \begin{matrix} 1^1 & \boxtimes & 1^2 & 2 & \boxdot & \boxdot & \boxdot \\ 4, & 5, & \boxdot, & \boxtimes, & 6, & 3, & 7 \end{matrix} \right\}, l(S_3) = 7. \quad (1)$$

Бачимо, що в усіх випадках були отримані упорядкування різної довжини, а отже додаткові умови є суттєвими. Можна вважати, що задача з вихідними є деякою комбінацією *задач 1а та 4*. Варто відмітити також, що у випадку $W^1 = \{3\}$ на третьому місці обидві позиції виявилися б порожніми, проте, з огляду на те, що вихідні є вимушеними перервами у процесі виконання робіт, довжину упорядкування, як час необхідний для виконання всіх завдань, доцільно вважати рівною 7, що суперечить вихідному означенню довжини упорядкування. В той же час додавання вихідного після 7 місця не має впливати на довжину, оскільки всі роботи на той час вже будуть виконані. Тому необхідно ввести скореговане означення для довжини упорядкування для *задачі 5*.

Означення 3.3. Довжиною l упорядкування S для задачі з вихідними називається число непорожніх місць та порожніх місць з вихідними, праворуч від яких є непорожні місця.

У подальшому довжину упорядкування будемо розуміти у значенні, відповідному до задачі, що розглядається, без додаткового уточнення. Також для подальших досліджень можливих підходів, що можна застосувати до розв'язання наведених задач знадобиться наступне означення.

Означення 3.4. Директивними термінами d_i^s, d_i^e для i-ої роботи називаються числа, що визначають проміжок місць, на якому може стояти відповідна вершина в упорядкуванні, де d_i^s – найменший номер місця, на якому вона може стояти, а d_i^e – найбільший.

Відомо, що при $h = 2$ розв'язок класичної задачі може бути отриманий за поліноміальний час. Розглянемо, як алгоритм, заснований на максимальному паросполученні (див. підрозділ 1.4), може бути модифікований для розв'язання задачі із вихідними на прикладі графу з рис. 3.9 та додаткових умов з прикладу до *задачі 5*.

Схема модифікованого алгоритму

1. Для графу $G(V, U)$ будуємо неорієнтований граф $\overline{G}(V, E)$, де $(i, j) \in E$, якщо у графі G немає ані прямого, ані зворотного шляху між вершинами i та j (граф досяжності).

2. Видаляємо з графу $\overline{G}$ ребра, які з'єднують вершини, що відповідають роботам, які призначені різним робітникам.

3. Додаємо до графу $\overline{G}$ вершини, що відповідають вихідним, та з'єднуємо їх також з усіма вершинами, що відповідають роботам, які призначені іншим виконавцям, та з вихідним іншого виконавця, призначеним на те ж місце в упорядкуванні, якщо він існує. Для кожної вершини вводимо директивні терміни, що відповідають місцям, які вона повинна займати в упорядкуванні.

4. В отриманому графі $\overline{G}$ знаходимо максимальне паросполучення, яке позначимо $M \subseteq E$, тобто підмножину ребер E, що не мають спільних вершин, з максимальною потужністю.

5. В шуканому упорядкуванні S^* вважаємо всі місця вільними, покладаємо $k = 1$.

6. Якщо G порожній, то кінець алгоритму.

7. Якщо є вихідні, які відповідають k-ому місцю, то віддаємо їм перевагу (якщо вони поодинокі) або ребрам, до яких вони відносяться.

8. Можливий один з наступних випадків:

а) Серед вільних вершин графу G існує одна чи декілька таких, що не належать жодному з ребер у M, тоді обираємо для розташування їх.

б) У множині M існує таке ребро (i, j), що вершини i та j є відкритими, тоді обираємо їх для розташування і видаляємо (i, j) з M.

в) У множині M знайдеться така пара ребер (i, p) та (j, q), що вершини i та j є відкритими та відповідають роботам, призначеним різним виконавцям, тоді для розташування обираємо вершини i та j, ребра (i, p) та (j, q) видаляємо з M, а ребро (p, q) додаємо у M, якщо воно є у графі $\overline{G}$.

г) Серед вільних вершин графу G існує одна вершина i, що не належить жодному з ребер у M, та деяка інша j, що належить деякому ребру (j, q) у M, та може бути виконана одночасно із i, тоді обираємо для розташування їх, а вершину q додаємо до поодиноких.

9. Об'єднуємо поодинокі вершини у пари, якщо між ними є ребра у графі досяжності, та додаємо отримані пари у M.

10. Обрані вершини розташовуємо на місце k в упорядкуванні S^* та видаляємо їх з графу G разом з вихідними дугами, якщо вони є. Приймаємо $G \coloneqq G, k \coloneqq k + 1$ та переходимо на 4.

Відмітимо також, що додатковою мотивацією для модифікації та застосування саме алгоритму, заснованого на максимальному паросполученні, є те, що для *задач 4 та 5* відповідний граф досяжності є дводольним. Це безпосередньо випливає з другого етапу алгоритму: залишаємо ребра лише між вершинам, що відповідають

роботам різних виконавців. Дводольність графу досяжності в свою чергу значно спрощує пошук максимального паросполучення.

Приклад 3.6 (розв'язання). На першому етапі алгоритму необхідно побудувати граф досяжності (неорієнтований граф, вершини якого з'єднані ребром, якщо у вихідному графі між ними немає орієнтованого шляху у будь-якому напрямку). Отримаємо граф $\overline{G_1}$ з рис. 3.10.

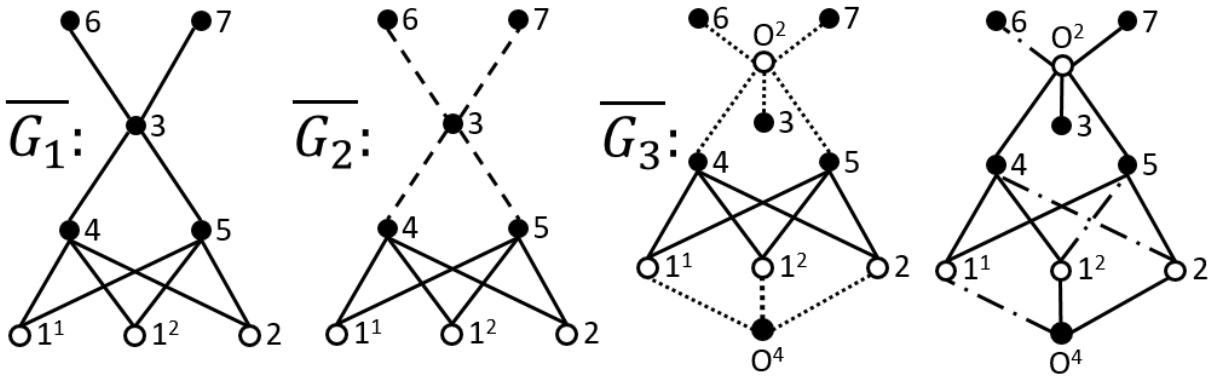

Рис. 3.10. Етапи перебудови графу досяжності з прикладу розв'язання задачі 5

Перед тим, як перейти до пошуку максимального паросполучення, згідно з алгоритмом, спочатку врахуємо той факт, що в шуканому упорядкуванні вершини, які відповідають роботам, що призначені різним робітникам, не можуть стояти на одному місці. Це можна зробити, видаливши з $\overline{G_1}$ ребра, що з'єднують вершини, які призначені одному виконавцю. Таким чином ці ребра не зможуть потрапити до максимального паросполучення, а отже в отриманому за алгоритмом упорядкуванні на кожному місці зможуть знаходитись лише вершини, що призначені різним виконавцям. Отримаємо граф $\overline{G_2}$, штрихові лінії позначають видалені ребра.

Врахуємо тепер наявність вихідних. Додамо до вихідного графу дві ізольовані вершини o^2 та o^4, які відповідають вихідним першого та другого виконавця відповідно. Для того, щоб врахувати, що вихідні призначені на конкретний день, визначимо для цих вершин директивні терміни рівні місцям в упорядкуванні, які вони мають займати. Вони мають бути додані до графу досяжності разом із ребрами, що з'єднують їх з усіма вершинами, призначеними для іншого виконавця. Отримаємо

граф $\overline{G_3}$, в якому додані ребра позначені пунктирними лініями. Наявність директивних термінів буде врахована пізніше.

Знайдемо тепер максимальне паросполучення в графі $\overline{G_3}$, ним буде $M = \{(1^1, o^4), (1^2, 5), (2,4), (o^2, 6)\}$ (на рис. 3.10 зображене штрихпунктирною лінією), при цьому вершини 3 та 7 не ввійшли до жодної з пар.

Перейдемо до побудови шуканого упорядкування.

Відкритими вершинами у вихідному графі є 1^1 та 4, вони належать до пар $(1^1, o^4)$ та $(2,4)$, тому, відповідно до алгоритму, розташовуємо їх в упорядкуванні на перше місце: $S[1] = \{1^1, 4\}$ та додаємо до паросполучення пару $(2, o^4) \to M$. Зазначимо, що вершини o^2 та o^4 також були відкритими, проте, відповідно, до директивних термінів вони не можуть бути розміщені на першому місці в шуканому упорядкуванні.

Відповідно до умов, у першого виконавця має бути вихідний на 2 місці, тому на цьому етапі маємо обов'язково розмістити вершину o^2, яка належить парі $(o^2, 6)$. Іншими відкритими вершинами є 1^2 та 5, які входять до однієї пари, та o^4, розміщення якої на 4 місці порушить директивні терміни. Аналогічно першому кроку об'єднаємо ці пари, тоді $S[2] = \{o^2, 5\}$. Проте пара $(1^2, 6)$ не може бути додана до M, оскільки між цими вершинами немає ребра у графі $\overline{G_3}$, тому додамо їх до вершин без пар.

На цьому етапі відкритими є лише вершини 1^2 та o^4, які не можна розмістити на 3 місці, тому можливим є лише розміщення $S[3] = 1^2$.

На 4 місці обов'язково має бути розміщена вершина o^4, вона утворює пару із вершиною 2, що також є відкритою, тому розташовуємо їх на це місце $S[4] = \{2, o^4\}$, а отже всі пари з множини M були використані.

Залишається розташувати вершини без пар, отримаємо: $S[5] = 6, S[6] = 3, S[7] = 7$. На цьому побудова упорядкування, відповідно до алгоритму, завершується.

Отже, упорядкування, отримане за модифікованим алгоритмом, збігається з оптимальним з прикладу для *задачі 5*.

Варто зробити наступні зауваження. По-перше, при об'єднанні пар важливо додатково слідкувати, аби на одне місце не потрапили вершини, що призначені одному виконавцю. По-друге, задачі із директивними термінами, на відміну *від задач 1, 1a, 4* та 5, взагалі кажучи, можуть не мати розв'язку, проте легко переконатися, що будь-який допустимий розв'язок *задачі 5* задовольняє введеним директивним термінам, згідно з означенням вихідного дня, а отже цей перехід є еквівалентним. По-третє, аналогічні модифікації алгоритму можуть бути застосовані до пошуку розв'язків для задачі зі змінною шириною при $h_i \in \{1,2\}$ та задачі з призначенням: для *задачі 1a* необхідно ввести ізольовані вершини з директивними термінами, які відповідають місцям з $h_i = 1$; для *задачі 4* достатньо лише видалити ребра у графі досяжності, що з'єднують вершини-роботи, що призначені одному виконавцю. По-четверте, запропонований алгоритм має принципові відмінності від алгоритму для класичної задачі, що не дозволяють стверджувати про оптимальність упорядкування, отриманого за модифікованим алгоритмом, у загальному випадку. Справді, бачимо, що оптимальний розв'язок (1) містить лише три заповнені місця, а отже він відповідає паросполученню з 3 парами вершин, тоді як максимальне паросполучення містить 4 пари. В зв'язку з цим при розміщенні вершин на 2 місце не змогли утворити нову пару та додали вершини до поодиноких, що також суперечить алгоритму.

З усього переліченого випливає необхідність обґрунтування оптимальності модифікованого алгоритму для *задачі 5*.

Розглянемо, що буде відбуватися із оптимальним упорядкуванням, якщо додамо до графу ізольовані вершини, що відповідають тим завданням, які можуть виконувати обидва виконавці. Слід відмітити, що поки в оптимальному упорядкуванні лишаються місця, на яких розташовано менше двох вершин, його довжина не буде змінюватися, оскільки додані вершини завжди можна розташувати на одну з вільних позицій. Отже, після приєднання деякої кількості таких вершин отримаємо упорядкування, в якому на всіх місцях розташовано по 2 вершини.

Зрозуміло, що кількість таких вершин, необхідна для отримання щільного упорядкування, заздалегідь не відома. Проте легко побачити, що парність кількості

необхідних вершин буде збігатися з парністю суми тривалостей виконання робіт та кількості вихідних. Окрім цього, якщо до графу із щільним упорядкуванням додати парну кількість ізольованих непризначених вершин, то оптимальне упорядкування для такого графу також буде щільним, оскільки надлишкові вершини можна розмістити разом.

Зроблені зауваження дозволяють довести твердження, аналогічне *твердженню 3*, а отже для доведення оптимальності модифікованого алгоритму достатньо показати, що він знайде щільне упорядкування, якщо воно існує, яке буде відповідати максимальному паросполученню.

Приклад 3.7. Нехай маємо граф G^2, зображений на рис. 3.11, $W^1 = \emptyset, W^2 = \{1\}, h = 2$.

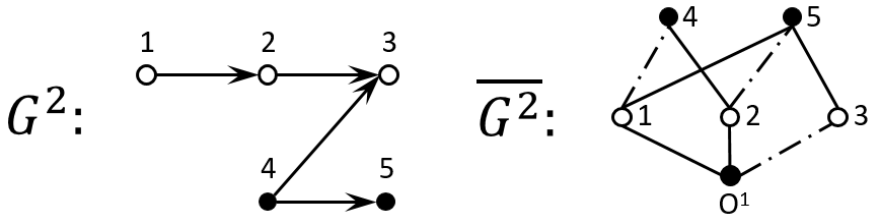

Рис. 3.11. Приклад застосування алгоритму для задачі з щільним упорядкуванням

Призначення робіт виконавцям, граф досяжності $\overline{G^2}$, отриманий за модифікованим алгоритмом, та максимальне паросполучення $M^2 = \{(1,4), (2,5), (3, o^1)\}$ зображені на рис. 3.11, згідно з попередньо використаними позначеннями.

Відповідно до алгоритму, на першому кроці маємо обов'язково розмістити вершину o^1. Вона належить до ребра $(3, o^1)$, в якому вершина 3 є закритою. Обидві відкриті вершини належать ребру $(1,4)$, тому розмітимо на перше місце в упорядкуванні пару $S[1] = \{1, o^1\}$, а вершини 3 та 4 вимушені додати до поодиноких, оскільки у $\overline{G^2}$ між ними немає ребра.

На другому кроці відкритими є вершини 2 та 4, перша з яких належить до ребра $(2,5)$, а друга – множині поодиноких. Для того, аби розташувати обидві вершини на друге місце, маємо розірвати пару: $S[2] = \{2,4\}$. Причому дві поодинокі вершини 3 та 5 у графі досяжності є суміжними, а отже можемо додати це ребро до M^2.

Таким чином, на останньому кроці розташовуємо вершини, що належать ребру $(3,5)$ на третє місце: $S[3] = \{3,5\}$. Вихідне упорядкування є щільним, а отже оптимальним:

$$S_4 = \left\{ \begin{matrix} 1 & 2 & 3 \\ \boxtimes, & 4, & 5 \end{matrix} \right\}, l(S_4) = 3.$$

Отримане упорядкування є оптимальним, але при цьому було порушено схему класичного алгоритму. Крім того слід відмітити, що не кожне початкове максимальне паросполучення гарантує нам отримання оптимального розв'язку без порушень класичного алгоритму, навіть у випадку щільного упорядкування.

Відзначимо також, що *задача 5* може бути зведена до *задачі 4* аналогічно тому, як *задача 1а* може бути зведена до *задачі 1*. Введемо фіктивного виконавця, призначені роботи якого утворюють ланцюжок, довжина якого дорівнює довжині оптимального упорядкування. Зрозуміло, що оптимальне упорядкування для отриманої задачі при $h' = h + 1$ можна побудувати шляхом приєднання до кожного місця оптимального упорядкування вихідної задачі однієї вершини з доданого ланцюжка. А отже кожна вершина-робота фіктивного виконавця буде однозначно відповідати деякому місцю в оптимальному упорядкуванні. З'єднаємо тепер кожну вершину, що відповідає вихідному, вхідною дугою з вершиною ланцюжка, що відповідає попередньому місцю в упорядкуванні до вихідного, та вихідною з вершиною, що відповідає наступному місцю в упорядкуванні після вихідного. Легко впевнитись, що розширене упорядкування після введення додаткових дуг також буде допустимим та оптимальним. При цьому вершини-вихідні тепер можуть бути розташовані лише на відповідних місцях, тому можемо виключити з задачі відповідні обмеження. Отримана задача є прикладом *задачі 4*, невідому довжину оптимального упорядкування можна знайти за допомогою бінарного пошуку, оскільки шукана довжина – мінімальна довжина ланцюжка, при якій на кожному місці в отриманому упорядкуванні буде знаходитись одна вершина з доданого ланцюжка.

Тому при подальшому розгляді увага приділялася лише *задачі 4*.

Насправді, можемо також взагалі не отримати оптимальне упорядкування через неоднозначність обрання пар вершин для розміщення при застосуванні алгоритму, навіть для найпростіших графів.

Приклад 3.8. Розглянемо *задачу 4* для графів з рис. 3.12 та $h = 2$, призначення робіт визначається помітками вершин на рисунку.

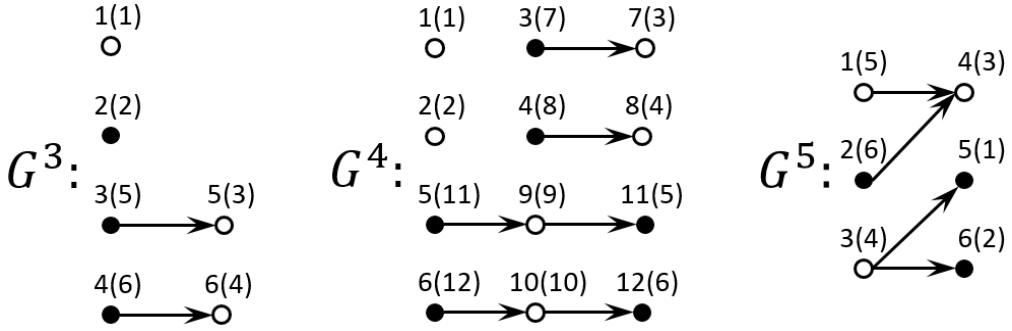

Рис. 3.12. Приклади неоптимальності алгоритму для задач з щільним упорядкуванням

Легко побачити, що допустимим максимальним паросполученням для графу G^3 буде $M^3 = \{(1,2), (6,3), (5,4)\}$. Відповідно до алгоритму можемо розмістити на перше місце пару вершин $(1,2)$, проте тоді нерозміщеними та відкритими залишаться лише вершини 3 та 4, які не можуть бути розміщені разом, оскільки відповідають роботам, що мають бути виконані одним і тим же виконавцем. Тому зможемо розмістити лише одну з цих вершин на другому місці, наприклад, вершину 3. На подальших кроках алгоритму зможемо розмістити пару вершин $(5,4)$ та вершину 6. У результаті отримаємо наступне упорядкування:

$$S_5 = \begin{Bmatrix} 1 & \boxdot & 5 & 6 \\ 2, & 3, & 4, & \boxdot \end{Bmatrix}, l(S_5) = 4.$$

Отримане упорядкування не є оптимальним. Справді, якби на першому етапі алгоритму замість розміщення пари вершин $(1,2)$ розмістили б пару $(1,3)$ та додали до M^3 нову пару $(6,2)$, то отримали б упорядкування меншої довжини:

$$S_5^* = \begin{Bmatrix} 1 & 5 & 6 \\ 3, & 4, & 2 \end{Bmatrix}, l(S_5^*) = 3.$$

Отже, бачимо, що не змогли отримати оптимальний розв’язок через невдале обрання пари вершин для розміщення. Насправді, в алгоритмі немає жодних вказівок

щодо пріоритетності розміщення вершин, відповідно до випадків а-г на 8 кроці. В свою чергу для класичного алгоритму доведено, що обрання вершин для розміщення не вплине на оптимальність отриманого розв'язку.

З аналізу прикладу 3.8 можна зробити припущення, що для отримання точного розв'язку потрібно забороняти розміщення ізольованих вершин парами, якщо у графі є інші неізольовані вершини. Таким чином матимемо вільні вершини для утворення пар на більш пізніх етапах побудови упорядкування. Виявляється, що ця умова не є достатньою для подолання проблеми неоднозначності вибору.

Перейдемо тепер до графу G^4. Максимальним паросполученням для нього буде наступне: $M^4 = \{(10,5), (9,6), (7,11), (8,12), (1,3), (2,4)\}$. Бачимо, що для цього графу ізольовані вершини належать до різних пар. При застосуванні алгоритму можемо розташувати спочатку пари $(1,3)$ та $(2,4)$. Далі маємо можливість лише комбінувати пари: розміщуємо пари $(7,5)$ і $(8,6)$ та додаємо до M^4 пари $(10,11)$ і $(9,12)$. Таким чином прийшли до підграфу та пар, що є аналогічними до попереднього прикладу. Результуюче упорядкування для цього прикладу:

$$S_6 = \begin{Bmatrix} 1 & 2 & 7 & 8 & 9 & 10 & \boxdot \\ 3' & 4' & 5' & 6' & \boxdot' & 11' & 12 \end{Bmatrix}, l(S_6) = 7.$$

У цьому прикладі також змогли б зменшити довжину при іншому порядку розташування пар. Розпочнемо з комбінування пар $(1,3)$ і $(10,5)$ та $(2,4)$ і $(9,6)$, тоді на перші два місця будуть розміщені пари $(1,5)$ і $(2,6)$, а пари $(10,3)$ і $(9,4)$ додані до M^4. Далі розміщуємо пари, що залишились у M^4, тоді отримаємо оптимальне упорядкування:

$$S_6^* = \begin{Bmatrix} 1 & 2 & 9 & 10 & 7 & 8 \\ 5' & 6' & 4' & 3' & 11' & 12 \end{Bmatrix}, l(S_6^*) = 6.$$

Аналізуючи даний приклад, можемо дещо уточнити попереднє припущення. Видно, що для отримання оптимального упорядкування мали спочатку розмістити вершини 5 та 6, що є початками найдовших ланцюжків у графі. В той же час при розміщенні на початковому етапі вершин 3 та 4, що також є початками ланцюжків, проте коротших, отримали неоптимальне упорядкування. Ці спостереження разом з

необхідністю збереження ізольованих вершин є основними положеннями рівневого принципу побудови упорядкувань [14] (див. пункт 1.4.3).

Відомим алгоритмом, який використовує рівневий принцип, є алгоритм, заснований на лексикографічному помічені вершин [87] (див. пункт 1.4.3).

Насправді, легко переконатися, що для попередніх двох графів за алгоритмом було б отримано оптимальне упорядкування (пріоритети вершин, отримані за алгоритмом, зазначені на рис. 3.12 у дужках). Проте виявляється, що можемо отримати неоптимальний розв'язок через довільність обрання міток для вершин без вихідних дуг.

Розглянемо граф G^5, нехай вершини без вихідних дуг мають наступні помітки: вершина 4 – помітку 3, вершина 5 – помітку 1, а вершина 6 – помітку 2. Відповідно до алгоритму, для вершини 1 послідовність поміток безпосередніх послідовників (3), для вершини 2 - (3), для вершини 3 - (2,1). Тоді, відповідно до лексикографічного упорядкування послідовностей, маємо (3) ≽ (3) ≽ (2,1), а отже вершина 3 отримає помітку 4, вершина 2 – помітку 5, а вершина 1 – помітку 6 (отримані помітки зображені на рис. 3.12 у дужках). При побудові упорядкування на перше місце розмістимо вершини 1 та 2, на друге місце можемо розмістити лише вершину 3, оскільки вершини 3 та 4 відповідають роботам, що мають бути виконані одним і тим самим виконавцем; після чого розміщуємо разом вершини 4 та 5 і на останньому місці буде вершина 6. Отримаємо наступне упорядкування:

$$S_7 = \begin{Bmatrix} 1 & 3 & 4 & \boxdot \\ 2, & \boxdot, & 5, & 6 \end{Bmatrix}, l(S_7) = 4.$$

Аналогічно попереднім прикладам, могли б отримати коротше упорядкування, якщо б розмістили спочатку вершини 2 та 3, потім змогли б розташувати пари вершин 1 та 5 і 4 та 6. При цьому результуючим упорядкуванням було б наступне:

$$S_7^* = \begin{Bmatrix} 3 & 5 & 6 \\ 2, & 1, & 4 \end{Bmatrix}, l(S_7^*) = 3.$$

Отже, бачимо, що обидва алгоритми є в загальному випадку лише наближеними. До переваг модифікованого алгоритму можна віднести легкість

впровадження обмежень, обумовлених структурою шуканого розв’язку, а до недоліків – значну невизначеність виконання кроків при побудові упорядкування. В той же час алгоритм, заснований на лексикографічному помічені, має недоліками переваги попереднього та навпаки.

В подальшому цікавим є дослідження питання можливостей поєднання переваг розглянутих алгоритмів в одному, чисельне визначення частоти знаходження точних розв’язків алгоритмами для різних класів графів, а також пошук та обґрунтування класів графів, для яких зазначені алгоритми є точними.

3.4 Висновки до розділу

В розділі наводяться нові результати для узагальнених задач оптимального упорядкування, що стосуються методів та алгоритмів їх розв’язання.

Доведено твердження, яке дозволяє звести задачу оптимального упорядкування зі змінною шириною упорядкування до класичної. Запропоновано підхід, що дозволяє використовувати оцінки знизу довжини для класичної задачі до цих задач у методі гілок та меж. До цього для таких задач був відомий лише аналог базової оцінки.

Також для цих задач були досліджені випадки порушення оптимальності алгоритму, заснованого на рівневому принципі, для вхідних бінарних дерев та на основі цього аналізу сформульовані бажані вимоги, яким має відповідати алгоритм. Зокрема була виділена вимога бути online-алгоритмом, тобто вибір вершин для розміщення не має залежати від місткостей наступних місць.

Запропоновано новий поліноміальний алгоритм, що задовольняє усім цим вимогам. Його теоретичний аналіз підтвердив, що він дозволяє знайти розв’язки для тих задач, де помиляється алгоритм, заснований на рівневому принципі.

При проведенні обчислювального експерименту встановлено, що запропонований алгоритм не є точним для цього класу, проте дозволяє розширити клас прикладів, для яких можемо знайти оптимальний розв’язок цієї задачі за поліноміальний час.

Продемонстровано, що побудова точного online-алгоритму для цього класу задач паралельного упорядкування в принципі неможлива.

Розглянуто новий клас узагальнених задач оптимального упорядкування, які враховують неповне завантаження виконавців. Продемонстрована істотність введених обмежень, порівняно з іншими відомими задачами паралельного упорядкування.

Для розв’язання цієї задачі запропоновано модифікацію алгоритму, заснованого на максимальному паросполученні, яка дозволила врахувати обмеження у структурі розв’язку. Показано, що цей алгоритм є лише наближеним.

По аналогії зі зведенням задачі зі змінною шириною упорядкування до класичної, обґрунтовано можливість зведення задач цього класу до задач упорядкування з призначенням, який також є мало вивченим.

Проведено порівняння запропонованої модифікації алгоритму, заснованого на максимальному паросполученні, та алгоритму, заснованого на лексикографічному помічені, при їх застосуванні до задач з призначенням. Виділені переваги та недоліки, які має кожен з них, в контексті задач з ускладненою структурою.

Основні результати розділу опубліковані у [2,9-11,13].

Розділ 4. ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1 Опис основних структурних компонентів програми

Для визначення порівняльної оцінки якості запропонованих у роботі алгоритмів та методів із відомими алгоритмами та методами розв'язання розглянутих задач було розроблено програмне забезпечення засобами об'єктно-орієнтованої мови програмування C#.

Основним структурним елементом програми є клас *Graph*, що призначений для роботи із графами у програмі. Відповідно до термінів об'єктно-орієнтованого програмування цей клас містить 5 полів, одну властивість та 23 методи, які будуть розглянуті нижче.

Визначальним фактором при роботі з графами є спосіб їх представлення у оперативній пам'яті комп'ютера. Найрозповсюдженішими представленнями є матриця суміжності, список дуг та представлення двома списками. Серед наведених представлень вирішено зупинитись на останньому через зручність у використанні та менші витрати пам'яті, порівняно з іншими представленнями. В той же час для введення графів з файлу обрано представлення у вигляді списку дуг, оскільки воно є зручнішим для використання користувачем, адже не потребує додаткової роботи, пов'язаної з правильним заповненням списків.

Розглянемо варіант представлення двома списками, яке реалізує клас *Graph*, детальніше. Нехай n – кількість вершин у графі, а m – кількість дуг у графі, тоді побудуємо списки $(a_1, \dots, a_n | a_{n+1} = m + 1)$ та $(b_1, \dots, b_m)$ такі, що вершини $b_{a_i}, \dots, b_{a_{i+1}-1}$ є суміжними до вершини $i, i = \overline{1, n}$. Тобто перший список для кожної вершини зберігає індекс місця у другому, з якого починаються номера вершин, суміжних даній. Додатково для пришвидшення пошуку відкритих вершин зберігається список, в якому для кожної вершини міститься кількість вхідних дуг. Наведені списки відповідають полям StartVertex, AdjacentVertices та InDegree у класі *Graph*. Інші поля використовуються для пошуку циклів у графі та будуть описані далі.

Властивість NVertices повертає кількість вершин у графі.

Усі методи класу *Graph* можна умовно поділити на наступні групи:

1. Конструктори, тобто методи, що використовуються для створення об’єктів класу.

2. Методи, які використовуються конструкторами.

3. Методи збереження графів та їх візуалізації.

4. Методи для спрощення реалізації алгоритмів та методів пошуку розв’язку задач паралельного упорядкування.

Опис класу містить 4 конструктори, вони відрізняються кількістю та типами аргументів, що використовуються для побудови об’єкту типу *Graph*:

1. Єдиним аргументом першого конструктора є шлях до файлу із графом. Метод зчитує файл, в якому граф подано у вигляді списку дуг, створює та заповнює потрібні поля-списки та викликає метод перевірки вхідного графу на ациклічність, який буде описаний у другій групі методів.

2. Аргументами другого є число вершин у графі, генератор випадкових чисел та булевий аргумент «Створити випадковий граф чи з вершинами на критичних шляхах?». Аргументи передаються у метод генерації випадкового графу, який виводить згенерований граф у файл (сам метод описаний також у другій групі), після чого викликається перший конструктор.

3. У якості аргументів третій конструктор приймає число вершин, ширину щільного упорядкування та генератор випадкових чисел. Алгоритм роботи цього конструктору збігається із попереднім, проте для побудови графу використовується метод генерації випадкового графу, для якого існує щільне упорядкування (описано нижче).

4. Аргументами четвертого є два описаних вище списки. Отриманий граф перевіряється на ациклічність, та заповнюється список кількості вхідних дуг.

До другої групи належать наступні 7 методів:

1. Метод CalculateInDegree для визначення кількості дуг, що входять у кожну вершину. Цей метод перебирає вершини у другому списку, збільшуючи лічильник вхідних дуг для відповідної вершини.

2. Метод AcycleCheck для перевірки вхідного графу на ациклічність. Він приймає булевий аргумент deleteCycles (чи треба видаляти цикли, якщо вони знайдені) та результатом його виклику є прапорець «чи є у графі цикл». Цей метод ініціалізує список-поле Visited, та для кожної, досі не переглянутої, вершини викликає модифікований метод пошуку у глибину, який повертає *true*, якщо цикл знайдено, інакше *false* (наступний метод).

3. Метод Dfs пошуку у глибину для виявлення циклів у графі та їх наступного видалення за необхідності. Аргументами є вершина, з якої потрібно розпочати пошук, та булева змінна «чи потрібно видаляти цикли». Повертається прапорець «чи було знайдено цикл». Ідея реалізації полягає в наступному. Вихідна вершина у полі Visited, яка спочатку мала білий колір (досі не розглядалася), позначається сірим кольором, тобто стає такою, що зараз знаходиться у процесі розгляду. Далі відбувається рекурсивний виклик методу для усіх, суміжних із даною, вершин. Після повернення з рекурсії колір вершини змінюється на чорний (вже розглянута). Якщо на якомусь етапі виконання суміжною вершиною до поточної є вершина сірого кольору, це означає, що знайдено цикл. В цьому випадку остання дуга у циклі записується у поле CycleEdge і метод повертає *true* (цикл знайдено). У випадку, коли потрібно знайти та видалити з графу усі цикли, то згадана дуга з графу видаляється, а поточна вершина у кінці позначається білим кольором.

4. Метод RandomGraph генерації випадкового графу. Його аргументами є кількість вершин та генератор випадкових чисел. У задачах паралельного упорядкування розглядаються графи, які є орієнтованими та ациклічними. Вершини таких графів можна розбити на рівні так, що всі дуги, які входять у вершини рівня, виходять з вершин з попередніх рівнях. Таким розбиттям на рівні є, наприклад, упорядкування $\underline{S}$. Ця ідея покладена у побудову випадкового графу: спочатку випадково визначається кількість рівнів у графі, потім вершини випадковим чином розподіляються між рівнями, та для кожної вершини обирається деяка кількість вершин з попередніх рівнів, з яких в неї будуть йти дуги.

5. Метод RandomCriticalGraph генерації випадкового графу, у якого усі вершини знаходяться на критичних шляхах. Він приймає ті ж аргументи, як і попередній, проте на відміну від нього для кожної вершини обираються вершини виключно з попереднього рівня. Додатково для кожної вершини, окрім вершин останнього рівня, також перевіряється чи мають вони вихідні дуги, і якщо це не так, то додаються дуги у випадково обрані вершини наступного рівня.

6. Метод RandGraph генерації випадкового графу або графу, в якого всі вершини знаходяться на критичних шляхах. Має такі ж аргументи, як і два попередні, та булевий аргумент «Створити випадковий граф з вершинами на критичних шляхах?». Застосовується для передання виклику з другого конструктору до одного з двох попередніх методів, відповідно до значення булевого аргументу.

7. Метод RandomFilledSequencingGraph генерації випадкового графу, для якого існує щільне упорядкування. Його аргументами є кількість вершин, ширина щільного упорядкування та генератор випадкових чисел. Метод базується на ідеї, що щільне упорядкування завжди є оптимальним. Він будує щільне упорядкування, після чого для кожної вершини обираються випадкові вершини з наступних місць упорядкування, з якими вона з’єднується дугами. За такої побудови початкове упорядкування буде допустимим і оптимальним, а отже шуканим. Для того, щоб послідовне розміщення вершин в порядку збільшення їх номерів не завжди було розв’язком задачі, вершини випадковим чином перенумеровуються.

Примітка. Графи, згенеровані методами 4, 5 та 7 є ациклічними, проте їх не доцільно будувати одразу у вигляді двох списків, оскільки це буде потребувати багато зайвих операцій при роботі з другим списком. Зручніше записати ребра у файл, а потім зчитати його за допомогою першого конструктору.

До третьої групи методів входять наступні:

1. Метод WriteToFile виведення графу у файл. Єдиним аргументом є шлях до вихідного файлу. Граф записується у цей файл: перший рядок файлу містить кількість вершин у графі, наступні рядки - список його ребер.

2. Метод DrawGraph візуалізації графу. Його аргументами є довжина та ширина результуючого зображення. У методі спочатку обирається масштаб, що відповідає заданим аргументам, після чого будується упорядкування $\underline{S}$. На наступному етапі зображуються вершини та їх підписи на відповідних рівнях. На останньому етапі різними випадковими кольорами відображаються дуги графу. Кольори обираються випадково, аби можна було чітко візуально визначити, де починаються і закінчуються дуги.

3. Метод TransitivityReduction транзитивного спрощення графу. Метод використовуються для покращення подання графів. Так при генерації графів в них зазвичай присутня велика кількість транзитивних дуг (дуги, видалення яких не призведе до того, що зникне шлях між вершинами, які вони поєднують). Такі дуги не впливають на розв'язок *задачі 1*, проте дуже загромаджують зображення графу, а отже при візуалізації їх краще видаляти. Для знаходження та видалення таких дуг була використана наступна модифікація алгоритму Флойда-Уоршала пошуку найкоротшого шляху між усіма парами вершин [115]. У ній мінімум було замінено на максимум, а нескінченості – на мінус нескінченності. Тоді, якщо у результуючій матриці суміжності після виконання алгоритму у комірці стоїть одиниця, то відповідна дуга не є транзитивною, інакше – транзитивною або неіснуючою. Наведений алгоритм є оптимальним з точки зору складності, оскільки потребує $O(n^3)$ операцій для транзитивних дуг будь-якого порядку, в той час як для транзитивних дуг тільки першого порядку оптимальний алгоритм має ту ж складність.

4. Метод RelabelToLevelPrinciple перенумерації вершин згідно з рівневим принципом. Відповідно до рівневого принципу, будується упорядкування $\overline{S}$ і вершини перенумеровуються так, аби номери вершин, що стоять у ньому раніше, були меншими за номери вершин, які знаходяться на наступних рівнях. Це забезпечує правильну нумерацію вершин.

Примітка. Хоча третій метод зменшує кількість дуг у графі, проте не використовується усталено перед застосуванням методу гілок та меж та інших алгоритмів, оскільки потребує великої кількості операцій, порівняно з цими

алгоритмами, і значно не впливає на їх роботу. Тому використовується лише для спрощення зображення графу, отримання точних упорядкувань алгоритмом, заснованим на лексикографічному помічені, при $h = 2$ та для експериментів з впливом транзитивних дуг на цей алгоритм.

Решта методів класу утворюють четверту групу:

1. Метод GetSources отримання списку вершин без вхідних дуг. Він проходить по списку із кількістю вхідних дуг вершин і повертає номера тих, які їх не мають, тобто на відповідному місці стоїть 0.

2. Метод GetSinks отримання списку вершин без вихідних дуг. Цей метод проходить по списку StartVertex і повертає номера вершин без вихідних дуг, тобто тих, для яких на відповідному та наступному за ним місцях стоять однакові значення.

3. Метод GetAdjacent отримання списку вершин, які є суміжними до даної вершини. Він повертає частину списку AdjacentVertices, яка відповідає аргументу – вхідній вершині.

4. Метод IsEmpty перевірки, що граф порожній. Він повертає *true*, якщо граф не містить вершин, інакше *false*.

5. Перевизначення оператору мінус (–) для видалення з графу деякої множини вершин. У якості аргументів він приймає граф та список вершин для видалення та повертає новий граф. Вершини видаляються одна за одною, у порядку спадання їх номерів, що дозволяє оминути проблему зміни індексації.

6. Статичний метод MyEstimation знаходження удосконаленої оцінки довжини, який застосовується для методу гілок та меж.

7. Статичний метод PermutateLabels для перенумерації вершин графу. Аргументами є граф, генератор випадкових чисел та словник із відповідністю між старими та новими номерами вершин, усталено порожній. У випадку, коли аргумент-словник порожній, генерується випадковий словник відповідності. Метод записує граф з новими помітками до файлу і повертає новий граф, побудований за допомогою першого конструктора.

8. Метод CountDfs визначення вершин, які є досяжними з даної. В якості аргументів приймає номер вихідної вершини та булевий масив відвіданих вершин. Метод реалізує рекурсивний пошук у глибину, який продовжується з вершин, суміжних до початкової, і помічає у масиві номера відвіданих вершин. В результаті отримуємо масив, в якому всі досяжні з даної вершини мають помітки *true*.

Додатково програма містить два методи, які не пов'язані із класом *Graph,* та також необхідні для спрощення роботи алгоритму методу гілок та меж.

1. Метод NextCombination отримання наступної комбінації. Аргументами є поточна комбінація та максимальне можливе значення її елементів. Метод повертає *true*, якщо йому вдалося побудувати наступну комбінацію, або *false*, якщо можливі комбінації вичерпані.

2. Метод WriteSequencing для виведення упорядкування у елемент керування ListBox. Як аргументи приймає саме упорядкування, його порядковий номер у дереві варіантів та отриману для нього оцінку. Формат подання інформації розглядається у керівництві користувача.

Центральним методом програми є алгоритм методу гілок та меж BranchNBounds для знаходження розв'язку *задачі 1*. Аргументами є вхідний граф, метод обчислення оцінки знизу довжини упорядкування, ширина упорядкування h та максимальна кількість розгалужень, необов'язковий аргумент, з усталеним значенням рівним 1000.

Для представлення вершин дерева варіантів у оперативній пам'яті використовується спеціальна структура даних OptionTreeNode. Вона містить інформацію про граф, що відповідає цій вершині, упорядкування, список досі неупорядкованих вершин та значення оцінки знизу для довжини упорядкування. Список неупорядкованих вершин використовується для збереження нумерації вершин в упорядкуванні, відповідно до вхідного графу, оскільки нумерація вершин у об'єкті класу *Graph* завжди починається з 0.

Метод реалізовано відповідно до алгоритму, наведеному у пункті 1.4.2. Вершини дерева варіантів зберігаються у списку з елементами типу OptionTreeNode,

рівень дерева визначається за кількістю вже заповнених місць в упорядкуванні. Спочатку список містить лише початкову вершину – корінь дерева, для якої граф співпадає із початковим, упорядкування є порожнім, усі вершини є нерозташованими, а значення оцінки розраховане для вхідного графу. Перебір виконується у циклі з передумовою, яка перевіряє, що кількість вершин у дереві не перевищує максимально дозволену. Кожна ітерація розпочинається з обирання найглибшої з вершин, які мають найкращі значення оцінки знизу довжини упорядкування. За допомогою метода IsEmpty перевіряється чи є граф, що відповідає цій вершині, порожнім, якщо так, то відбувається переривання циклу та повертається кількість вершин у дереві і довжина знайденого оптимального упорядкування. Якщо відповідний граф не порожній, то за допомогою методу GetSources визначається множина вершин без вхідних дуг у поточному графі. Якщо її потужність не перевищує ширину упорядкування h, то до списку додається одна нова вершина, а розташовані вершини видаляються з графу за допомогою перевизначеного оператору мінус (–). Якщо ж потужність множини перевищує h, то перебираються усі комбінації цих вершин з h елементів за допомогою методу NextCombination, відповідні вершини додаються до списку вершин дерева варіантів. Інформація про усі додані вершини виводиться у ListBox за допомогою методу WriteSequencing, після чого розглянута вершина видаляється зі списку. Якщо кількість доступних розгалужень вичерпана, то у елемент керування ListBox виводиться відповідне повідомлення, а метод повертає максимальну кількість вершин у дереві та найкращу з оцінок довжини у дереві розгалуження зі знаком мінус (щоб розуміти, що це значення не обов’язково співпадає із довжиною оптимального упорядкування).

Метод LexicographOrder реалізує алгоритм, заснований на лексикографічному помічені. У якості аргументів він приймає вхідний граф та місткості місць упорядкування h_i. Цей метод точно відтворює алгоритм, наведений у пункті 1.4.3, за винятком випадкових кроків, натомість перевага завжди віддається вершинам з меншими номерами. Алгоритм був реалізований в узагальненій формі, аби можна

було застосовувати його до *задачі 1а*. Для застосовування його до *задачі 1*, у якості другого аргументу передається список з усіма однаковими елементами, рівними h.

Метод MaxMatchingOrder, який є реалізацією алгоритму, заснованого на максимальному паросполученні. Він має такі ж аргументи, як і попередній, а також булевий аргумент «Чи варто розділяти пари?». Цей аргумент впливає на те, який алгоритм буде застосований до пошуку розв'язку: *false* – класичний алгоритм, в якому пари не розділяються (відтворює алгоритм з пункту 1.4.3, за винятком випадкових кроків, аналогічно, перевага віддається вершинам з меншими номерами), *true* – модифікований алгоритм з пункту 2.2.2. У методі застосовуються ряд допоміжних методів для виконання його кроків. Метод GetReachabilityGraph будує по заданому графу відповідний неорієнтований граф досяжності, що подається у пам'яті у вигляді списків суміжності; для визначення недосяжних вершин з даної використовується результати методу CountDfs. Для знаходження максимального паросполучення у отриманому графі досяжності використовується клас *Edmonds*, який реалізує алгоритм стиснення квіток Едмондса [90]. Він є адаптацією коду з [116] для мови програмування C#. Метод PlaceVerticesStep обирає пару вершин або одиночну вершину для розміщення, відповідно до класичного або модифікованого алгоритму, залежно від значення відповідного аргументу; якщо нічого розмістити не вдається, то повертається порожній результат. Метод PlaceVertices відповідає за підготовку необхідних даних для PlaceVerticesStep та реалізує узагальнення алгоритму на випадок ширини упорядкування більшої, аніж 2, шляхом багатократного виклику PlaceVerticesStep допоки не будуть вичерпані вільні позиції або метод не поверне порожній результат.

4.2 Інтерфейс та інструкція користувача

Після запуску програми користувач побачить вікно, зображене на рис. 4.1.

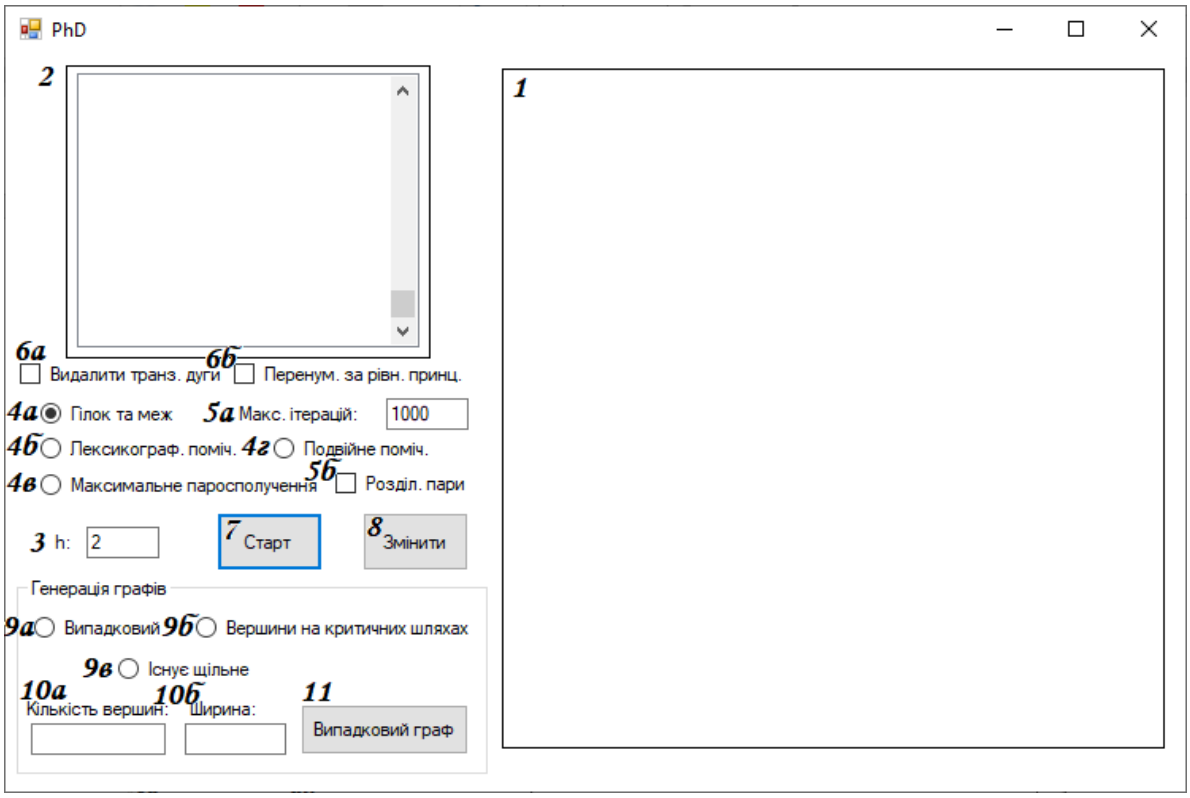

Рис. 4.1. Вікно програми до введення даних

Для введення даних про задачу, яка розв’язується, користувач має виконати наступні кроки:

1. Ввести значення ширини упорядкування h у елемент керування TextBox, який на рис. 4.1 позначений «3» (усталене значення дорівнює 2).

2. Обрати метод, який буде використаний для розв’язання задачі паралельного упорядкування. Для цього необхідно натиснути на відповідний елемент RadioButton: для застосування методу гілок та меж з удосконаленою оцінкою довжини – на елемент керування, який на рис. 4.1 позначений «4а»; для алгоритму, заснованого на лексикографічному помічені – позначений «4б»; для алгоритму, заснованого на максимальному паросполученні – «4в»; для алгоритму, заснованого на подвійному помічені – «4г».

3. У випадку обрання методу гілок та меж чи алгоритму, заснованого на максимальному паросполученні, можна також змінити значення їх параметрів. Для

методу гілок та меж у елементі TextBox, що на рис. 4.1 позначений «5а», можна вказати максимальну допустиму кількість вершин у дереві варіантів (усталене значення 1000). Для алгоритму, заснованого на максимальному паросполученні, елемент CheckBox, що на рис. 4.1 має позначку «5б», відповідає за те, чи дозволяється алгоритму розділяти пари: порожній квадрат – «ні» (класичний варіант), квадрат із галочкою – «так»; усталене значення – «ні».

4. Визначитися чи потрібно змінювати подання графу. Користувачу доступні дві модифікації подання графу. По-перше, з графу можуть бути видалені транзитивні дуги, за їх наявності; це визначається значенням у елементі CheckBox «6а» з рис. 4.1: порожній квадрат – «не видаляти», квадрат з галочкою – «видаляти»; усталеним значенням є «не видаляти». По-друге, нумерація вершин у графі може бути змінена на нумерацію, яка відповідає рівневому принципу; для цього призначений елемент CheckBox «6б» з рис. 4.1: порожній квадрат – «не перенумеровувати», із галочкою – «перенумерувати»; стандартне значення – «не перенумеровувати». Відмітимо, що всі зміни вносяться лише до представлення графу у оперативній пам’яті комп’ютера, а збережене подання графу не змінюється.

На наступному етапі користувач має обрати граф, для якого він бажає застосувати обраний алгоритм. Користувач може вибрати один з двох передбачених у програмі варіантів задання графу: зчитування з файлу або генерація випадкового графу. Розглянемо їх детальніше.

а. У випадку, коли задача має бути розв’язана для якогось конкретного графу, то він має бути попередньо збережений користувачем у текстовому файлі із розширенням .txt. Перший рядок файлу має містити кількість вершин у графі, а наступні – дуги цього графу, по одній у кожному рядку, у форматі «$a\ b$» (без лапок), де a і b – номери вершин, які визначають початок та кінець дуги. Нумерація вершин починається з нуля. Файл не має містити сторонніх символів, зайвих пробілів, абзаців, тощо. Приклад правильно заповненого файлу можна побачити на рис. 4.2.

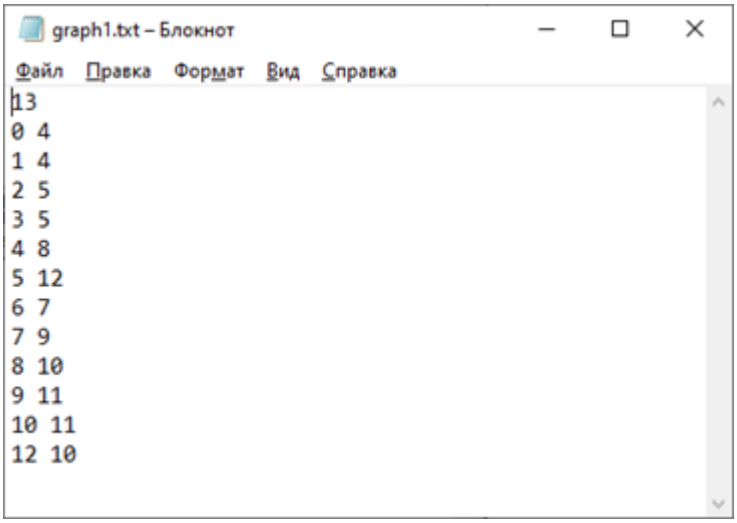

Рис. 4.2. Приклад правильного введення графу

b. Для випадкового графу, користувач спочатку має обрати тип генерованого графу. Це можна зробити, натиснувши на відповідний елемент RadioButton: елемент «9а» з рис. 4.1 для побудови випадкового графу без додаткових властивостей; елемент «9б» - для графу, в якого всі вершини лежать на критичних шляхах, елемент «9в» - для графу, для якого існує щільне упорядкування. Після чого потрібно визначитись із кількістю вершин у графі та ввести це число у елемент TextBox, який на рис. 4.1 позначено «10а». Якщо генерується граф, для якого існує щільне упорядкування, необхідно додатково вказати ширину щільного упорядкування у TextBox «10б» з рис. 4.1 (кількість вершин має бути кратною обраній ширині). Після чого, при натисканні на кнопку «Випадковий граф», яку позначено «11», у елементі керування PictureBox, який позначено «1», з’явиться згенерований граф (див. рис. 4.3). Якщо користувач бажає змінити параметри генерації, то він може обрати новий тип, обравши один з елементів «9а»-«9в», ввести нову кількість вершин у «10а», нову ширину у «10б» та знову натиснути на кнопку «11». Якщо ж користувача не влаштовує лише граф, то він може просто знову натиснути на «11» і отримати новий граф.

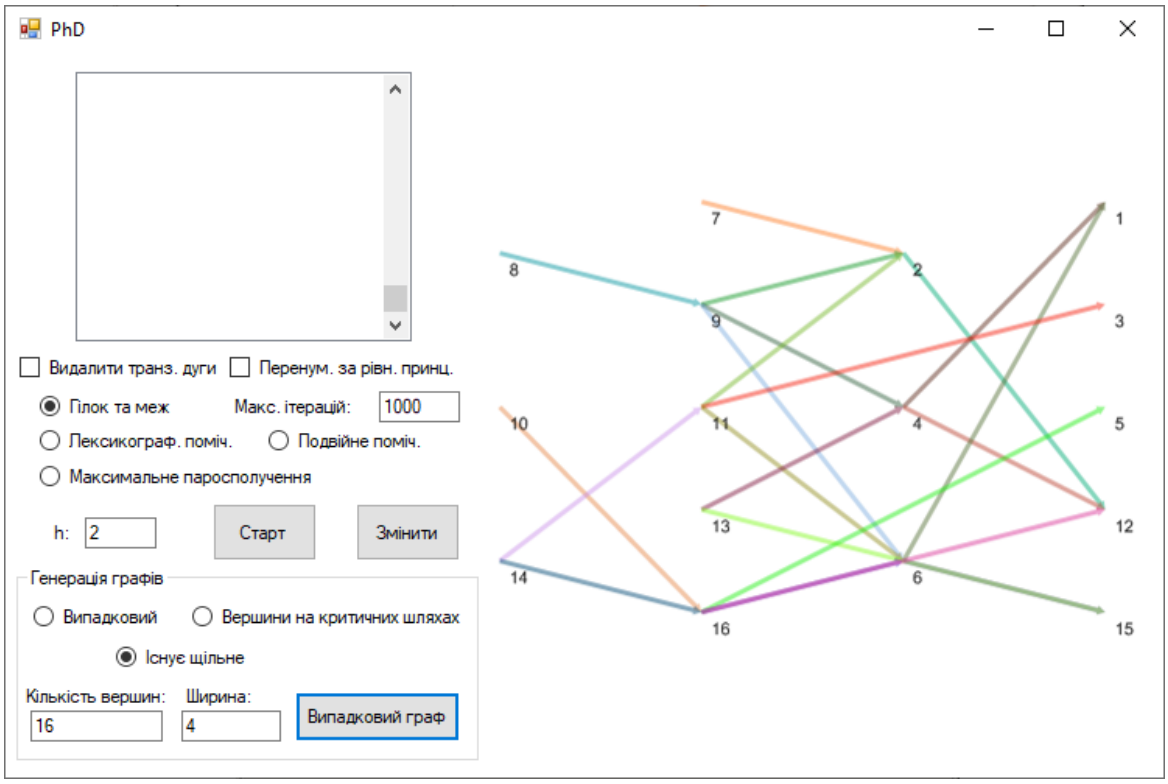

Рис. 4.3. Генерація випадкового графу

Після того, як усі необхідні дані було введено, можна переходити до запуску програми. Для цього потрібно натиснути на кнопку «Старт», яку на рис. 4.1 позначено «7». Якщо усі попередні кроки було зроблено вірно, то користувач побачить діалогове вікно вибору файлу, зображене на рис. 4.4.

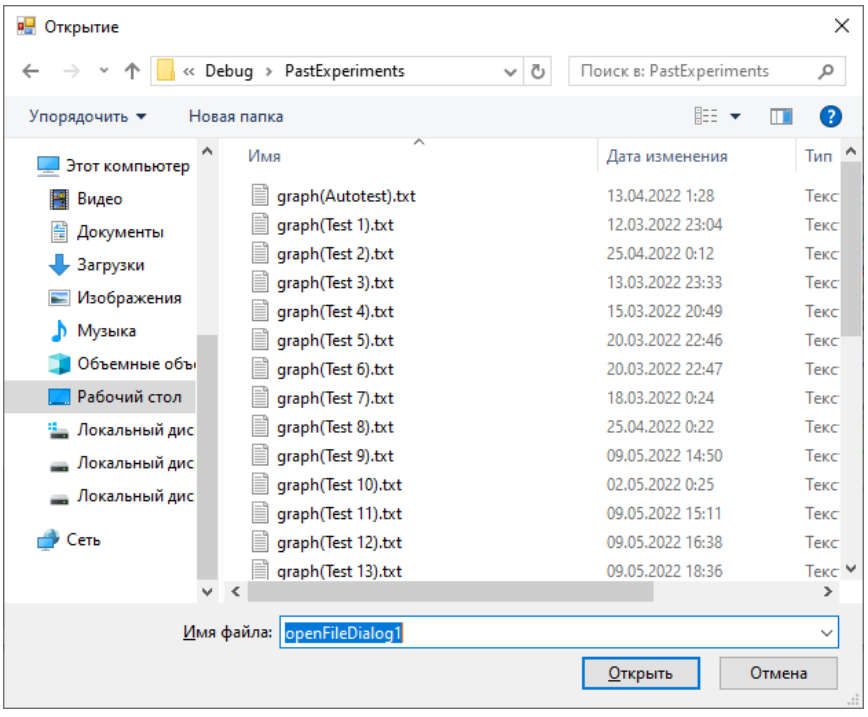

Рис. 4.4. Вікно вибору файлу із графом

У вікні усталено буде відображений вміст теки, у якій знаходиться виконуваний файл програми із розширенням .exe. Для використання згенерованого випадкового графу потрібно обрати автоматично створений файл “graph(Autotest).txt” (на рис. 4.4, він знаходиться у першому рядку вікна вибору). У випадку конкретного графу користувачу за допомогою цього вікна потрібно спочатку перейти до теки, де знаходиться збережений файл із графом, після чого обрати відповідний файл, натиснувши на кнопку «Відкрити» у діалоговому вікні.

Якщо обрано правильний файл, то програма почне пошук розв’язку за обраним алгоритмом із заданими графом та шириною упорядкування. У результаті у елементі ListBox («2» на рис. 4.1) з’явиться результат пошуку, а у елементі керування PictureBox («1» на рис. 4.1) – обраний граф, як це показано на рис. 4.5. Детально формат вмісту «2» буде описано нижче.

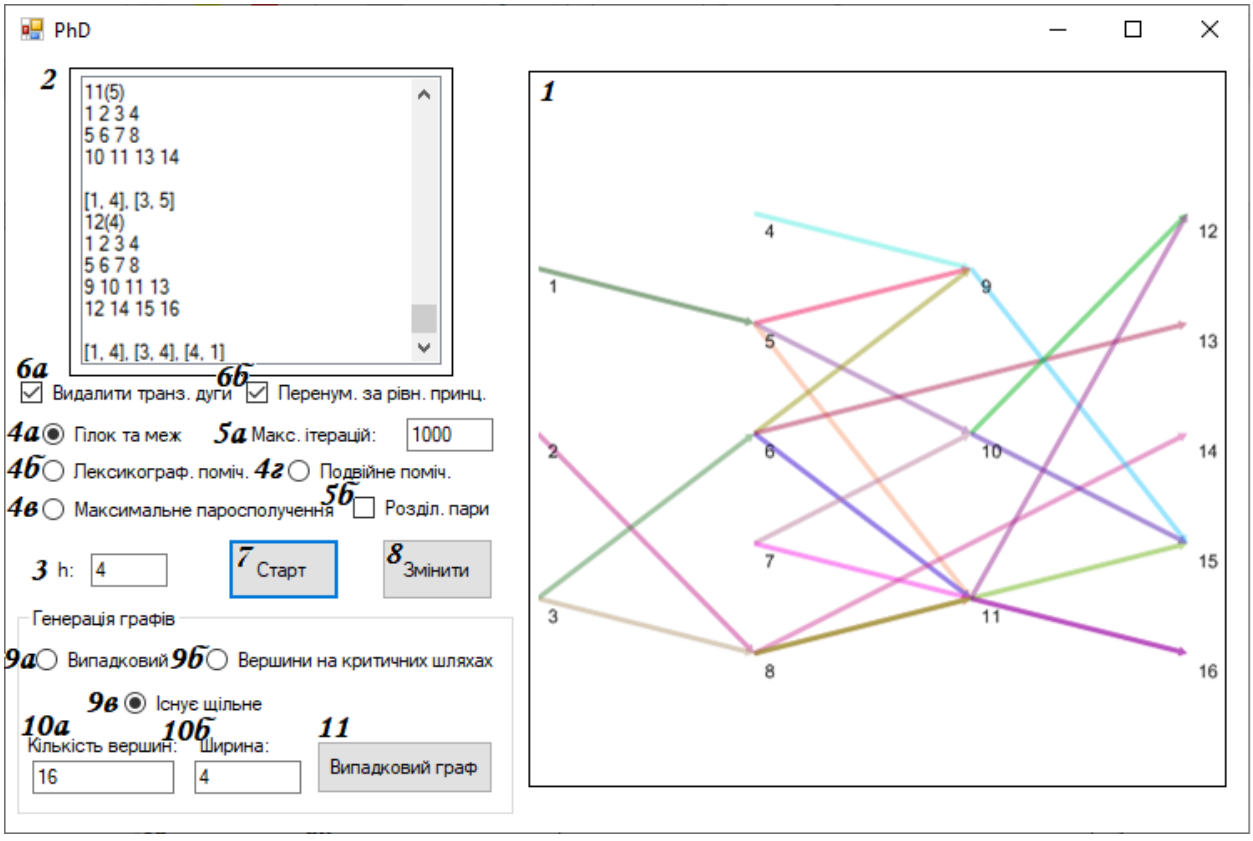

Рис. 4.5. Результат роботи програми

Якщо потрібно обрати інший алгоритм чи ширину упорядкування, то це можна зробити так само, як і на початковому етапі, увівши нові значення та натиснувши на

«7», при цьому знову обирати граф не потрібно. Оновлені результати можна побачити у елементах «1» та «2».

Для зміни випадкового графу, для якого програма шукала оптимальне упорядкування, у елементи «9а»-«9в» та «10а»,«10б» можна ввести нові дані, якщо це потрібно. Після чого користувач має натискати на «11», доки не визначиться із новим графом. В кінці необхідно натиснути на кнопку «7», щоб оновити результати (обирати файл знов не знадобиться).

Для зміни графу на власний, потрібно створити новий файл із графом, після чого натиснути на кнопку «Змінити», яка на рис. 4.1 позначена «8». Відкриється вікно з рис. 4.4, у якому буде відображено вміст теки, у якій знаходився попередній файл із графом. Аналогічно попередньому опису користувач має обрати новий файл та знову натиснути кнопку «7».

Результати виконання алгоритмів у елементі керування ListBox («2») подані у згрупованому вигляді. Кожна група відповідає одному проміжному упорядкуванню. Групи рядків розділені між собою порожнім рядком. Якщо це упорядкування є першим з отриманих при розгалуженні, то перший рядок містить інформацію про кількість досі нерозглянутих вершин на різних рівнях дерева варіантів у форматі «$[a_1, b_1], \dots, [a_k, b_k]$» (без лапок), де a_i – номер рівня, а b_i – кількість вершин на рівні a_i, інакше цей рядок відсутній, k – кількість непорожніх рівнів (додатковий порожній рядок відсутній). Далі подано інформацію про саме упорядкування. Спочатку його номер у дереві варіантів та отримане значення нижньої межі довжини упорядкування у дужках. У наступних рядках відображені місця в упорядкуванні, на яких вже розташовані вершини.

Останнім у списку упорядкувань буде оптимальне. Автоматично у «2» смуга прокрутки знаходиться у нижньому положенні, щоб відобразити оптимальне упорядкування. Якщо за вказану кількість вершин у дереві варіантів оптимальне упорядкування отримати не вдалося, то унизу ListBox («2») можна буде побачити відповідне повідомлення про перевищення кількості кроків алгоритму. У випадку,

коли вміст рядка не вміщається у елемент «2», користувач може побачити повне значення у відповідній підказці, натиснувши на цей рядок.

Якщо для пошуку розв’язку задачі паралельного упорядкування обрано інший алгоритм, у елементі керування ListBox («2») буде записане єдине упорядкування – упорядкування, отримане за цим алгоритмом, у тому ж форматі, що і при виборі методу гілок та меж.

ВИСНОВКИ

У дисертаційній роботі вирішено актуальне наукове завдання прикладної математики щодо розробки точних та наближених алгоритмів та методів розв’язання класичних та узагальнених задач упорядкування вершин орієнтованих ациклічних графів.

Основні наукові результати дисертації полягають у наступному:

1. Дістала подальшого розвитку теорія розв’язання задач дискретної оптимізації та її використання в задачах упорядкування, зокрема отримані результати, що дозволяють розглядати лише класичні задачі, в яких шукане упорядкування є щільним та ширина парною; розроблений підхід, що дозволяє перенести результати для класичних задач на задачі зі змінною шириною; отримані узагальнення відомих спеціальних упорядкувань $\underline{S}$ та $\overline{S}$, які враховують ширину упорядкування, та запропонована необхідна умова існування щільних упорядкувань.

2. Розроблено новий апарат для отримання точних розв’язків задач у загальних постановках, який включає новий підхід до скорочення перебору у методі гілок та меж за рахунок видалення гілок, які відповідають ізоморфним підграфам; покращенні оцінки знизу довжини упорядкування для класичної задачі та послідовність таких оцінок, кожна наступна з яких, теоретично, є точнішою за попередні.

3. Розроблені та обґрунтовані методи та алгоритми поліноміальної складності для отримання точних та наближених розв’язків, до яких відносяться використання алгоритму, заснованого на максимальному паросполученні, до задач із щільним упорядкуванням та парною шириною упорядкування, модифікації цього алгоритму; наближений підхід до скорочення перебору у методі гілок та меж на базі потужних інваріантів графів, алгоритм для пошуку щільних упорядкувань, заснований на методі гілок та меж з обмеженою глибиною пошуку, та наближений поліноміальний online-алгоритм розв’язання задачі зі змінною шириною.

4. Вперше розглянуто клас задач упорядкування з вихідними, який враховує неповне завантаження працівників. Проведено його порівняння з іншими відомими

класами, продемонстрована істотність введених обмежень та запропоновано наближений алгоритм їх розв’язання.

5. Засобами об’єктно-орієнтованої мови програмування C# розроблений програмний продукт з інтерфейсом користувача, що реалізує відомі та запропоновані алгоритми для проведення обчислювальних експериментів та розв’язання модельних прикладів.

6. На базі розробленого програмного продукту проведено обчислювальні експерименти для перевірки гіпотез та результатів для деяких отриманих наближених алгоритмів, зокрема для ітеративної модифікації алгоритму, заснованого на максимальному паросполученні, шляхом перевірки припущень та висування нових на основі аналізу отриманих результатів; для визначення ефективності наближених запропонованих алгоритмів для задачі з шуканим щільним упорядкуванням та задачі зі змінною шириною в порівнянні з іншими відомими поліноміальними алгоритмами, заснованими на рівневому принципі.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Караваєв К. Д. Про необхідні умови існування щільних упорядкувань в класичній задачі паралельного упорядкування. *Збірник наукових праць «Системні технології»*, м. Дніпро, 2024. Вип. 151. С. 76–91.

2. Караваєв К. Д., Турчина В. А. Узагальнення задач упорядкування з урахуванням неповного завантаження. *Збірник наукових праць «Питання прикладної математики і математичного моделювання»*, м. Дніпро, 2022. Вип. 22. С. 67–79.

3. Караваєв К. Д., Турчина В. А. Аналіз впливу автоморфізму графу на схеми направленого перебору. *Збірник наукових праць «Питання прикладної математики і математичного моделювання»*, м. Дніпро, 2021. Вип. 21. С. 94–104.

4. Turchyna V., Karavaiev K. Analysis of algorithms for constructing dense sequencing of digraphs vertices. *Proceedings of The Third International Workshop on Computer Modeling and Intelligent Systems (CMIS-2020),* Zaporizhzhia, 2020. P. 690–703.

5. Турчина В. А., Караваєв К. Д. Дослідження оцінок довжини паралельного упорядкування вершин графу. *Збірник наукових праць «Питання прикладної математики і математичного моделювання»*, м. Дніпро, 2018. Вип. 18. С. 186–195.

6. Турчина В. А., Караваєв К. Д. Контрприклад до алгоритму перерахування паралельно-послідовних графів без ізоморфізму. *Математичне та програмне забезпечення інтелектуальних систем (MSSIS-2023): Матеріали XXI міжнародної науково-практичної конференції, 22-24 листопада 2023 р.*, м. Дніпро, 2023. С. 291–292.

7. Караваєв К. Д., Турчина В. А. Про необхідні та достатні умови наявності автоморфізму у паралельно-послідовних графах. *Комбінаторні конфігурації та їхні застосування: Матеріали XXV Міжнародного науково-практичного семінару імені А. Я. Петренюка, (Запоріжжя – Кропивницький, 14-16 червня 2023 року)* / за ред. Г.П. Донця, м. Запоріжжя - Кропивницький, 2023. С. 122–129.

8. Караваєв К. Д. Використання інваріантів графів для скорочення напрямленого перебору у задачах упорядкування. *Математичне та програмне забезпечення*

інтелектуальних систем (MSSIS-2022): Матеріали XX ювілейної міжнародної науково-практичної конференції, 23-25 листопада 2022 р., м. Дніпро, 2022. С. 96–97.

9. Караваєв К. Д., Турчина В. А. Про зв'язок задач ізоморфізму графів та упорядкування їх вершин. *Комбінаторні конфігурації та їхні застосування: Матеріали XXIV Міжнародного науково-практичного семінару імені А. Я. Петренюка, (Запоріжжя – Кропивницький, 13-14 травня 2022 року)* / за ред. Г.П. Донця, м. Запоріжжя - Кропивницький, 2022. С. 30–33.

10. Караваєв К. Д., Турчина В. А. Деякі узагальнення задачі паралельного упорядкування. *Комбінаторні конфігурації та їхні застосування: Матеріали XXIII Міжнародного науково-практичного семінару імені А. Я. Петренюка, присвяченого 70-річчю Льотної академії Національного авіаційного університету, (Запоріжжя – Кропивницький, 13-15 травня 2021 року)* / за ред. Г.П. Донця, м. Запоріжжя - Кропивницький, 2021. С. 93–97.

11. Karavaiev K., Turchyna V., Hurko O. The problem of consistencing the architecture of computational systems and algorithms. *Сучасні науково-технічні дослідження у контексті мовного простору (іноземними мовами) 13 травня 2021 року: матеріали X Регіональної науково-практичної конференції молодих учених та студентів*, м. Дніпро, 2021. С. 142–145.

12. Турчина В. А., Караваєв К. Д. Дослідження оцінки точності алгоритму, заснованого на лексикографічному порядку. *Математичне та програмне забезпечення інтелектуальних систем (MSSIS-2019): Матеріали XVII міжнародної науково-практичної конференції, 20-22 листопада 2019 р.*, м. Дніпро, 2019. С. 258–259.

13. Турчина В. А., Караваєв К. Д. Застосування рівневого принципу до аналізу задач паралельного упорядкування та їх узагальнення. *Міжнародний науковий симпозіум «Інтелектуальні рішення». Обчислювальний інтелект (результати, проблеми, перспективи): праці міжнар. наук.-практ. конф., 15-20 квітня 2019 р.*, м. Ужгород, 2019. С. 56–57.

14. Hu T. C. Parallel Sequencing and Assembly Line Problems. *Operations Research*. 1961. Vol. 9, no. 6. P. 841–848.

15. Fox K. A. MRP-II providing a natural hub for computer-integrated manufacturing system. *Ind. Eng*. 1984. P. 44–50.

16. Jacobs F. R. OPT uncovered: Many production planning and scheduling concepts can be applied with or without the software. *Ind. Eng.* 1984. P. 32–41.

17. Jacobs F. R. The OPT scheduling system: A review of a new production scheduling system. *Product. Inventory Manage*. 1983. Vol. 24. P. 47–51.

18. Levulis R. J. Finite capacity scheduling and simulation systems. MTIAC. 1985. 33 p.

19. Schonberger R. J. Just-In-Time production systems: Replacing complexity with simplicity in manufacturing management. *Ind. Eng.* 1984. P. 52–63.

20. Rodammer F. A., White K. P. A recent survey of production scheduling. *IEEE Transactions on Systems, Man, and Cybernetics*. 1988. Vol. 18, no. 6. P. 841–851.

21. Rippin D. W. T. Batch process systems engineering: A retrospective and prospective review. *Computers & Chemical Engineering*. 1993. Vol. 17, no. 1. P. 1–13.

22. Wang K., Chau V., Li M. Scheduling fully parallel jobs. *J Sched*. 2018. Vol. 21. P. 619–631.

23. an Heuven van Staereling I. I. Scheduling Problems in the Service Industry: Theory and Applications: PhD-Thesis. Amsterdam, 2023. 163 p.

24. Taylor, K. E., Stouffer, R. J., Meehl, G. A. An Overview of CMIP5 and the Experiment Design. *Bulletin of the American Meteorological Society*. 2012. Vol. 93, no. 4. P. 485–498.

25. Duveau, P., Eggers, T., Hallard, R., Novelli, P. Aerodynamic Performance Analysis of the Hypersonic Airbreathing Vehicle JAPHAR. *Office national d'etudes et de recherches aerospatiales*. 1999. 11 p.

26. Deng M., Tian H., Fan B. Fine-granularity based application offloading policy in cloud-enhanced small cell networks. *Proc. IEEE Int. Conf. Commun. Workshops*. 2016. P. 638–643.

27. Guo S., Xiao B., Yang Y., Yang Y. Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing. *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.* 2016. P. 1–9.

28. Jia M., Cao J., Yang L. Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing. *Proc. IEEE Conf. Comput. Commun. Workshops*. 2014. P. 352–357.

29. Khalili S., Simeone O. Inter-layer per-mobile optimization of cloud mobile computing: A message-passing approach. *Trans. Emerg. Telecommun. Technol*. 2016. Vol. 27, no. 6. P. 814–827.

30. Liang J., Li K., Liu C., Li K. Joint offloading and scheduling decisions for DAG applications in mobile edge computing. *Neurocomputing*. 2021. Vol. 424. P. 160–171.

31. Lorenzo P. D., Barbarossa S., Sardellitti S. Joint optimization of radio resources and code partitioning in mobile edge computing. 2016. 13 p. (Preprint).

32. Mahmoodi S. E., Uma R. N., Subbalakshmi K. P. Optimal joint scheduling and cloud offloading for mobile applications. *IEEE Trans. Cloud Comput.* 2019. Vol. 7, no. 2. P. 301–313.

33. Li K. Scheduling Precedence Constrained Tasks for Mobile Applications in Fog Computing. *IEEE Transactions on Services Computing.* 2022. P. 1–14.

34. Muhuri P. K., Biswas S. K. Bayesian optimization algorithm for multi-objective scheduling of time and precedence constrained tasks in heterogeneous multiprocessor systems. *Applied Soft Computing*. 2020. Vol. 92. P. 106–274.

35. Yoo M. Real-time task scheduling by multiobjective genetic algorithm. *Journal of Systems and Software*. 2009. Vol. 82, no. 4. P. 619–628.

36. Zhuang X., Eichenberger A., Luo Ya., O'Brien K., O'Brien K. Exploiting Parallelism with Dependence-Aware Scheduling. *Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT.* 2009. P. 193–202.

37. Midkiff S. P. Automatic parallelization: An Overview of Fundamental Compiler Techniques. *Synthesis lectures on computer architecture. Springer Cham.* 2012. 157 p.

38. Vandierendonck H., Rul S., De Bosschere K. The Paralax infrastructure: automatic parallelization with a helping hand. *In Proceedings of the 19th international conference on Parallel architectures and compilation techniques (PACT '10).* 2010. P. 389–400.

39. Alur R., Stanford C., Watson C. A Robust Theory of Series Parallel Graphs. *Proceedings of the ACM on Programming Languages*. 2023. Vol. 7, POPL. P. 1058–1088.

40. Harary F. Graph theory. Reading, Mass : Addison-Wesley Pub. Co., 1969. 274 p.

41. Coffman, E. G., Bruno, J. Computer and job-shop scheduling theory. New York : Wiley, 1976. 299 p.

42. Ramamoorthy C. V., Gonzalez M. J. A survey of techniques for recognizing parallel processable streams in computer programs. *1969 Fall Joint Comput. Conf., AFIPS Conf Proc.* 1969. Vol. 35. P. 1–15.

43. Ramamoorthy C. V. A structural theory of machine diagnosis. *1967 Spring Joint Comput. Conf AFIPS Conf. Proc.* 1967. Vol. 30. P. 743–756.

44. Vásquez Ó. C. The Scheduling Zoo. URL: http://schedulingzoo.lip6.fr/ (date of access: 01.03.2024).

45. Rayward-Smith V. J. UET scheduling with unit interprocessor communication delays. *Discrete Applied Mathematics*. 1987. Vol. 18, no. 1. P. 55–71.

46. Lenstra J. K., Veldhorst M., Veltman B. The complexity of scheduling trees with communication delays. *Journal of Algorithms*. 1996. Vol. 20, no. 1. P. 157–173.

47. Varvarigou T., Kailath T., Roychowdhury V., Lawler E. Scheduling in and out forests in the presence of communication delays. *IEEE Transactions on Parallel & Distributed Systems*. 1996. Vol. 7, no. 10. P. 1065–1074.

48. Picouleau C. Ordonnancement de taches de duree unitaire avec des delais de communication unitaires sur m processeurs. *Inst. Blaise Pascal, Univ*. 1991.

49. Lenstra J. K., Rinnooy Kan A. H. G. Complexity of scheduling under precedence constraints. *Operations Research*. 1978. Vol. 26, no. 1. P. 22–35.

50. Hoogeveen J. A., Lenstra J. K., Veltman B. Three, four, five, six, or the complexity of scheduling with communication delays. *Operations Research Letters*. 1994. Vol. 16, no. 3. P. 129–137.

51. Garey M., Johnson D., Tarjan R., Yannakakis, M. Scheduling Opposing Forests. *SIAM Journal on Algebraic Discrete Methods*. 1983. Vol. 4. P. 72–93.

52. Dolev D. Warmuth M. K. Profile Scheduling of Opposing Forests and Level Orders. *SIAM Journal on Algebraic Discrete Methods*. 1985. Vol. 6, no. 4. P. 665–687.

53. Dolev D., Warmuth M. K. Scheduling precedence graphs of bounded height. *Journal of Algorithms*. 1984. Vol. 5. P. 48–59.

54. Ullman J. D. NP-complete scheduling problems. *J. of Comput. And Syst. Sci*. 1975. P. 384–393.

55. Турчина В. А., Коваленко Є. О. Дослідження задачі упорядкування з перериваннями для одного підкласу дерев. *Збірник наукових праць «Питання прикладної математики і математичного моделювання»*, м. Дніпро, 2023. Вип. 23. С. 118–125.

56. Турчина В. А., Коваленко Є. О. Вплив початкових даних задачі паралельного упорядкування з перериваннями на оптимальність розв'язку. *Збірник наукових праць «Питання прикладної математики і математичного моделювання»*, м. Дніпро, 2022. Вип. 22. С. 158–167.

57. Коваленко Є. О., Турчина В. А. Аналіз впливу структури графів на оптимальність розв'язку задач паралельного упорядкування з перериваннями. *Збірник наукових праць «Питання прикладної математики і математичного моделювання»*, м. Дніпро, 2021. Вип. 21. С. 130–137.

58. Garey M. R., Johnson D. S. Scheduling tasks with non-uniform deadlines on two processors. *Journal of the ACM*. 1976. Vol. 23, no. 3. P. 461–467.

59. Garey M. R., Johnson D. S. Two-Processor Scheduling with Start-Times and Deadlines. *SIAM J. Comput*. 1977. Vol. 6, no. 3. P. 416–426.

60. Sergienko I. V., Shylo V. P., Roshchyn V. O. Algorithm Unions for Solving Discrete Optimization Problems. *Cybernetics and Systems Analysis.* 2023. Vol. 59, no. 5. P. 753–762.

61. Yakovlev S., Kartashov O., Pichugina O. Optimization on Combinatorial Configurations Using Genetic Algorithms. *Proceedings of The Second International Workshop on Computer Modeling and Intelligent Systems (CMIS-2019),* Zaporizhzhia, 2019. P. 28–40.

62. Hulianytskyi L., Riasna I. Formalization and classification of combinatorial optimization problems. *Optimization Methods and Applications: In Honor of Ivan V. Sergienko's 80th Birthday*. 2017. P. 239–250.

63. Gulyanitskii L. F., Sergienko I. V., Khodzinskii A. N. Discrete optimization methods for multiprocessor computer systems. *Cybernetics.* 1988. Vol. 24, no. 4. P. 418–427.

64. Донець Г. П. Графовий підхід до розв’язання задач комбінаторного розпізнавання. Кібернетика і системний аналіз. 2017. Вип. 53, №. 6. С. 44–53.

65. Romanova T. E., Stetsyuk P. I., Chugay A. M., Shekhovtsov S. B. Parallel computing technologies for solving optimization problems of geometric design. *Cybernetics and Systems Analysis.* 2019. Vol. 55. P. 894–904.

66. Чуб І. А., Новожилова М. В., Беленченко І. В. Математична модель та розв’язок оптимізаційної задачі розподілу ресурсів проекту. *Системи обробки інформації*. 2011. Вип. 2. С. 291–294.

67. Леонова М. В., Ємець О. О. Задача розкладу для одного приладу з функцією мінімізації часу обслуговування всіх завдань. *Інформатика та системні науки (ІСН-2015)*, м. Полтава, 2015. 8 с.

68. Koliechkina L., Pichugina O., Yakovlev S. A graph-theoretic approach to multiobjective permutation-based optimization. *International Conference on Optimization and Applications*. Cham: Springer International Publishing. 2019. P. 383–400.

69. Semeniuta M. F., Donets G. A. Group labeling of some graphs. *Cybernetics and Systems Analysis*. 2020. Vol. 56, no. 5. P. 701–709.

70. Semeniuta M. F. Combinatorial Configurations in the Definition of Antimagic Labelings of Graphs. *Cybernetics and Systems Analysis*. 2021. Vol. 57, no. 2. P. 30–40.

71. Kozin I. V., Maksyshko N. K., Perepelitsa V. A. A Fragmented Model for the Problem of Land Use on Hypergraphs. *Cybernetics and Systems Analysis*. 2020. Vol. 56, no. 5. P.753–757.

72. Козін І., Максишко Н., Терешко Я. Метод імітації відпалу для задачі рівноважного розміщення. *Фізико-математичне моделювання та інформаційні технології*. Львів, 2021. Вип. 32. С.152–158.

73. Kozin I. V., Narzullaev U. Kh., Sardak O. V., Sabirov Z. R. Simulation of Annealing Algorithm for the Flat Rectangular Cutting Problem. *International Journal of Theoretical and Applied Issues of Digital Technologies*. 2023. Vol. 1, no. 3. P. 16–24.

74. Гук Н., Диханов С., Долотов І. Аналіз структури сайту з використанням поняття модулярності. *Математичне та комп'ютерне моделювання. Серія: Фізико-математичні науки*. 2020. С. 99–114.

75. Guk N., Verba O., Yevlakov V. Design of a Recommendation System Based on the Transition Graph. *Eastern-European Journal of Enterprise Technologies*. 2021. Vol. 3, no. 4. P. 24–31.

76. Bulat A. F., Kiseleva E. M., Hart L. L., Prytomanova O. M. Generalized models of logistics problems and approaches to their solution based on the synthesis of the theory of optimal partitioning and neuro-fuzzy technologies. In: Zgurovsky M., Pankratova N. (eds) *Studies in Computational Intelligence*. Springer: Cham, 2023. Vol. 1107, Chapter 21. P. 355–376.

77. Kiseleva E. M., Prytomanova O. M., Hart L. L. Application of optimal set partitioning theory to solving problems of artificial intelligence and pattern recognition. *System Research & Information Technologies*. 2021. Vol. 4. P. 91–101.

78. Турчина В. А., Федоренко Н. К. Наближені алгоритми побудови оптимальних паралельних упорядкувань заданої довжини. *Питання прикладної математики і математичного моделювання: зб. наук. праць.* Дніпропетровськ: ДНУ, 2010. С. 312–319.

79. Турчина В. А., Федоренко Н. К. Алгоритми побудов всіх паралельних упорядкувань заданої довжини. *Питання прикладної математики і математичного моделювання: зб. наук. праць.* Дніпропетровськ: ДНУ, 2011. С. 263–268

80. Турчина В. А., Федоренко Н. К. Узагальнення задачі паралельного упорядкування на випадок двох типів вершин. *Питання прикладної математики і математичного моделювання: зб. наук. праць.* Дніпропетровськ: ДНУ, 2007. С. 303–316.

81. Турчина В. А., Федоренко Н. К. Алгоритми розв'язання узагальненої задачі упорядкування. *Питання прикладної математики і математичного моделювання: зб. наук. праць.* Дніпропетровськ: ДНУ, 2006. С. 191–199.

82. Турчина В. А., Федоренко Н. К. Дослідження узагальнених моделей в задачах упорядкування. *Збірник праць 6-ї міжнародної конференції «Сучасні проблеми гуманізації та гармонізації управління»*, м. Харків, 2005. С. 190–192.

83. Little J. D. C., Murty K. G., Sweeney D. W., Karel, C. An Algorithm for the Traveling Salesman Problem. *Operations Research*. 1963. Vol. 11, no. 6. P. 972–989.

84. Fernandez E. B., Buspell B. Bounds on the number of processors and time for multiprocessor optimal schedules. *IEEE Transactions on Computers*. 1973. Vol. C-22, no. 8. P. 745–751.

85. Graham R. L. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math*. 1969. Vol. 17, no. 2. P. 416–429.

86. Ramamoorthy C. V., Chandy K. M., Gonzalez M. J. Optimal scheduling strategies in a multiprocessor system. *IEEE Trans. Comput.* 1972. Vol. C-21. P. 137–146.

87. Coffman E. G., Graham R. L. Optimal scheduling for two-processor systems. *Acta Informatica*. 1972. Vol. 1. P. 200–213.

88. Chardon M., Moukrim A. The Coffman-Graham Algorithm Optimally Solves UET Task Systems with Overinterval Orders. *SIAM Journal on Discrete Mathematics*. 2005. Vol. 19, no. 1. P. 109–121.

89. Fujii M., Kasami T., Ninomija K. Optimal sequencing of two equivalent processors. *SIAM J. Appl. Math*. 1969. Vol. 17, no. 4. P. 784–789.

90. Edmonds J. Paths, trees, and flowers. *Can. J. Math*. 1965. Vol. 17. P. 449–467.

91. Gabow H. N. An almost linear algorithm for two processor scheduling. *Journal of the ACM*. 1982. Vol. 29, no. 3. P. 766–780.

92. Gandal D., Ranade A. G. Precedence constrained scheduling in (2 – 7 / (3p+1)) optimal. *Journal of Computer and System Sciences*. 2008. Vol. 74, no. 7. P. 1139–1146.

93. Levey E., Rothvoss T. A (1+epsilon)-Approximation for Makespan Scheduling with Precedence Constraints Using LP Hierarchies. *SIAM Journal on Computing*. 2019. P. 201–217.

94. Garg Sh. Quasi-PTAS for scheduling with precedences using LP hierarchies. *In 45th International Colloquium on Automata, Languages, and Programming, ICALP*. 2018. 18 p. (Preprint).

95. Li S. Towards PTAS for Precedence Constrained Scheduling via Combinatorial Algorithms. *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2021. P. 2991–3010.

96. Nederlof J., Swennenhuis C. M. F., Węgrzycki K. A Subexponential Time Algorithm for Makespan Scheduling of Unit Jobs with Precedence Constraints. 2023. 25 p. (Preprint).

97. Pinedo M. L. Scheduling: Theory, Algorithms, and Systems. 6th ed. Springer Cham, 2022. 698 p.

98. Luce R., Perry A. A method of matrix analysis of group structure. *Psychometrika*. 1949. Vol. 14. P. 95–116.

99. McKay B. D. Practical Graph Isomorphism. *Congressus Numerantium*. 1981. P. 45–87.

100. Junttila T., Kaski P. Engineering an efficient canonical labeling tool for large and sparse graphs. *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments*. 2007. P. 135–149.

101. Darga P., Sakallah K., Markov I. L. Faster Symmetry Discovery using Sparsity of Symmetries. *Proceedings of the 45th Design Automation Conference*. 2008. P. 149–154.

102. Katebi H., Sakallah K., Markov I. L. Symmetry and Satisfiability: An Update. *Proc. Satisfiability Symposium*. 2010. P. 113–127.

103. Matsumoto Y., Moriyama S., Imai H., Bremner D. Matroid enumeration for incidence geometry. *Discrete Comput. Geom*. 2012. P. 17–43.

104. McKay B. D., Royle G. F. Constructing the cubic graphs on up to 20 vertices. *Ars Combinatoria*. 1986. P. 129–140.

105. Mayhew D., Royle G. F. Matroids with nine elements. *J. Comb. Theory*. 2008. P. 415–431.

106. Grüner T., Laue R., Meringer M. Algorithms for Group Actions: Homomorphism Principle and Orderly Generation Applied to Graphs. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. 1997. P. 113–122.

107. McKay B. D. Isomorph-Free Exhaustive Generation. *Journal of Algorithms*. 1998. P. 306–324.

108. Babai L., Erdős P., Selkow S. M. Random graph isomorphism. *SIAM J. Comput*. 1980. P. 628–635.

109. Arvind V., Köbler J., Rattan G., et al. Graph Isomorphism, Color Refinement, and Compactness. *Comput. complex*. 2017. Vol. 26, no. 3. P. 627–685.

110. Valdes J., Tarjan R. E., Lawler E. L. The recognition of series parallel digraphs. *SIAM J. Comput*. 1982. P. 298–313.

111. Aho A. V., Hopcroft J. E., Ullman J. D. The Design and Analysis of Computer Algorithms. *Reading, Mass.: Addison-Wesley*. 1974. 470 p.

112. Hong S.-H., Eades P., Lee S.-H. Drawing series parallel digraphs symmetrically. *Computational geometry*. 2000. Vol. 17, no. 3-4. P. 165–188.

113. Bentley J. Programming pearls. *Communications of the ACM*. 1984. Vol. 27, no. 9. P. 865–873.

114. Hoogeveen J. A., van de Velde S. L., Veltman B. Complexity of scheduling multiprocessor tasks with prespecified processor allocations. *Discrete Applied Mathematics*. 1994. Vol. 55, no. 3. P. 259–272.

115. Floyd R. W. Algorithm 97: Shortest path. 1962. *Commun. ACM.* Vol. 5, no. 6. P. 345.

116. Shoemaker A., Vare S. Edmonds' Blossom Algorithm. *Technical Report CME 323*. Standord University: Stanford, CA, USA, 2016. P. 1–27.

ДОДАТОК А. АКТ ВПРОВАДЖЕННЯ РЕЗУЛЬТАТІВ РОБОТИ

ПОГОДЖЕНО
Проректор з наукової роботи Дніпровського національного університету імені Олеся Гончара
_____ Олег МАРЕНКОВ
«07» березня 2024 р.

ЗАТВЕРДЖЕНО
В.о, першого проректора Дніпровського національного університету імені Олеся Гончара
_____ Валентина СІЛІЧ-БАЛГАБАЄВА
«07» березня 2024 р.

АКТ

впровадження результатів роботи, поданої на здобуття наукового ступеня доктора філософії **Караваєва Костянтина Дмитровича** на тему **«Методи і алгоритми розв'язання класичних та узагальнених задач упорядкування вершин орграфів»**, в освітній процес Дніпровського національного університету імені Олеся Гончара

1. Вчена рада факультету прикладної математики у складі 17 осіб заслухала повідомлення аспіранта кафедри обчислювальної математики та математичної кібернетики Караваєва Костянтина Дмитровича про результати наукового дослідження та їх використання в освітньому процесі кафедри обчислювальної математики та математичної кібернетики.

2. Стисла характеристика дослідження:

Математична теорія паралельного упорядкування є одним з потужніх інструментів розв'язання багатьох теоретичних та практичних задач, що можуть бути зведені до оптимізаційних задач на графах. Робота Караваєва К.Д. присвячена подальшій розробці теоретичного апарату, що лежить в основі методів і точних та наближених алгоритмів розв'язання задач паралельного упорядкування і складається із вступу, основної частини та висновків.

Для класичної задачі була покращена нижня оцінка довжини упорядкування, побудована нова схема методу гілок та меж, яка враховує обмеження, що накладає на упорядкування його щільність та модифіковано класичний алгоритм для побудови щільних упорядкувань.

Для задачі зі змінним значенням ширини упорядкування було теоретично обґрунтовано можливість її зведення до класичної задачі та запропоновано наближений on-line алгоритм до розв'язання цієї задачі для бінарних дерев. Вперше розглянуто новий клас узагальнених задач оптимального упорядкування з неповним завантаженням, показана можливість зведення до інших відомих класів задач паралельного упорядкування та розроблено наближений алгоритм для розв'язання задач з цього класу.

Під час досліджень також запропоновано новий підхід для скорочення перебору у методі гілок та меж, пов'язаний із виключенням гілок, що відповідають ізоморфним підграфам, були розглянуті точні та наближені варіанти його реалізації.

3. Використання в освітньому процесі:

Результати дисертаційних досліджень впроваджено в освітній процес кафедри обчислювальної математики та математичної кібернетики факультету прикладної математики ДНУ під час викладання дисципліни «Методи і алгоритми розв'язання задач дискретної оптимізації» для здобувачів вищої освіти рівня PhD спеціальності 113 Прикладна математика ОНП «Прикладна математика» та дисципліни «Паралельні алгоритми та системи» для здобувачів другого (магістерського) рівня освіти спеціальності 124 Системний аналіз ОП «Системний аналіз». Окремі теоретичні результати було використано при виконанні курсових та кваліфікаційних робіт студентами кафедри обчислювальної математики та математичної кібернетики.

4. Відомості про впроваджені об'єкти інтелектуальної власності:

1. Караваєв К.Д., Турчина В.А. Узагальнення задач упорядкування з урахуванням неповного завантаження. *Збірник наукових праць «Питання прикладної математики і математичного моделювання»*, м. Дніпро, 2022. Вип. 22. С. 67-79.
Режим доступу до ресурсу:
https://pm-mm.dp.ua/index.php/pmmm/article/view/341.
(особистий внесок: розглянуто узагальнення задачі упорядкування на випадок неповного завантаження; запропоновано модифікацію алгоритму, заснованого на максимальному паросполученні, яка враховує обмеження на структуру шуканого розв'язку, показано, що відомі та модифікований алгоритм для цієї задачі є лише наближеними).

2. Караваєв К.Д., Турчина В.А. Аналіз впливу автоморфізму графу на схеми направленого перебору. *Збірник наукових праць «Питання прикладної математики і математичного моделювання»*, м. Дніпро, 2021. Вип. 21. С. 94-104.
Режим доступу до ресурсу:
https://pm-mm.dp.ua/index.php/pmmm/article/view/313.
(особистий внесок: досліджена залежність між наявністю автоморфізму у графі та ізоморфізмом його підграфів, отриманих шляхом видаленням з нього відкритих вершин; запропоновано алгоритм для скорочення кількості розгалужень у методі гілок та меж ; показано, що результуюча множина містить всі неізоморфні підграфи).

3. Turchyna V., Karavaiev K. Analysis of algorithms for constructing dense sequencing of digraphs vertices. *Proceedings of The Third International Workshop on Computer Modeling and Intelligent Systems (CMIS-2020)*, Zaporizhzhia, 2020. P. 690-703.
Режим доступу до ресурсу:
https://ceur-ws.org/Vol-2608/paper53.pdf. (**Scopus**)
(особистий внесок: розроблено декілька модифікацій відомого точного алгоритму поліноміальної складності які застосовуються для довільних графів; проведені обчислювальні експерименти, результати яких показали, що запропоновані модифікації є ефективними і суттєво підвищують точність знаходження оптимального розв'язку до 98,5%; додатково доведено, що довільна задача оптимального упорядкування може бути зведена до задачі з графом, який має щільне упорядкування при парній ширині).

4. Турчина В.А., Караваєв К.Д. Дослідження оцінок довжини паралельного упорядкування вершин графу. *Збірник наукових праць «Питання прикладної математики і математичного моделювання»*, м. Дніпро, 2018. Вип. 18. С. 186-195.
Режим доступу до ресурсу:
https://pm-mm.dp.ua/index.php/pmmm/article/view/236.
(особистий внесок: отримана покращена оцінка знизу часових витрат для задачі пошуку паралельного упорядкування вершин графу з мінімальною довжиною, запропонована оцінка зверху, обґрунтований зв'язок задачі із оберненою; на основі обчислювального експерименту досліджено вплив точності оцінки на швидкість знаходження точного розв'язку методом гілок та меж).

5. Пропозиції ради:

Запропоновано впровадити результати дисертаційної роботи Караваєва Костянтина Дмитровича «Методи і алгоритми розв'язання класичних та узагальнених задач упорядкування вершин орграфів» в освітній процес Дніпровського національного університету імені Олеся Гончара.

Голова вченої ради
факультету прикладної математики — Олена КІСЕЛЬОВА

Секретарка — Наталія ЛИСИЦЯ

ДОДАТОК Б. СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА

Наукові праці, в яких опубліковані основні наукові результати дисертації:

1. Караваєв К. Д. Про необхідні умови існування щільних упорядкувань в класичній задачі паралельного упорядкування. *Збірник наукових праць* «*Системні технології*», м. Дніпро, 2024. Вип. 151. С. 76–91. doi: https://doi.org/10.34185/1562-9945-2-151-2024-07. Режим доступу до ресурсу: https://journals.nmetau.edu.ua/index.php/st/article/view/1711.

2. Караваєв К. Д., Турчина В. А. Узагальнення задач упорядкування з урахуванням неповного завантаження. *Збірник наукових праць* «*Питання прикладної математики і математичного моделювання*», м. Дніпро, 2022. Вип. 22. С. 67–79. doi: https://doi.org/10.15421/322207. Режим доступу до ресурсу: https://pm-mm.dp.ua/index.php/pmmm/article/view/341.

3. Караваєв К. Д., Турчина В. А. Аналіз впливу автоморфізму графу на схеми направленого перебору. *Збірник наукових праць* «*Питання прикладної математики і математичного моделювання*», м. Дніпро, 2021. Вип. 21. С. 94–104. doi: https://doi.org/10.15421/322110. Режим доступу до ресурсу: https://pm-mm.dp.ua/index.php/pmmm/article/view/313.

4. Turchyna V., Karavaiev K. Analysis of algorithms for constructing dense sequencing of digraphs vertices. *Proceedings of The Third International Workshop on Computer Modeling and Intelligent Systems (CMIS-2020)*, Zaporizhzhia, 2020. P. 690–703. doi: https://doi.org/10.32782/cmis/2608-53. Режим доступу до ресурсу: https://ceur-ws.org/Vol-2608/paper53.pdf (Scopus).

5. Турчина В. А., Караваєв К. Д. Дослідження оцінок довжини паралельного упорядкування вершин графу. *Збірник наукових праць* «*Питання прикладної математики і математичного моделювання*», м. Дніпро, 2018. Вип. 18. С. 186–195. doi: https://doi.org/10.15421/321819. Режим доступу до ресурсу: https://pm-mm.dp.ua/index.php/pmmm/article/view/236.

Наукові праці, які засвідчують апробацію матеріалів дисертації:

6. Турчина В. А., Караваєв К. Д. Контрприклад до алгоритму перерахування паралельно-послідовних графів без ізоморфізму. *Математичне та програмне забезпечення інтелектуальних систем (MSSIS-2023): Матеріали XXI міжнародної науково-практичної конференції, 22-24 листопада 2023 р.*, м. Дніпро, 2023. С. 291–292. Режим доступу до ресурсу: http://mpzis.dnu.dp.ua/wp-content/uploads/2023/11/mpzis-2023.pdf.

7. Караваєв К. Д., Турчина В. А. Про необхідні та достатні умови наявності автоморфізму у паралельно-послідовних графах. *Комбінаторні конфігурації та їхні застосування: Матеріали XXV Міжнародного науково-практичного семінару імені А. Я. Петренюка, (Запоріжжя – Кропивницький, 14-16 червня 2023 року)* / за ред. Г.П. Донця, м. Запоріжжя - Кропивницький, 2023. С. 122–129. Режим доступу до ресурсу: https://zp.edu.ua/uploads/dept_s&r/2023/conf/1.4/Petrenyuk_ISPS-25-proc.pdf.

8. Караваєв К. Д. Використання інваріантів графів для скорочення напрямленого перебору у задачах упорядкування. *Математичне та програмне забезпечення інтелектуальних систем (MSSIS-2022): Матеріали XX ювілейної міжнародної науково-практичної конференції, 23-25 листопада 2022 р.*, м. Дніпро, 2022. С. 96–97. Режим доступу до ресурсу: http://mpzis.dnu.dp.ua/wp-content/uploads/2022/12/MPZIS-2022-1.pdf.

9. Караваєв К. Д., Турчина В. А. Про зв’язок задач ізоморфізму графів та упорядкування їх вершин. *Комбінаторні конфігурації та їхні застосування: Матеріали XXIV Міжнародного науково-практичного семінару імені А. Я. Петренюка, (Запоріжжя – Кропивницький, 13-14 травня 2022 року)* / за ред. Г.П. Донця, м. Запоріжжя - Кропивницький, 2022. С. 30–33. Режим доступу до ресурсу: https://zp.edu.ua/uploads/dept_s&r/2023/conf/1.4/Petrenyuk_ISPS-25-proc.pdf.

10.Караваєв К. Д., Турчина В. А. Деякі узагальнення задачі паралельного упорядкування. *Комбінаторні конфігурації та їхні застосування: Матеріали XXIII Міжнародного науково-практичного семінару імені А. Я. Петренюка, присвяченого 70-річчю Льотної академії Національного авіаційного університету, (Запоріжжя –*

Кропивницький, 13-15 травня 2021 року) / за ред. Г.П. Донця, м. Запоріжжя - Кропивницький, 2021. С. 93–97. Режим доступу до ресурсу: https://www.glau.kr.ua/images/docs/sbornik/materiali_23_mnp_seminaru.pdf.

11.Karavaiev K., Turchyna V., Hurko O. The problem of consistencing the architecture of computational systems and algorithms. *Сучасні науково-технічні дослідження у контексті мовного простору (іноземними мовами) 13 травня 2021 року: матеріали X Регіональної науково-практичної конференції молодих учених та студентів*, м. Дніпро, 2021. С. 142–145. Режим доступу до ресурсу: https://www.dnu.dp.ua/docs/ndc/2021/19_Сучасні%20науково-технічні%20дослідження%20у%20контексті%20мовного%20простору.pdf.

12.Турчина В. А., Караваєв К. Д. Дослідження оцінки точності алгоритму, заснованого на лексикографічному порядку. *Математичне та програмне забезпечення інтелектуальних систем (MSSIS-2019): Матеріали XVII міжнародної науково-практичної конференції, 20-22 листопада 2019 р.*, м. Дніпро, 2019. С. 258–259. Режим доступу до ресурсу: http://mpzis.dnu.dp.ua/wp-content/uploads/2019/12/MPZIS_2019.pdf.

13.Турчина В. А., Караваєв К. Д. Застосування рівневого принципу до аналізу задач паралельного упорядкування та їх узагальнення. *Міжнародний науковий симпозіум «Інтелектуальні рішення». Обчислювальний інтелект (результати, проблеми, перспективи): праці міжнар. наук.-практ. конф., 15-20 квітня 2019 р.*, м. Ужгород, 2019. С. 56–57. Режим доступу до ресурсу: https://er.chdtu.edu.ua/bitstream/ChSTU/3192/1/OI-2019.pdf.