МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Кваліфікаційна наукова
праця на правах рукопису

БАБИЧ СЕРГІЙ ВАСИЛЬОВИЧ

УДК 519.7:007:004

ДИСЕРТАЦІЯ

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ СКЛАДАННЯ РОЗКЛАДУ ЗАНЯТЬ ЗГІДНО ПЕРМАНЕНТНОЇ ДЕКОМПОЗИЦІЇ

05.13.06 – інформаційні технології

Подається на здобуття наукового ступеня кандидата наук

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

_____/С.В.Бабич/

Науковий керівник Турбал Юрій Васильович, доктор технічних наук, професор

Ідентичність тексту примірників дисертації засвідчую Вчений секретар ерр Ківа Бабровікова

Хмельницький – 2023

АНОТАЦІЯ

Бабич Сергій Васильович. Інформаційна технологія складання розкладу занять згідно перманентної декомпозиції. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня кандидата технічних наук за спеціальністю *05.13.06 – інформаційні технології.* – Хмельницький національний університет, Хмельницький, 2023.

Зміст анотації. У дисертаційній роботі розв'язана актуальна науково–прикладна задача формування матриць розкладів на основі перманентного підходу із застосуванням спеціальних адитивно – диз'юнктивних форм, яка дозволяє вдосконалити процеси автоматизації трудомістких задач формування розкладів, зокрема розкладів занять закладів вищої освіти, з врахуванням низки додаткових критеріїв.

Об'єктом дослідження є процеси складання розкладу занять з використанням перманентного підходу.

Предметом дослідження є моделі, методи та засоби інформаційної технології складання оптимального розкладу занять із використанням перманентного підходу.

У роботі виконано аналіз відомих підходів, методів, інформаційних технологій, інструментів для розв'язання задач календарного планування.

Наявність великої різноманітності відомих підходів зумовлена значною обчислювальною складністю відповідних задач, зокрема, задача складання розкладу занять є багатокритеріальною та NP–повною. А тому для розв'язання таких задач широко використовуються евристичні підходи, експертні оцінки, генетичні алгоритми поряд із оптимізаційними методами та комбінаторними алгоритмами, які мають точні оцінки обчислювальної складності.

Актуальність таких задач зумовлюється ще й тим, що на практиці вони часто мають низку специфічних вимог, що суттєво впливають на алгоритмічну

реалізацію та вибір методу їх розв'язання. Критерії, що виникають при складанні розкладів, часто є антагоністичними і вимагають певних компромісних рішень. Задачі автоматизованої генерації розкладів мають значну обчислювальну складність. А тому розробка нових підходів та алгоритмів до розв'язання задач складання розкладів і сьогодні є актуальною задачею, незважаючи на існування низки відомих методів.

У дисертаційній роботі вперше запропоновано конфігураційний підхід до розв'язання задач складання розкладів, зокрема, формування та оптимізації допустимих матриць розкладів. Досліджено низку властивостей матриць розкладів, запропоновано формалізацію критеріїв, вимог та алгоритмів формування й оптимізації матриць розкладів. Строго обґрунтовано низку тверджень стосовно існування допустимих матриць розкладів для різних умов, зокрема, існування та відсутність нульових елементів («вікон»), наявності та відсутності потоків за умови виконання різних критеріїв.

Основним результатом роботи є розробка низки нових методів та алгоритмів на основі властивостей модифікованих спеціальним чином перманент матриць розкладів. Перманентний підхід до розв'язання задач генерації комбінаторних об'єктів у системах складання розкладів був запропонований вперше. В основі такого підходу лежить процедура розкладання модифікованого перманента за рядком із запам'ятовуванням ідентифікаторів елементів матриці, що дозволяє вибирати зручні структури даних та здійснювати миттєвий безпосередній запис окремих складових об'єктів, що генеруються. Зауважимо, що навіть процес обрахунку стандартного алгебраїчного перманента є NP-повною задачею. В даному випадку, мова йде про суттєву модифікацію як матриці інцидентності, так і самої процедури обрахунку перманента. В процесі обчислення модифікованого перманента важливим є не саме значення перманента, а саме процедура його розкладу, в ході якої формуються необхідні конфігурації розкладів.

Оскільки алгоритми, що пропонуються у роботі, є новими, як і сам перманентний підхід, суттєва увага приділяється складності відповідних алгоритмів та задач. Перманентний підхід порівнюється з відомими підходами до генерації комбінаторних об'єктів, зокрема, розглядається як тестова задача генерація перестановок з точки зору обчислювальної складності. Показано, що алгоритми на основі перманентного підходу на тестових задачах належать до одного і того ж класу складності, що і відомі підходи, які базуються, наприклад, на концепції лексикографічного порядку чи мають незначну перевагу по кількості арифметичних операцій (наприклад, метод перманентної декомпозиції за кількістю арифметичних операцій на 46% ефективніший від методу, що ґрунтується на відношенні порядку). Однак, перманентний підхід має низку переваг за рахунок певної універсальності – він дозволяє розв'язувати як відносно прості задачі, типу генерації перестановок, так і значно складніші – генерації розкладів за наявності потоків, пар «мигалок» (чисельник – знаменник) тощо. Відповідно, він надає можливість розробникам універсалізувати структури даних.

Оскільки результатом розкладу модифікованого перманента є усі можливі системи різних представників множин, що утворюються стовпцями матриць розкладів, то виникає проблема – як згенерувати на основі таких систем різних представників (СРП) всі можливі конфігурації розкладів, що задовольняють необхідним критеріям. Для розв'язання цієї проблеми у роботі вперше запропоновано спеціальне числення адитивно-диз'юнктивних форм (АДФ): введено означення АДФ, описано основні властивості, запропоновано застосування АДФ у процедурі декомпозиції перманента, що дозволяє генерувати усі допустимі варіанти розкладів у процесі декомпозиції модифікованого перманента матриці інцидентності. На основі відповідного підходу запропоновані відповідні алгоритмічні рішення, які реалізовані у відповідному програмному забезпеченні.

Перманентний підхід, що запропонований у роботі, та низка технічних рішень, зокрема, що ґрунтуються на АДФ, має універсальний характер та може бути використаний для розв'язання широкого кола задач у теорії розкладів.

В сучасних умовах надзвичайно важливою є проблема якомога ширшої участі усіх стейкхолдерів у процесах реалізації освітніх програм закладів вищої освіти, якомога ширше врахування їх інтересів. В роботі ця проблема розглядається у контексті врахування інтересів стейкхолдерів у процесі формування розкладу занять в закладі вищої освіти. Пропонується евристичний підхід до проблеми автоматизованого складання розкладу занять, в межах якого інтереси широкого кола стейкхолдерів максимально враховуються. Відповідний підхід був апробований при складанні розкладу занять Рівненського державного гуманітарного університету. Запропоновано відповідний програмний комплекс та низка оригінальних рішень, зокрема специфічна система кодування даних, оптимізовані з точки зору пам'яті структури даних, реалізовано основні алгоритми з використанням побітових операцій. Запропонована інформаційна система має практичне значення та може використовуватись у процесах генерації допустимих матриць розкладів.

Основні наукові результати дисертації опубліковано в 13 працях, зокрема: одна стаття [1] у періодичному науковому виданні держави, що входить до Організації економічного співробітництва та розвитку та/або Європейського Союзу; сім статей [2–8] у наукових фахових періодичних виданнях України; чотири публікації [9–13] у матеріалах міжнародних та всеукраїнських наукових, науково–технічних конференцій. З них три роботи входить до міжнародної наукометричної бази Scopus [9–11], одна робота [7] входить до міжнародної наукометричної бази Web of Science, одна робота [10] входить до міжнародної наукометричної бази Index Copernicus.

Ключові слова: інформаційна технологія, перманент, декомпозиція, матриця розкладу, адитивно–диз'юнктивна форма, алгебраїчна структура.

ABSTRACT

Babich S.V. Information technology for drawing up a schedule of classes according to permanent decomposition. – Qualifying scientific work on manuscript rights.

Dissertation for obtaining the scientific degree of candidate of technical sciences in the specialty 05.13.06 – information technologies. – Khmelnytskyi National University, Khmelnytskyi, 2023.

Abstract content. The dissertation solves the actual scientific and applied problem of forming timetable matrices based on a permanent approach with the use of special additive–disjunctive forms, which allows for improving the processes of automating time–consuming tasks of forming timetables, in particular timetables of higher education institutions, taking into account several additional criteria.

The object of the study is the process of drawing up an optimal schedule of classes using a permanent approach.

The subject of the study is the models, methods, and means of information technology for making an optimal schedule of classes using a permanent approach.

The work includes an analysis of known approaches, methods, information technologies, and tools for solving calendar planning problems.

The presence of a wide variety of known approaches is due to the significant computational complexity of the corresponding tasks, in particular, the task of creating a class schedule is multi–criteria and NP-complete. Therefore, heuristic approaches, expert evaluations, and genetic algorithms, along with optimization methods and combinatorial algorithms, which have accurate estimates of computational complexity, are widely used to solve such problems.

The relevance of such problems is also determined by the fact that in practice they often have several specific requirements that significantly affect the algorithmic implementation and the choice of the method of their solution. The criteria that arise when drawing up schedules are often antagonistic and require certain compromise

solutions. The tasks of automated schedule generation have significant computational complexity. Therefore, the development of new approaches and algorithms for solving scheduling problems is still an urgent task today, despite the existence of some known methods. In the dissertation, for the first time, a configurational approach to the solution of scheduling problems is proposed, in particular, the formation and optimization of admissible scheduling matrices. Some properties of schedule matrices were studied, and formalization of criteria, requirements, and algorithms for the formation and optimization of schedule matrices was proposed. Some assertions regarding the existence of admissible matrices of schedules for various conditions are strictly substantiated, in particular, the existence and absence of zero elements, and the presence and absence of flows provided that various criteria are met.

The main result of the work is the development of many new methods and algorithms based on the properties of specially modified permanent matrices of schedules. A permanent approach to solving the problems of generating combinatorial objects in scheduling systems was proposed for the first time. The basis of this approach is the procedure of decomposing the modified permanent by row with memorization of the identifiers of the matrix elements, which allows you to choose convenient data structures and carry out the instant direct recording of individual component objects that are generated. Note that even the process of calculating the standard algebraic constant is an NP–complete problem. In our case, we are talking about a significant modification of both the incidence matrix and the permanent calculation procedure itself. In the process of calculating the modified permanent, it is not the value of the permanent that is important, but rather the procedure of its schedule, during which the necessary schedule configurations are formed.

Since the algorithms proposed in the paper are new, as is the permanent approach itself, significant attention is paid to the complexity of the corresponding algorithms and tasks. The permanent approach is compared with known approaches

to the generation of combinatorial objects, in particular, the generation of permutations is considered a test task from the point of view of computational complexity. It is shown that algorithms based on the permanent approach to test problems belong to the same complexity class as well–known approaches that are based, for example, on the concept of lexicographic order or have a slight advantage in terms of the number of arithmetic operations (for example, the method of permanent decomposition by the number of arithmetic operations is 46% more efficient than the method based on the order ratio). However, the permanent approach has several advantages due to a certain universality – it allows solving both relatively simple tasks, such as the generation of permutations, and much more complex ones – the generation of schedules in the presence of flows, pairs of "blinkers" (numerator – denominator), etc. Accordingly, it enables developers to universalize data structures.

Since the result of the schedule of the modified permanent is all possible systems of different representatives of sets formed by the columns of the schedules matrices, the problem arises – how to generate based on such systems of different representatives (SRP) all possible configurations of schedules that satisfy the necessary criteria. To solve this problem, a special calculation of additive–disjunctive forms (ADF) is proposed for the first time in the work: the definition of ADF is introduced, the main properties are described, the application of ADF in the permanent decomposition procedure is proposed, which allows generating all admissible variants of schedules in the process of decomposition of a modified permanent incidence matrix. Based on the appropriate approach, appropriate algorithmic solutions are proposed, which are implemented in the appropriate software.

The permanent approach proposed in the work and several technical solutions, in particular, based on ADF, have a universal character and can be used to solve a wide range of problems in the theory of schedules. In today's conditions, the problem of the widest possible participation of all stakeholders in the processes of

implementing educational programs of higher education institutions, the widest possible consideration of their interests, is extremely important. In the work, this problem is considered in the context of taking into account the interests of stakeholders in the process of forming the schedule of classes in a higher education institution. A heuristic approach to the problem of automated preparation of the class schedule is proposed, within which the interests of a wide range of stakeholders are taken into account as much as possible. The corresponding approach was tested when drawing up the class schedule of the Rivne State Humanitarian University. An appropriate software package and some original solutions are proposed, including a specific data encoding system, optimized from the point of view of the memory of the data structure, and basic algorithms using bitwise operations are implemented. The proposed information system is of practical importance and can be used in the processes of generating admissible matrixes of schedules.

The main scientific results of the dissertation were published in 13 works, in particular: one article [1] in periodical scientific publications of other countries that are members of the Organization for Economic Cooperation and Development and/or the European Union; seven articles [2–8] in scientific and professional periodicals of Ukraine; four publications [9–13] in the materials of international and Ukrainian scientific, scientific and technical conferences. Of them, three works are included in the international scientometric database Scopus [9–11], one work [7] is included in the international scientometric database Web of Science, and one work [10] is included in the international scientometric database Index Copernicus.

Keywords: information technology, permanent, decomposition, decomposition matrix, additive–disjunctive form, algebraic structure.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Статті у періодичних наукових виданнях інших держав, які входять до Організації економічного співробітництва та розвитку та/або Європейського Союзу:

1. Turbal Y. V., Babych S. V. Methods of the schedule matrix forming based on the modified permanent. *Telekomunikacja i Elektronika. Zeszyty Naukowe. Uniwersytet Technologiczno–Przyrodniczy Im. Jana i Jedrzeja Sniadeckich w Bydgoszczy.* 2018. Vol. 268, No. 21. Pp. 85–92.

Статті у періодичних фахових наукових виданнях України:

2. Turbal Y. V., Babych S. V., Kunanets N. E. Permanent Decomposition Algorithm for the combinatorial object's generation. *Radio Electronics, Computer Science, Control.* 2022. Vol. 2. Pp. 74–79. (Фахове видання України, WoS, категорія А).

3. Бабич С. В. Алгоритм побудови допустимої матриці розкладів. *Вісник Національного університету водного господарства та природокористування. Сер. Технічні науки.* 2014. Вип. 4, ч. 68. С. 274–281.

4. Бабич С. В. Активне навчання студентів у проектній формі. Управління проектами. Порівняння проектного та операційного підходів. *Вісник НУВГП. Сер. Технології навчання.* 2015. Вип. 14. С. 88–94.

5. Бабич С. В., Турбал Ю. В. Методи формування матриць розкладів на основі модифікованих перманент. *Інформаційні системи та мережі.* 2017. Вип. 872. С. 204–209.

6. Бабич С. В., Турбал Ю. В. Програмне забезпечення задач календарного планування на основі конфігураційних підходів. *Вісник НУВГП. Сер. Технічні науки.* 2022. Вип. 2, ч. 98. С. 258–267.

7. Бабич С. В. Інформаційна технологія генерації матриць розкладів згідно перманентної декомпозиції. Міжнародний науково–технічний журнал

«Вимірювальна та обчислювальна техніка в технологічних процесах». 2022. Вип. 4. С. 120–127. URL: https://doi.org/10.31891/2219-9365-2022-72-4-17.

8. Turbal Y. V., Babych S. V. Information technology for the schedule generation based on the algebra of additive–disjunctive forms and the modified method of permanent decomposition. *Computer sytems and information technologies*. 2022. Vol. 4, No. 9. Pp. 120–127. URL: https://doi.org/10.31891/CSIT.

Публікації у матеріалах конференцій:

9. Turbal Y. V., Babych S. V., Bachyshyna L., Kunanets N. E. and Kovalchuk N.. Modification of the Permanent Decomposition Method for the Meeting Schedule Problem: *The 1st International Workshop on Information Technologies: Theoretical and Applied Problems (ITTAP–2021)*. Ternopil, Ukraine, 2021. Pp. 126–131. (Scopus).

10. Turbal Y.V., Babych S.V., Kunanets N. E., Melnyk L. and Pasichnyk V. "Permanent" algorithm for the meeting shedule problem: *IEEE 16th International Conference on Computer Sciences and Information Technologies (CSIT)*. 2021. Pp. 355–359. (Scopus).

11. Бабич С.В., Турбал Ю.В. Алгоритм формування матриці розкладів в задачах календарного планування: *Сучасні проблеми математичного моделювання та обчислювальних методів:* матеріали Всеукраїнської наукової конференції. Рівне, Україна, 2015. С. 6.

12. Бабич С.В., Турбал Ю.В. Алгоритми оптимізації матриць розкладу за базовими критеріями в межах конфігураційного підходу: *XXIV International conference PDMU*. Skhidnytsia, Ukraine, 2014. Pp. 97–98.

13. Бабич С.В., Турбал Ю.В. Алгоритм побудови допустимої матриці розкладів: *XXV International conference PDMU*. Odesa, Ukraine, 2015. Pp. 60–61.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

АДФ (ADF) – адитивно–диз'юнктивна форма

ГА – генетичні алгоритми

ДАФ – диз'юнкція адитивних форм

ДФ – диз'юнктивна форма

ЗВО – заклад вищої освіти

ІС – інформаційна система

ІТ – інформаційна технологія

СРП (SRP) – система різних представників

СРПК (SRPC) – система різних представників конфігурацій

СРПС (SRPS) – система різних представників стовпців

Per – перманент матриці

Permod – модифікований перманент матриці інцидентності

R – матриця розкладу

V – матриця вимог

ВСТУП

Актуальність теми. Задачі складання розкладів сьогодні не втрачають своєї актуальності, незважаючи на наявність великої кількості теоретичних результатів і практичних підходів до їх вирішення, напрацьованих щонайменше протягом останніх ста років. І це не дивно, адже проблема складання розкладу – одна із найскладніших задач прикладної математики. Вона характеризується наявністю багатьох критеріїв, вага яких на практиці може бути різною навіть для схожих на перший погляд задач, необхідністю розв'язувати проблему впорядкування значних за розміром дискретних множин, що призводить до виникнення алгоритмів експоненційної складності, відповідні задачі часто є NP-повними.

На практиці іноді необхідно враховувати і людський фактор, психолого-педагогічні особливості, що призводить до проблем виключно формалізованої постановки задачі та автоматизації процесу її оптимального вирішення. Виникає необхідність у застосуванні експертних оцінок, певних евристичних підходів.

Складність відповідних точних оптимізаційних алгоритмів при розв'язанні задач квадратичного програмування, що часто тут виникають, досліджувалась у низці робіт [15–20]. Вона зумовлює широке застосування певних альтернативних підходів, де пропонувалось знаходження близького до оптимального розв'язання за допомогою певних евристичних алгоритмів [20–23]. Так, наприклад, останніми роками при складанні розкладів занять ЗВО почали широко застосовуватись генетичні алгоритми, які, як відомо, дають розв'язок, близький до оптимального, та вимагають окремого дослідження їх ефективності, збіжності [24, 25].

Розв'язання задач складання розкладів занять, зустрічей вимагає застосування нестандартних методів, оригінального творчого мислення і глибокого розуміння суті й складності проблеми. Але для організацій типу

закладів вищої освіти вирішення цієї проблеми вкрай необхідно, оскільки розклад або графік є основним інструментом управління часом, від нього безпосередньо залежить, наприклад, продуктивність праці викладачів і швидкість навченості студентів, а значить і ефективність ЗВО в цілому.

У школі кібернетики та інформатики, котра створена В. М. Глушковим, досягнуто значних результатів у напрямку розвитку новітніх інформаційних технологій, зокрема, розв'язання задач складання розкладів. У цьому контексті можна відзначити, зокрема, роботи В. М. Томашевського, В. С. Моркуна, В.Є. Снитюка, М. Д. Годлевського, С. Д. Штовби.

Передумовою побудови відповідних програмних комплексів для автоматизації задач складання розкладів стали роботи Baptiste Philippe, Blazewicz Jacek, Burke Edmund, Peter Brucker, Kendall Graham, Lawler Eugene Leighton (Gene), Leung Joseph Y.T., Werner Frank [26–34].

А тому розробка нових підходів у алгоритмізації задач календарного планування та розробка відповідного програмного забезпечення є актуальною науково–практичною задачею [35–44].

Зв'язок роботи з науковими програмами, планами, темами. Дисертаційне дослідження проведено у рамках науково–дослідної роботи Національного університету водного господарства та природокористування, зокрема наукової теми «Методи ідентифікації параметрів та математичні моделі, що ґрунтуються на базі багатоканальних систем масового обслуговування», номер держреєстрації №0114U0011181, де здобувачу належить реалізація окремих алгоритмів для моделювання багатоканальних систем масового обслуговування та оптимізації кількості обслуговуючих пристроїв.

Мета і завдання дослідження. Метою роботи є підвищення ефективності інформаційних технологій формування розкладів, в основу яких покладено методи перманентної декомпозиції.

Задачі дослідження:

1. Дослідити сучасні методи та інформаційні технології складання розкладів занять у закладах освіти.

2. Розробити нові критерії для перевірки допустимості матриць розкладів та можливості оптимізації їх структури в межах конфігураційного підходу, що дозволить вдосконалити існуючі процедури аналізу даних у програмних комплексах для автоматизованого створення розкладів.

3. Дослідити можливість створення нових ефективних методів генерації комбінаторних об'єктів, в основі яких покладено процедуру перманентної декомпозиції.

4. Модифікувати класичні визначення перманент та матриць інцидентності систем різних представників множин так, щоб вони враховували наявність потоків у матрицях розкладів.

5. Розробити нові підходи до розв'язання задач складання розкладів у межах перманентного підходу на основі систем різних представників конфігурацій шляхом використання спеціальних адитивно–диз'юнктивних форм.

6. Розробити інформаційну технологію для формування розкладів, що максимально враховує інтереси усіх стейкхолдерів.

Об'єктом дослідження є процеси складання розкладу занять з використанням перманентного підходу.

Предметом дослідження є моделі, методи та засоби інформаційної технології складання розкладу занять з використанням перманентного підходу.

Методи дослідження. Для досягнення поставленої у дисертаційній роботі мети використані методи системного та порівняльного аналізу для обґрунтування актуальності та постановки наукового завдання; теоретико–множинні підходи при розробці алгоритмів формування матриць розкладів на основі систем різних представників конфігурацій; методи формальних алгебр

при розробці концепції адитивно-диз'юнктивних форм; апарат модельно–орієнтованих підходів та методи формування ієрархічних систем у процесі розробки програмного комплексу з використанням мови програмування С++.

Наукова новизна отриманих результатів.

1. Вперше розроблено критеріальний метод перевірки допустимості матриць розкладів, який на відміну від відомих методів враховує конфігурації матриць, що дає змогу здійснити аналіз коректності вхідних даних.

2. Вперше розроблено метод перманентної декомпозиції, який відрізняється від відомих тим, що здійснює генерацію комбінаторних об'єктів згідно модифікованих перманент матриць інцидентності та дає змогу покращити процеси генерації матриць розкладів.

3. Вперше розроблено метод складання розкладу згідно адитивно–диз'юнктивних форм, який відрізняється від відомих методів використанням операцій вибору та включення та дає змогу створювати засоби, які на основі систем різних представників множин, що утворюються стовпцями матриць розкладів, генерують усі варіанти матриць розкладів у відповідності до заданих додаткових вимог.

4. Отримала подальший розвиток інформаційна технологія складання розкладів за принципами перманентної декомпозиції, яка дає змогу генерувати розклади з врахуванням заданих критеріїв та вимог стейкхолдерів, яка на відміну від відомих ІТ враховує пріоритети вимог стейкхолдерів.

Практичне значення отриманих результатів. Запропоновані у роботі алгоритми формування комбінаторних об'єктів на основі модифікованих перманент матриць інцидентності дозволили удосконалити сучасне інформаційне забезпечення для задач складання розкладів занять у закладах вищої освіти. Методи, представлені в дисертаційному дослідженні, реалізовано у відповідному програмному комплексі для формування розкладу занять. Відповідне програмне забезпечення було апробовано у Рівненському державному гуманітарному університеті на факультеті математики та

інформатики. Результати роботи впроваджено у навчальний процес Національного університету водного господарства та природокористування, Рівненського фахового коледжу при Національному університеті біоресурсів та природокористування України (акт впровадження від 12.12.2022 р.), у Рівненській обласній лікарні для складання графіків консультацій лікарів (акт впровадження від 12.12.2022 р.), а також у ТОВ «СМАРТ КІНГ ЛТД» для складання розкладів scrum-зустрічей працівників компанії (акт впровадження від 12.04.2023 р.) Акти впровадження результатів дисертаційної роботи наведені у додатку Г.

Особистий внесок здобувача. Усі наукові результати дисертаційної роботи сформульовані та отримані автором самостійно. У наведених працях, опублікованих зі співавторами, здобувачеві належать: декілька нових підходів та алгоритмів до формування допустимої матриці розкладів [1], запропоновано узагальнення методу перманентної декомпозиції на більш ширший клас задач генерації комбінаторних об'єктів [2, 3], здійснено оцінку складності алгоритмів у термінах О-оцінювання, здійснено дослідження допустимості матриць розкладів для різних варіантів конфігурацій [4], запропоновано новий підхід до алгоритмічної реалізації процесу формування різних представників стовпців матриці розкладів [5], запропоновано структури даних та здійснено програмну реалізацію процесу перманентної декомпозиції [6], запропоновано метод формування матриць розкладів на основі певних модифікацій перманент [7], реалізовано концепцію адитивно-диз'юнктивних форм у алгоритмах формування розкладів на основі систем різних представників [10], уточнено алгоритм формування матриці розкладів у задачах календарного планування [11], реалізовано алгоритми «витіснень» для формування матриць розкладів [12], запропоновано та реалізовано програмно алгоритми оптимізації матриць розкладу за базовими критеріями у межах конфігураційного підходу [12, 13], узагальнено концепцію адитивно-

диз'юнктивних форм та визначено класи задач з можливим їх застосуванням [9, 8].

З наукових робіт, опублікованих у співавторстві, у дисертаційній роботі використані результати особистих досліджень здобувача.

Апробація результатів дисертації. Результати дисертаційної роботи доповідались на конференціях: "Сучасні проблеми математичного моделювання та обчислювальних методів" – матеріали Всеукраїнської наукової конференції, м. Рівне, 2015р.; "Сучасні проблеми прикладної математики та інформатики (APAMCS – 2015)" – матеріали міжнародної конференції, Львів, 24-25 вересня, 2015р.; "Intel ITSIS' 2021: 2nd International Workshop on Intelligent Information Technologies and Systems of Information Security", March 24-26, 2021, Khmelnytskyi; Computational & Information Technologies for Control & Modeling (CITCM), 2021, Rivne; "Problems of decision making under uncertainties (PDMU – 2015)", XXVI Int. Conference, Shidnitsa, Ukraine, May 11-15 – 2014; "Problems of decision making under uncertainties (PDMU – 2015)": XXVII Int. Conference,Odesa, Ukraine, August, 23-28 – 2015.

Публікації. Основні наукові результати дисертації опубліковано в 13 працях, зокрема: сім статей [2–8] у наукових фахових періодичних виданнях України, з яких одна стаття [2, Web of Science] в журналі категорії А, шість статей в журналах категорії Б; одна стаття [1, Index Copernicus] у періодичному науковому виданні держави, яка входить до Організації економічного співробітництва та розвитку Європейського Союзу; чотири публікації [9–13] у матеріалах міжнародних та всеукраїнських наукових, науково-технічних конференцій, з них дві роботи включені до міжнародної наукометричної бази Scopus [7, 8].

Структура та обсяг дисертації. Дисертація складається зі вступу, чотирьох розділів, висновків, списку використаних джерел із 158 найменувань та додатків. Повний обсяг дисертації складає 194 сторінки, основний зміст викладено на 160 сторінках, де наведено 34 рисунки.

РОЗДІЛ 1
ОГЛЯД ВІДОМИХ ПІДХОДІВ, МЕТОДІВ ТА АЛГОРИТМІВ ДО РОЗВ'ЯЗАННЯ ЗАДАЧ СКЛАДАННЯ РОЗКЛАДІВ

1.1. Проблематика задач складання розкладів

Задача складання розкладів занять закладів освіти належить до задач теорії розкладів – розділу дослідження операцій, що вивчає задачі, в котрих необхідно упорядкувати або визначити послідовність виконання сукупності робіт за умови обмеженості в ресурсах. Історія досліджень розпочалась ще з робіт Г. Ґанта, який запропонував новий спосіб представлення розкладу [44], що отримав назву «Діаграма Ґанта» (схематичне зображення календарного плану) та конвеєрного виробництва Г. Форда.

Перші згадки про подібні рішення в Україні були в 60-70-ті роки, коли розроблялась автоматизована система управління на Львівському заводі телевізорів. У подальшому ці методики були прийняті японськими автомобільними заводами як методи «Канбан» та «Just-in-time».

Належність задачі складання розкладів занять до проблематики календарного планування призводить до можливості застосування для її розв'язку класичних методів теорії розкладів із врахуванням інтерпретації навчальної аудиторії як обслуговуючого пристрою у виробничому процесі. Ця задача не втрачає своєї популярності, незважаючи на вражаючу кількість науковців, що працюють у даній галузі та наявність значної кількості публікацій. Це пов'язане з обчислювальною складністю самої задачі складання розкладу занять, її багатокритеріальністю, наявністю низки специфічних вимог до розкладу у різних навчальних закладах, що відображає їх особливості, рівень сформованості культури якості та ускладнює адаптацію якихось відомих рішень. Належність задачі складання розкладу до класу NP-повних задач підкреслює її теоретичну важливість та актуальність розробки

нових підходів та алгоритмів її розв'язання, адже використання точних методів комбінаторної оптимізації при повному переборі варіантів може зайняти занадто багато часу і ресурсів [18–20]. Це призводить до того, що у багатьох закладах вищої освіти України, наприклад, розроблені власні методи, інформаційні технології та програмне забезпечення для складання розкладів [35–43].

Слід зважати також, що задача складання розкладу занять належить до класу задач календарного планування з врахуванням людського фактору, а тому має суттєві особливості. Адже часто виникає необхідність врахування інтересів усіх стейкхолдерів у процесі планування. Тоді, залежно від поставлених цілей, бажане використання набору різних інструментів оцінки в рамках міжособистісного, рольового, проблемно орієнтованого, цільового, компетентнісного підходів, які використовують, зокрема, при роботі з командами. В такому контексті задачі складання розкладу занять для ЗВО, наприклад, мають низку спільних рис із проблематикою управління проектами.

Для усестороннього аналізу проблематики складання розкладів недостатньо простого огляду, а необхідно провести глибокий бібліометричний аналіз [47]. Такий аналіз стосовно задач календарного планування був здійснений у роботах [46–50], де проведено, зокрема, бібліометричний аналіз генетичних алгоритмів, сіток і хмар, технологій Cloud (технології та генетичні алгоритми недомінованого сортування NSGA II). Систематичний огляд допомагає отримати нове розуміння та розкрити значущі знання на основі накопичених даних дослідницької галузі [43].

Існуючі оглядові роботи не зосереджувалися на загальних питаннях. Автори здебільшого пішли вузьким шляхом, зосереджуючись на конкретних темах або сферах застосування алгоритмів планування [124–128]. Ці дослідження [51–54] були зосереджені на широкому огляді методів планування у хмарних обчисленнях. Дослідження [55] зосереджено на огляді

алгоритмів планування операційної системи, тоді як інші дослідження [56] зосереджено на плануванні в grid-обчисленнях.

Проводячи огляд статей та робіт, що перекликаються із задачами календарного планування [129–134], варто зауважити, що більшість «слабких місць» та описаних, не вирішених, проблем – це наявність критеріїв, що унеможливлюють використання «єдиновірного» алгоритму чи концепції методики використання теорії календарного планування. Дану концепцію можна простежити у роботі [57], в котрій автори вказали на проблему створення розкладу саме для співробітників з врахуванням людського фактору, та роботах [135–142].

Результати, до котрих призводить те чи інше впорядкування, суттєво відрізняються [150–154]. Використовуючи матричні дані та маючи одну з цілей реалізації потенціалу алгоритму – оптимізація процесів управління проектами, було оглянуто досягнення в роботі актуальних досліджень [58], де пропонується матричний багатомодовий алгоритм планування проектів з використанням моделювання багаторівневої архітектури. Одним із вразливих місць даного алгоритму, знову таки, є наявність фіксованих обмежень, критеріїв (обчислюються лише запланованими завданнями та їх вимогами), що провокують зниження ефективності використання алгоритму (огляд вказав на недолік, що прямим чином компенсований у даній роботі).

Також PDM-матричні обчислення з використанням ERP-алгоритму простежуються в роботі [59], де запропоновано новий метод планування проектів на основі матриці, що враховує важливість завдання або ймовірність завершення, визначаючи (з ранжуванням) важливість та ймовірність можливих сценаріїв і структур проекту.

Проблему складання розкладів [143–149] занять можна розглядати як частину більш загальної проблеми «*resource scheduling*», яка є домінуючою в системах класу MRP (планування необхідності ресурсів), MRP II (планування ресурсів на виробництві) та ERP (планування ресурсів підприємства в цілому)

[60–62]. Саме це визначає можливості застосування до розв'язання задач складання розкладів занять підходів, реалізованих у таких системах.

Якщо враховувати людський фактор при складанні розкладів, то системи MRP відходять від старої логіки роботи з ресурсами чи виробничими процесами й стає схожим на новітню її версію «*HR-MRP*» (планування необхідності людських ресурсів), з акцентом саме на оперування та організацію людських ресурсів та роботи з ними [3]. Окрім згаданих вище, слід згадати про *constraint satisfaction problem* [63–65], *job shop* [66, 67], *prioritized task scheduling* [68–69], *random task scheduling* [70], *dynamic resource scheduling* [71, 72], розв'язки яких можна використати в задачах складання розкладів занять.

Публікації з проблеми автоматизованої генерації розкладу занять ЗВО можна поділити на декілька категорій, зокрема: формалізація предметної області [17, 19, 35, 43], аналіз існуючих та розробка нових алгоритмів й методів складання розкладу [18, 20–22, 33], аналіз існуючих та розробка нових програмних засобів генерації комп'ютерного розкладу: технології, які використовуються, та їх програмна реалізація [34–44].

Наприклад, в [39, 40] розглянуто математичну постановку задачі складання розкладу занять у ЗВО в умовах розбіжності вимог і побажань викладачів та студентів, введено поняття «віртуальних та узагальнених груп і підгруп», проаналізовано жорсткі і нежорсткі обмеження в задачі складання розкладу. Робота [18] присвячена стандартизації даних для складання розкладу в навчальних закладах.

Таким чином, з проведеного аналізу бачимо, що складання розкладу занять ЗВО є актуальною та практично значимою задачею, що потребує розробки нових методів та алгоритмів. Для їх деталізації необхідно розглянути існуючі методи для складання розкладів занять.

1.2. Методи, на яких базуються інформаційні технології для складання розкладів занять.

Існуючі задачі складання навчального розкладу розрізняються кількістю, видом обмежень та критеріями оптимальності. В роботах [73, 74] проведено огляд методів складання розкладу ЗВО, проаналізовані їхні переваги та недоліки. Наприклад у [36] розглянуто математичний підхід до вирішення задачі на основі різних побажань викладачів та студентів до розкладу у вищих навчальних закладах, введено поняття узагальнених та віртуальних груп. Також проаналізовано групи обмежень для задачі створення розкладів вищих навчальних закладів. В роботі [18] розглядають процес стандартизації вимог до складання розкладів.

Для автоматичного складання широко використовуються як класичні методи (лінійне цілочисельне програмування, метод розфарбовування графу, метод імітаційного моделювання), так і метаевристичні методи [18–20, 36, 41].

Задачі складання розкладу належать до класу комбінаторних, для яких суттєве значення має розмірність [119–123], яка може бути настільки великою, що розв’язати їх простим перебором варіантів є неможливо. Часто такі задачі зводяться до задач цілочисельного лінійного програмування [73, 76], для розв’язання яких використовуються методи відсікання, гілок та меж. Традиційними методами дослідження операцій для задач планування є комбінаторні процедури, імітаційне моделювання, мережні методи й евристичні підходи.

Задача цілочислового програмування зводиться до формалізації змінних, значення яких необхідно знайти, побудови математичної моделі задачі у вигляді обмежень, що описують задачу і накладають певні обмеження на змінні та побудови цільової функції [39, 40, 42]. Основні недоліки: експоненціальне збільшення витрат часу на пошук кращого (прийнятного) рішення із зростанням розмірності розв'язуваної задачі, відсутність гарантії

отримання прийнятного рішення, в силу великої розмірності математичної моделі складно оцінити вплив різних чинників на процес вирішення завдання і його результат.

Суть застосування агентного підходу для розв'язання будь-якої задачі є наступною: розділення завдання на дрібніші, для вирішення кожного з яких виділяється агент. Мета агента – знайти такий розв'язок, щоб він узгоджувався з рішеннями інших агентів. Агенти домагаються узгодження один з одним обміном інформаційними повідомленнями. Недоліки: відсутність гарантії отримання прийнятного розкладу занять; практично неможливо оцінити вплив значень параметрів для внутрішньої логіки кожного з агентів на результат розв'язання задачі.

Оскільки задачу складання розкладу занять можна віднести до класу NP–важких задач зі значною кількістю обмежень і складністю побудови математичної моделі, то використання класичних методів обмежене. Тому натепер для розв'язання задачі складання розкладу ЗВО дуже поширене застосування метаевристичних методів [20, 77, 78]. Застосовують такі методи як імітацію відпалу, генетичні алгоритми, метод мурашиних колоній тощо.

Побудувати єдину класифікацію методів складання навчального розкладу досить складно, однак можемо виділити окремі групи за певними класифікаційними ознаками, зокрема:

– методи комбінаторної оптимізації;

– декомпозиційні методи;

– методи евристичного пошуку;

– метаевристичні методи.

Методи комбінаторної оптимізації включають задачі генерації розміщень дисциплін у таблиці розкладу і включають, зокрема, алгоритми послідовного почергового розміщення дисциплін у таблицю до того часу, поки вона буде заповнена [73]. При цьому можна формально ввести критерії оптимальності, які максимально враховують додаткові вимоги, що дозволяє

говорити в певному сенсі про оптимальність. У таких задачах доволі часто використовується представлення додаткових вимог за допомогою графів, у яких в якості вершин виступають окремі дисципліни, а інформація про сумісність окремих дисциплін представляється як дуги у графі (наприклад, якщо два заняття несумісні – утворюється дуга). В такому випадку з врахуванням сумісності окремих занять та графового представлення можна прийти до задачі про розмальовку графа. [79, 80].

В процесі заповнення таблиці розкладу застосовуються різні евристики, наприклад такі [80]:

– в таблицю розкладу спочатку розставляються заняття, які мають найбільшу кількість несумісних із ними за часом занять, таким чином будується пріоритезація;

– аналог попередньої евристики, при якому вводиться ваговий коефіцієнт, що відповідає числу студентів, задіяних на занятті, при впорядкуванні кількість занять, несумісних за часом із біжучим заняттям, множиться на цей ваговий коефіцієнт (заняття з великою кількістю студентів отримують більший пріоритет);

– пріоритет отримують заняття, для яких в часовій сітці є мінімальна кількість вільних періодів часу;

– насамперед у таблицю розкладу розставляються заняття, які несумісні з великою кількістю занять у складі вже розставлених.

Описаний вище евристичний підхід має певні недоліки. Наприклад, для врахування особливостей кожної конкретної задачі та алгоритмічної реалізації необхідно будувати особливу систему евристик. Знижує привабливість використання описаного підходу і його низька ефективність для фарбування графів великої розмірності.

Однією з найважливіших властивостей методів комбінаторної оптимізації можна назвати те, що вони гарантують знаходження розв'язку задачі, якщо він існує. Але при цьому очевидно, що може трапитись найгірша

ситуація, коли оптимальний розв'язок знайдено в самому кінці пошуку. За умов задач великої розмірності такий підхід може бути неприйнятним із практичних міркувань. До таких методів можна віднести методи математичного цілочисельного програмування, метод пошуку в графі у глибину та ширину, метод гілок і границь, методи, що ґрунтуються на теорії потоків у мережах. Перелічені методи широко використовувалися при вирішенні завдань складання розкладу занять.

Відзначимо низку їх особливостей та недоліків.

При використанні методу цілочисельного програмування, наприклад, на практиці виникають проблеми представлення обмежень у стандартній формі як лінійні обмеження. Більшість обмежень у задачі складання оптимального навчального розкладу формулюються за допомогою деякого логічного виразу, що складається з операцій алгебри та логічних операцій над предикатами. Такий загальний вид обмежень обґрунтований теоретико-множинною природою завдань складання навчального розкладу [45]. При цьому їх необхідно представити як лінійні обмеження у стандартній формі. Вектор параметрів, які визначають оптимальний план, має бути однаковим для великої кількості обмежень. При цьому для коректного запису обмежень часто доводиться вводити додаткові змінні, що збільшують розмірність задачі і ускладнюють алгоритми її розв'язання.

Декомпозиційні методи полягають у розбитті вихідного завдання на підзавдання меншого розміру (кластери). Отримані підзадачі вирішуються окремо, а потім окремі розв'язки поєднуються. Розбиття проводиться за різними ознаками, наприклад, за аудиторіями, де проводяться заняття, окремими днями тижня, групами занять по чисельнику чи знаменнику тощо. Недоліком цієї групи методів є складність доведення оптимальності одержаного рішення. Рішення вихідної задачі складається послідовно з рішень підзадач, тому часто параметр оптимізації вихідної задачі в процесі рішення мають фіксовані значення. Загалом, за наявності великої кількості параметрів

оптимізації та обмежень у задачі можна отримати лише розв’язок, який близький до оптимального.

Методи евристичного пошуку призначені для вирішення задачі пошуку з обмеженнями. До цієї групи методів належить, наприклад, метод поширення обмежень. В основі відповідного підходу розглядається множина змінних (параметрів задачі), кожна з яких може приймати скінченну множину значень та множина обмежень (заборонені значення параметрів). В процесі розв'язання задачі поширення обмежень є таке присвоєння значень для всіх змінних, щоб виконувався набір обмежень. При цьому використовуються алгоритми перебору, методи віток та границь, алгоритм Ленд-Дойг тощо.

В процесі алгоритмічної реалізації загальний обсяг обчислень, необхідний для отримання розв’язку, включає у себе обчислювальні витрати на перебір варіантів та обчислювальні витрати на перевірку виконання обмежень. Використання глобальних обмежень спільно зі спеціальними алгоритмами перевірки та виконання (називаються інакше фільтруючими алгоритмами) дозволяє суттєво (на порядок) зменшити кількість обчислень, необхідних для знаходження розв’язку у завданнях великий розмірності [18].

До недоліків даного підходу можна віднести те, що оптимальність отриманих розкладів може бути не високою з точки зору наперед визначених критеріїв оптимальності, оскільки ці критерії не враховуються у процесі пошуку рішення.

До групи метаевристичних методів відносяться методи локального пошуку, що включають пошук із заборонами і стохастичний пошук, метод імітації відпалу, генетичні чи еволюційні алгоритми, метод мурашиних колоній тощо. Основною особливістю даної групи методів є можливість отримання кількох оптимальних чи суб-оптимальних рішень та використання «чорнових» рішень на початку пошуку. Пошук оптимального рішення здійснюється шляхом послідовного покращення кожного з попередніх рішень. Як правило, у таких алгоритмах передбачається можливість виходу в процесі

оптимізації з точок локальних мінімумів оптимізованої функції. Метаевристичні методи широко застосовуються в даний час для складання розкладу занять. Такі методи починають з одного або декількох початкових розв'язків і використовують стратегії пошуку, за допомогою яких намагаються уникнути локальних оптимумів. Ці алгоритми пошуку можуть забезпечувати високоякісні рішення, але часто мають значну обчислювальну вартість. Класичні методи здебільшого використовують ітераційну техніку неперервної оптимізації, тому можливе зациклення в локальному оптимумі [14].

Класифікація основних метаевристичних методів, які застосовуються при автоматичному складанні розкладів, наведена на рисунку 1.1.

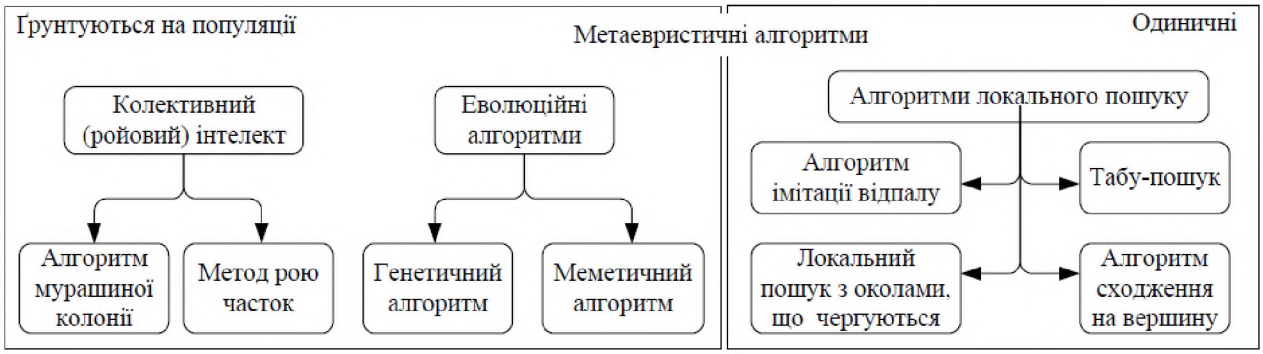

Рис.1.1. Класифікація метаеврестичних методів

Метод мурашиної колонії ґрунтується на здатності мурах знаходити найкоротші шляхи до їжі за допомогою виділення феромону. Недоліки методу: збіжність алгоритму гарантується, але час збіжності не визначено; результат роботи методу залежить від початкових параметрів пошуку, які підбираються експериментально; теоретичний аналіз значення початкових параметрів ускладнено, дослідження є більше експериментальними.

Всі розглянуті вище методи в своїй основі використовують ітераційну техніку для покращення результатів. Протягом однієї ітерації йде пошук розв'язку, який буде кращий за попередній. Якщо такий розв'язок знайдено, він стає поточним, і починається нова ітерація. Так триває, доки приріст

цільової функції не зменшиться практично до нуля або не виконається задана кількість ітерацій [46, 77, 82].

Генетичні алгоритми – це евристичні оптимізаційні методи, основну ідею яких запозичено з теорії еволюційного розвитку видів [83]. Основним механізмом еволюції є природний відбір, суть якого полягає в тому, що більш пристосовані особини мають більше шансів на виживання та розмноження. При цьому в процесі породження нащадків відбувається передавання генетичної інформації, нащадки успадковують від батьків деякі їх риси. Як зазначається в роботі [84], «генетичні алгоритми моделюють природні процеси: відбір, рекомбінацію, мутацію, міграцію, інверсію».

Відбір визначає, які істоти вибираються для виробництва нових нащадків. Стратегії відбору є складовими ГА і визначають придатних для схрещення істот. До генетичних операторів належать: оператор рекомбінації чи кросовера; оператор мутації, інверсії; оператор відбору.

Перевага цієї групи методів полягає в тому, що оптимізація проводиться відразу для деякої множини рішень, і можливий вибір найкращого розкладу з множини допустимих. До переваг генетичних алгоритмів можемо віднести універсальність їх структури для вирішення широкого кола задач, простота програмної реалізації, стійкість до потрапляння в точки локальних. Крім переваг, аналізовані методи мають недоліки. Наприклад, до недоліків генетичних алгоритмів можна віднести недоведеність збіжності у випадку до оптимального вирішення завдання, значний час роботи. Також зазначено, що у завданнях складання розкладу занять, які вирішуються за допомогою генетичних алгоритмів, на результат оптимізації впливає вибір оператора мутації. На відміну від генетичних алгоритмів, методи стохастичного пошуку в деяких випадках мають математичний доказ збіжності до оптимального вирішення задачі. Наприклад, для метода Pareto simulated annealing [80], для двокритеріальної задачі оптимізації доведено збіжність принаймні рішення до

множини Парето оптимальних рішень вихідного завдання. Також існує адаптивний алгоритм стохастичного пошуку типу simulated annealing.

Вступом до повноцінного використання генетичних алгоритмів можна вважати працю Fatma A. Omara та Mona M. Arafa [85]. Kamaljit Kaur, Amit Chhabra та Gurvinder Singh [86] запропонували новий генетичний алгоритм *Heuristics based Genetic* розкладу статичних завдань на однорідні паралельні системи. Цей алгоритм мінімізує час завершення завдання і збільшує пропускну здатність системи. Lawrence D. Davis [87] висвітлив основні принципи використання генетичних алгоритмів та кількість варіацій розміру популяції, у методах ініціалізації, у визначенні придатності, у виборі і стратегії заміни при кросинговері та мутації. В роботі [88] поєднано використання генетичного алгоритму (двофазного та монолітного) з комбінаторним «аукціоном» для ресурсного кейсу з ціллю організації ефективного використання різнотипних ресурсів між проектами. Дана колаборація евристичних алгоритмів та комбінаторних операцій є спробою реалізувати концепцію «критеріїв».

Алгоритми побудови розкладів на основі генетичного алгоритму описані у роботах [43, 74, 90]. В роботі [43] розроблено алгоритм побудови розкладу для дистанційного навчання на основі генетичного алгоритму, реалізованого в якості підсистеми для системи дистанційного навчання «Віртуальний Університет». Генетичний алгоритм для побудови розкладу також розглядається в роботі [90] для побудови розкладу іспитів. При цьому особлива увага приділяється послідовності іспитів. В роботі [15] запропоновано новий варіант алгоритму імітації відпалу, який називається FastSA, для автоматичної побудови розкладу іспитів у ЗВО. В FastSA кожен іспит, обраний для планування, переміщається (і такий рух оцінюється) лише в тому випадку, якщо цей іспит мав якісь прийняті рухи на попередньому відрізку температури.

В останній час доволі часто застосовуються також гібридні методи складання розкладу у вищих навчальних закладах. Наприклад, у роботі [91] автор пропонує метод генерації розкладу за допомогою комбінованого підходу методом підживлення бактерій та генетичного алгоритму. Даний підхід є доволі складний та ресурсозатратний. Запропоновано композиційний генетичний алгоритм, особливістю якого є: структуроване уявлення об'єктів (особи та хромосом) генетичного алгоритму; модифікація процедур схрещування та мутації стосовно структурованого представлення об'єктів генетичного алгоритму.

У публікації [92] досліджено ефективність паралельних генетичних алгоритмів для вирішення задачі складання оптимального розкладу занять. Причому розклад повинен відповідати певним вимогам. А саме, певні викладачі, групи та аудиторії не можуть брати участь в навчальному процесі у певні проміжки часу, не повинно бути вікон, для деяких занять можуть створюватись підгрупи або об’єднуватись у великі групи, викладачі та групи не повинні протягом дня пересуватись між заняттями на великі відстані, заняття, бажано, повинні починатись зранку. Провівши ряд досліджень, автор зазначає, що використання паралельних генетичних алгоритмів є ефективним рішенням для пошуку оптимального розв’язку для задачі створення розкладу занять. Для пошуку глобального оптимуму доцільно використовувати еволюційні алгоритми.

Описана вище класифікація методів явно неповна, але відбиває основні групи існуючих методів складання розкладу занять. При виборі способу розв'язку слід враховувати такі важливі характеристики.

Як було зазначено раніше, алгоритми перебору, наприклад метод гілок та границь, не здатні за прийнятний час вирішувати існуючі завдання складання оптимального навчального розкладу великої розмірності без використання евристик та застосування декомпозиції вихідного завдання. Якщо ж евристики або декомпозиція вихідної задачі використовується, то

ускладняється доказ повноти методу, що використовується. Метод евристичного пошуку, на відміну від розглянутих вище методів, в основному є неповними, хоча існують і винятки.

В силу зазначених недоліків має сенс розглянути можливість поєднання різних методів оптимізації, так звані гібридні алгоритми. Під гібридним алгоритмом розуміється алгоритм, у якому поєднуються кілька підходів до розв'язання задачі складання оптимального навчального розкладу [24]. Ефективність гібридного алгоритму може бути вищою від сумарної ефективності окремих алгоритмів, що входять до нього. Це можливо, якщо окремі алгоритми, що входять до гібридного, поєднуються таким чином, що недоліки деяких з них усуваються перевагами інших. Як приклади гібридних алгоритмів, що успішно застосовувалися для складання оптимальних навчальних розкладів, можна відзначити меметичний алгоритм, що є гібридом генетичних алгоритмів і методом локального пошуку.

1.3. Перманенти та їх властивості

В роботах [2, 5] пропонується новий підхід до складання розкладів, який можна віднести до методів комбінаторної оптимізації та в основі якого покладено процедуру перманентної декомпозиції. Тут суттєво використовуються перманенти та їх властивості. Причому певні модифікації класичного поняття перманента лягли в основу оригінальних алгоритмів формування розкладів, що запропоновані автором.

Нехай задана деяка матриця

$$A = \begin{pmatrix} \alpha_{11} & \alpha_{12} & \dots & \alpha_{1n} \\ \alpha_{21} & \alpha_{22} & \dots & \alpha_{2n} \\ \dots & \dots & \dots & \dots \\ \alpha_{m1} & \alpha_{m2} & \dots & \alpha_{mn} \end{pmatrix}, \tag{1.1}$$

причому $m \leq n$.

Перманентом матриці A називається число виду:

$$PerA = \sum_{\{j_1, j_2, \dots j_m\}} a_{1j_1} a_{2j_2} \dots a_{mj_m}, \quad (1.2)$$

де $(j_1, j_2, \dots, j_m)$ – впорядкована m–елементна вибірка з множини $\{1,2,\dots,n\}$.

Тобто перманент матриці – це сума усіх добутків її елементів, взятих по одному з кожного рядка і не більше ніж по одному з кожного стовпця. Бачимо, що перманент відрізняється від детермінанта лише додатніми знаками усіх добутків. Перманент квадратної матриці рівний сумі усіх добутків її елементів, взятих по одному з кожного рядка та кожного стовпця.

Розглянемо деякі очевидні властивості перманента:

1. При перестановках рядків та стовпчиків матриці її перманент не змінюється.

2. Перманент не змінюється при транспонуванні.

3. Перманент матриці є лінійною функцією її рядків:

$$Per\begin{pmatrix} a_1 \\ \dots \\ \alpha a_i^{'} + \beta a_i^{''} \\ \dots \\ a_m \end{pmatrix} = \alpha Per\begin{pmatrix} a_1 \\ \dots \\ a_i^{'} \\ \dots \\ a_m \end{pmatrix} + \beta Per\begin{pmatrix} a_1 \\ \dots \\ a_i^{''} \\ \dots \\ a_m \end{pmatrix}, \quad (1.3)$$

деa_1,a_2,…,a_m-рядки матриці.

На відміну від задачі знаходження визначника, задача знаходження перманента є NP-складною [1] навіть у випадку двійкової матриці.

Перманент можна обчислювати розкладом по рядку: $PerA = \sum_{j=1}^{n} a_{ij} PerA_{ij}$, де A_{ij}-матриця, що отримується з матриці A викреслюванням і-того рядка та j-того стовпчика.

У випадку квадратної матриці можемо розкладати і по стовпчику. Якщо, наприклад, Е-квадратна матриця розмірності *nxn*, що складається з одиниць,

то *PerE=n!*. Якщо I-одинична матриця, то $Per(E-I)=D_n$, де D_n-кількість безпорядків, тобто перестановок чисел *1,2,…,n* без нерухомих точок [2].

Очевидно, що при обрахунку перманента матриці розмірності mxn необхідно A_n^m-1 додавань та $A_n^m(m-1)$ добутків.

Зауважимо, що відомі більш ефективні способи обрахунку перамнента. Слід згадати, зокрема, алгоритм Г. Дж. Райзера, який ґрунтується на узагальненій формулі включень-виключень.

Нехай A – деяка скінченна множина, $A_1, A_2, \dots, A_n$ – її підмножини. Кожному елементу aмножини A поставлене у відповідність деяке число $\mu(a)$ – вага елемента. Вага деякої множини B визначається як сума ваг її елементів. Визначимо функцію $w()$ наступним чином:

$$w(0)=\mu(A),\ w(k)=\sum_{1\le i_1\le i_2\le\dots\le i_k\le n}\mu(A_{i_1}\cap A_{i_2}\cap\dots\cap A_{i_k}),\ k\le n. \quad (1.4)$$

Нехай $M(r)$ – сума ваг всіх таких елементів множини A, які належать рівно r різним підмножинам A_i. Тоді:

$$M(r)=w(r)-C_{r+1}^1 w(r+1)+C_{r+2}^2 w(r+2)+\dots+(-1)^{n-r}C_n^{n-r}w(n) \quad (1.5)$$

Нехай задана деяка матриця $A=(a_{ij})_{i=\overline{1,m},j=\overline{1,n}}$, X – множина рядків матриці,Y – множина стовпців.

Розглянемо множину функцій $U=\{f: X\to Y\}$.

Визначимо вагу функції:

$$\mu(f)=a_{1f(1)}a_{2f(2)}\dots a_{mf(m)} \quad (1.6)$$

Для випадку квадратної матриці:

$$PerA=\sum_{k=0}^{n}(-1)^k w(k)=\sum_{k=0}^{n}(-1)^k\sum_{|I|=n-k}S(I)=\sum_{I\subset Y}(-1)^{n-I}S(I) \quad (1.7)$$

Наприклад:

$$Per\begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{pmatrix} = (a_1 + b_1 + c_1)(a_2 + b_2 + c_2)(a_3 + b_3 + c_3) -$$

$$-(a_1 + b_1)(a_2 + b_2)(a_3 + b_3) - (a_1 + c_1)(a_2 + c_2)(a_3 + c_3) - (b_1 + c_1)(b_2 + c_2)(b_3 + c_3) + a_1a_2a_3 + b_1b_2b_3 + c_1c_2c_3 \quad (1.8)$$

Рекурсивна процедура розкладу перманента за рядком стала основним прототипом методів перманентної декомпозиції, що розглядаються в даній роботі.

Аналіз методів інформаційних технологій, які використовуються сьогодні для складання розкладів, показує, що кожен з них має свою оптимальну сферу застосування та має певні недоліки, що вказує на необхідність пошуку подальшого вдосконалення існуючих методів та розробки нових. Також виникає необхідність проаналізувати інформаційні системи, у яких ці методи реалізовані, та їх особливості.

1.4. Аналіз інформаційних систем для складання розкладів занять

Проведено аналіз інформаційних систем для складання розкладів занять для українського та міжнародного ринків програмного забезпечення.

Якщо розглянути ринок програмного забезпечення України, основними системами для автоматичного складання розкладу є:

– комплексна система розкладу університету UniTime;

– програма Ректор–ВНЗ;

– програма складання розкладу занять «НІКА»;

– інформаційна система Розклад Нова Школа.

UniTime – це комплексна система створення навчального розкладу [96]. Вона підтримує розробку розкладів, керування змінами в цих розкладах,

спільне використання аудиторій декількома групами. Ця система є розподіленою, що в свою чергу дозволяє кільком операторам зі складання розкладу університету об'єднувати зусилля для створення та внесення змін до розкладу, який відповідає їхнім різноманітним організаційним потребам, одночасно дозволяючи звести до мінімуму накладки в розкладі. Ця система розповсюджується безкоштовно за ліцензією з відкритим вихідним кодом.

Програма «Ректор ВНЗ» призначена для складання розкладу занять у ЗВО. Даний програмний продукт має чотири складових: «Списки», «Навантаження», «Розклад», «Заміни». Розділ «Списки» реалізовує функції для заповнення, редагування і друку різних списків (наприклад, груп, занять, аудиторій, викладачів). Для роботи з навчальними планами зі спеціальностями, навантаженнями викладачів, різноманітних звітів у цілому використовується розділ «Навантаження». «Розклад» реалізовує функціональні можливості складання розкладу відповідно до груп, викладачів та аудиторій. А у розділі «Заміни» вносяться дані щодо замін викладачів. Можна в будь-який момент переключатись між трьома режимами створення розкладу – ручному, автоматизованому та змішаному. У автоматичному режимі створення враховуються усі вимоги щодо створеного розкладу, а при складанні в ручному режимі програма пропонує варіанти розміщення занять для відповідних груп у певні аудиторії та слідкує, щоб не було великих проміжків між заняттями. Розклад є можливість зберегти у декількох форматах, що зручні для використання: Microsoft Word, Excel або HTML [73].

Програма «НІКА» повністю автоматизує процес складання розкладу занять в освітніх закладах різного статусу та профілю (школи, гімназії, коледжі, училища), а також враховує особливості навчальних планів та організації навчального процесу. Програма легка у розумінні та зручна у роботі. Результатом роботи програми є розклад занять, складений з мінімальною кількістю вікон для викладачів. Програма «НІКА» враховує [73]:

– наявність другої зміни, п'яти- або шестиденної форм навчання;

– особливості наповнення аудиторії;

– методичні дні та години, що небажані для роботи вчителів;

– граничні рівні навчального навантаження з урахуванням складності предметів.

ІС Розклад Нова Школа – розклад навчальних занять автоматизує процес складання розкладу, враховуючи обов'язкові, умовно обов'язкові та бажані умови, зокрема, облаштування аудиторій під окремі дисципліни, належність до одного чи різних навчальних корпусів, завантаженість викладачів, формування зведених груп студентів тощо [98]. Важливим інструментом системи «Розклад навчальних занять» є можливість планування порядку вивчення академічних предметів. Ця опція включає в себе графік щотижневого вивчення дисциплін і технологічну карту з послідовністю їх проходження.

Розклад має бути побудований таким чином, щоб забезпечити логічну послідовність вивчення тих чи інших дисциплін, – і система це враховує.

В загальному, ринок програмного забезпечення щодо систем автоматичного складання розкладу виглядає наступним чином: якщо проаналізувати закордонні розробки, які стосуються проблеми складання розкладів у дещо ширшому аспекті, то можемо виділити до двух сотень інформаційних систем (було оглянуто 196 ІС, станом на 01.01.2023), що представлені на ринку (мають відгуки та проіндексовані інформаційними порталами – експертами галузі: 20.11.2022 [116], 01.01.2023 [117], 24.01.2023 [118]). Деякі з оглянутих далі програм представлені також і на українському ринку програмного забезпечення [97].

Окремими параметрами класифікації по складності інформаційних систем можемо визначити такі: параметри *«глибоке календарне планування»* (далі *SM+*), що є демонстрацією більш складних процесів task scheduling в інформаційній системі (в контексті, безпосередньо, побудови розкладів) та параметр *«просте календарне планування»* (далі *SM(b)*, що є прикладом простих процесів task scheduling в інформаційній системі (побудова *простих*

розкладів, що не являються NP-повною задачею, *booking*, за класифікацією даних процесів).

До першого списку (*SM+*) відносяться такі інформаційні системи, як Acuity Scheduling, Booksy, eTermin, Journyx, Schedule It, 10to8, Bookwhen, evQueue, JS7 JobScheduler, Schedule.cc , SyncThemCalendars, 7shifts, Booxi, EZ Transaction, Kantata, ScheduleAnywhere, Teamdeck, AccountSight, Boulevard, EZnet Scheduler, KeyedIn, ScheduleFlex, Tes Timetable, AI Field Management, CalendarHero, Findmyshift, Kirona Solutions та низка інших [97, 116–118].

До другої групи (*SM(b)* можемо віднести: Synchroteam, TimeForge, TimeLog PSA, TimeLog PSA, Timezynk, Timely, TimeSimplicity, TimeTabler, TimeTap, TIMIFY, TutorShell, Timely, TimeSimplicity, TimeForge, SimplyBook.me, Visual Planning, SimplyMeet.me, Skedda, Voxiplan, Vyte, Sling, Snap Schedule, Wix, SnapAppointments, SocialSchedules, StaffScheduleCare, Wrike, Storrito [97, 116-118].

До першої групи належить значно більша кількість ІС, що вказує на тенденцію використання методів генерації розкладів значної складності (*SM+*) та важливості подальшого розвитку та оптимізації методів побудови розкладів загалом.

Розглянемо приклади найбільш поширених програм, які безпосередньо використовуються для складання розкладів занять закладів вищої освіти. Переважна кількість зі списку – *SM+* інформаційні системи:

Apptavi – ІС для розкладу, яке зосереджується на мінімізації повторюваної роботи вчителів. Особливістю цього програмного забезпечення є те, що воно розроблено вчителями для вчителів [99]. Недоліком є відносна складність налаштування.

Skolaris – це ІС для онлайн–розкладу, яке може створювати розклади будь-яких розмірів і форм [100]. Вона проста у використанні та найкраще підходить для вчителів, зацікавлених у творчому підході до розкладів.

Omniscol – це ІС для онлайн-розкладу, яка може створювати розклади для різноманітних налаштувань [101]. Інтерфейс дуже простий у використанні, що значно полегшить життя під час створення розкладів. Особливості: доступна безкоштовна пробна версія, модифікація у реальному часі, система управління обмеженнями.

Prime Timetable – одне з найкращих доступних програм для складання шкільних розкладів [102]. Вона може створювати автоматичні та ручні розклади на будь-якому типі пристрою. Достатньо розповсюджена в різних країнах світу.

SchoolBooking – програмне забезпечення, яке використовується для керування кімнатами, ресурсами та класами [103]. Це дуже поширене програмне забезпечення є одним із провідних у світі – тисячі розкладів щодня створюються на SchoolBooking по всьому світу. Дуже популярне і просте у використанні, можна використовувати для різноманітних налаштувань та функцій. Можна отримати доступ з будь-якого пристрою, який має доступ до Інтернету. За запропоновою вище класифікацією відноситься до SM(b) класу.

TimeTabler існує вже понад чотири десятиліття. Насправді, це було перше програмне забезпечення для розкладу, яке використовувалося на персональному комп'ютері. До недоліків можна віднести відсутність WEB-версії. Особливо цікава дана інформаційна система демонстрацією роботи розкладів у складі ERP-систем [104].

Edutimer є одним із найпростіших засобів для створення розкладів [155]. Це програмне забезпечення вимагає від користувача лише вводу даних та вимог.

Teachmint – це онлайн-програмне забезпечення, яке створює цифрові розклади [105]. Мета цього програмного забезпечення – допомогти школам ефективно керувати своїм часом, енергією та ресурсами.

Timelabs Professional – це високоякісна онлайн ІС-система, яка готова полегшити роботу вчителів [106]. Вона керує відвідуванням, відпустками,

нарахуванням заробітної плати та інший типовий функціонал. Варто відмітити, що дана ІС має функціонал *HR-MRP*, що, знову-таки, свідчить про залежність подібних систем від побудови розкладів.

Canva дозволяє розробляти та створювати розклади професійної якості. Можна створювати тижневі розклади за допомогою редактора шаблонів. Інструмент дозволяє публікувати, завантажувати та ділитися розкладами. Можна налаштувати, обрізати або використовувати фільтри, щоб налаштувати вбудовані шаблони розкладу. Відноситься до SM(b) класу інформаційних систем [107].

Free College Schedule Maker – це веб-програма, яка дозволяє безкоштовно створювати щотижневі розклади занять [108]. Розклади можна зберігати на комп'ютері. Можна імпортувати збережений розклад, якщо хочете змінити курси. За допомогою безкоштовного розкладника коледжу, доступне налаштування розкладу, змінивши день початку тижня, тривалість збільшення часу та тип годинника (12-годинний, або 24-годинний). Також можна налаштувати вигляд розкладу, увімкнувши/вимкнувши рамку, зменшивши висоту розкладу та відобразивши вихідні.

Schedule Builder – програма для планування, яку можна використовувати для безкоштовного створення розкладів онлайн. Додаток дозволяє створювати до п'яти щоденних або тижневих розкладів. Зберігається розклад як зображення або PDF-файл, забезпечено друк. Додаток підтримує дев'ять мов, включаючи англійську, французьку, шведську та інші. Відноситься до SM(b) класу інформаційних систем [109].

Adobe Spark – це безкоштовна веб-програма, за допомогою якої можна скласти розклад [110]. Дозволяє створювати розклади занять, бізнес-розклади або особисті розклади за допомогою програми онлайн-розкладу. Додаток дозволяє створювати індивідуальні розклади, вибираючи зображення, тексти та логотипи. Відноситься до *SM(b)* класу інформаційних систем, але, після інтеграції в Adobe Express, можемо очікувати зміни та тяжіння до SM+.

Visme – дизайнерський інструмент для створення індивідуальних розкладів онлайн. Програма дозволяє створювати професійно розроблені розклади з налаштованими макетами, темами та кольорами. Відноситься до SM(b) класу інформаційних систем [111].

Doodle – популярний онлайн-додаток для створення особистого чи професійного розкладу. Можна створювати місячні або тижневі розклади [112].

College Schedule Maker створено для створення розкладу занять [113]. Додаток дозволяє додати тему, час, тип курсу, місце та ім'я викладача. Можна встановити час збільшення планувальника на 30 хвилин або годину. Додаток також можна використовувати для планування занять у класі. Можна зберегти свій розклад як зображення або роздрукувати розклад. Онлайн-планувальник можна використовувати на будь-якому пристрої з підключенням до Інтернету. Можна використовувати програму як на ПК, так і на смартфоні. Відноситься до SM(b) класу інформаційних систем.

Coursicle – це програма для створення розкладів, яку можна використовувати для створення тижневих розкладів занять в Інтернеті [114]. Онлайн-програма має простий і зручний інтерфейс. Додаток дозволяє додавати коледж і шукати курси до тижневого планувальника.

Наведено вище лише інформаційні системи, що виділяються функціональними особливостями, в списку SM+ є й інші продукти, такі як *ASM Timetables, Omnify, IQ Session* та інші, але в них типовий функціонал, а їхні методи реалізації недоступні до аналізу *[115]*.

Проведений аналіз інформаційних систем з використанням методів інформаційної технології складання розкладу вказує на необхідність продовження оптимізації методів та алгоритмів, що стосуються їх, так як інформаційні системи, наведені у аналізі, схиляються до використання складних календарних обчислень, оперуючи великою кількістю даних (NP-повна задача).

Варто окремо зазначити, що після пандемії COVID-19, можемо спостерігати стрімкий розвиток інформаційних систем, направлених на організацію дистанційної роботи, яка базується саме на оптимально побудованих розкладах дистанційної роботи.

1.5. Висновки до першого розділу

Аналіз існуючих методів складання розкладів показує, що низка методів потребує вдосконалення та подальшого розвитку.

Потребують вдосконалення алгоритми комбінаторної оптимізації. Через значну обчислювальну складність алгоритми перебору, зокрема, метод гілок та границь, не здатні за прийнятний час вирішувати існуючі завдання складання оптимального навчального розкладу. А тому важливою задачею є розробка нових підходів до задач генерації комбінаторних об'єктів.

Застосування евристик та методів декомпозиції ускладнюють доведення оптимальності розв'язку. При використанні методу цілочисельного програмування виникає проблема коректного представлення обмежень у стандартному лінійному вигляді, що призводить до введення додаткових параметрів та зростання розмірності задачі.

Перспективним є підхід застосування гібридних алгоритмів, оскільки ефективність гібридного алгоритму може бути вищою від сумарної ефективності алгоритмів, що входять до нього. Це можливо за рахунок усунення недоліків окремих методів, своєрідного синергетичного ефекту.

Застосування генетичних алгоритмів до задач складання розкладів призводить до виникнення проблем зупинки генетичного алгоритму, складності алгоритму та швидкості досягнення результату в рамках генетичного підходу, виникає проблема коректного вибору генетичних операторів та необхідність подальшого їх вдосконалення в задачах складання

розкладів занять. Аналіз сучасних програмних засобів, що використовуються для задач складання розкладів дозволяє виділити низку проблем:

– потребують вдосконалення системи організації даних, ефективність та правильність їх опрацювання;

– потребують вдосконалення модулі глибокого аналізу даних для задач складання розкладів;

– вимагає вдосконалення аналітика великих даних у задач календарного планування, основні функціональні вимоги до планування завдань.

На основі проведеного аналізу визначено наукове завдання, яке полягає у підвищенні ефективності методів формування розкладів занять шляхом розробки інформаційної технології, яка базується на використанні алгоритмів перманентної декомпозиції та включає такі підзадачі:

– розробка нових критерів для перевірки допустимості матриць розкладів та можливості їх оптимізації в межах конфігураційного підходу, що дозволить вдосконалити існуючі процедури аналізу даних в інформаційних системах для автоматизованого створення розкладів;

– розробка нових алгоритмів генерації комбінаторних об'єктів, що ґрунтуються на властивостях модифікованих перманент матриць інцидентності;

– розробка нових методів до розв'язання задач складання розкладів у межах перманентного підходу на основі систем різних представників конфігурацій шляхом використання спеціальних адитивно-диз'юнктивних форм.

1.6. Постановка задачі

Аналіз існуючих методів складання розкладів показує, що низка методів потребує вдосконалення та подальшого розвитку. Через значну обчислювальну складність алгоритми перебору, зокрема метод гілок та

границь, не здатні за прийнятний час вирішувати існуючі завдання складання оптимального навчального розкладу. А тому важливою задачею є розробка нових підходів до задач генерації комбінаторних об'єктів. Застосування евристичних підходів та методів декомпозиції ускладнюють доведення оптимальності розв'язку.

При використанні методу цілочисельного програмування виникає проблема коректного представлення обмежень у стандартному лінійному вигляді, що призводить до введення додаткових параметрів та зростання розмірності задачі. Перспективним є підхід застосування гібридних алгоритмів, оскільки ефективність гібридного алгоритму може бути вищою від сумарної ефективності алгоритмів, що входять до нього. Це можливо за рахунок усунення недоліків окремих методів, своєрідного синергетичного ефекту. Застосування генетичних алгоритмів до задач складання розкладів призводить до виникнення проблем зупинки генетичного алгоритму, складності алгоритму та швидкості досягнення результату в рамках генетичного підходу, виникає проблема коректного вибору генетичних операторів та необхідність подальшого їх вдосконалення в задачах складання розкладів занять.

Аналіз сучасних програмних засобів, що використовуються для задач складання розкладів дозволяє виділити низку проблем:

– потребують вдосконалення системи організації даних, ефективність та правильність їх опрацювання;

– потребують вдосконалення модулі глибокого аналізу даних для задач складання розкладів;

– вимагає вдосконалення аналітика великих даних у задач календарного планування, основні функціональні вимоги до планування завдань.

На основі проведеного аналізу визначено наукове завдання, яке полягає у підвищенні ефективності методів формування розкладів занять шляхом

розробки інформаційної технології, яка базується на використанні алгоритмів перманентної декомпозиції та включає такі підзадачі:

– розробка нових критеріїв для перевірки допустимості матриць розкладів та можливості їх оптимізації в межах конфігураційного підходу, що дозволить вдосконалити існуючі процедури аналізу даних в інформаційних системах для автоматизованого створення розкладів;

– розробка нових алгоритмів генерації комбінаторних об'єктів, що ґрунтуються на властивостях модифікованих перманент матриць інцидентності;

– розробка нових методів до розв'язання задач складання розкладів у межах перманентного підходу на основі систем різних представників конфігурацій шляхом використання спеціальних адитивно-диз'юнктивних форм.

РОЗДІЛ 2
ОСНОВИ ІТ СКЛАДАННЯ РОЗКЛАДІВ ТА МЕТОД ЗАСТОСУВАННЯ КРИТЕРІЇВ ДЛЯ ПЕРЕВІРКИ ДОПУСТИМОСТІ МАТРИЦЬ РОЗКЛАДІВ

2.1. Основи інформаційної технології складання розкладів

В процесі складання розкладів виникають задачі, які характеризуються багатокритеріальністю, важкоформалізованістю, значною обчислювальною складністю. Розв'язання їх класичними методами часто вимагає залучення значних обчислювальних ресурсів та адаптації алгоритмів із врахуванням особливостей кожної конкретної задачі. Тому виникає необхідність у розробці інформаційної технології, яка дозволила б підвищити ефективність існуючих технологій складання розкладів шляхом розробки нових методів аналізу вхідних даних, пріоритизації вимог та побажань стейкхолдерів і нових швидких алгоритмів генерації комбінаторних об'єктів.

Задачі складання розкладів належать до широкого класу задач комбінаторної оптимізації, спрямованих на знаходження оптимальної відповідності зі скінченної множини об'єктів.

При аналізі багатьох задач складання розкладів, очевидно, слід враховувати специфіку вимог до розкладу у кожному конкретному випадку. Розклад занять закладу вищої освіти чи відповідного його підрозділу відіграє фундаментальну роль при організації навчального процесу [3] та має ряд принципових особливостей. Тут необхідно враховувати три головних аспекти навчального процесу: організаційний, психолого-педагогічний та гігієнічний. Між окремими одиницями планування (заняттями) при складанні розкладу важко виявити змістовні взаємозв'язки в рамках навчального тижня, в той час як для тем предметів – основних одиниць при розробці тематичного плану – такі взаємозв'язки відіграють важливу роль.

Для розкладу занять часто не вдається визначити який-небудь домінуючий критерій оцінки. В цьому контексті йде мова про важкоформалізованість. Це пояснюється тим, що організаційно-педагогічні вимоги до розкладу занять зазвичай розглядаються стосовно однієї навчальної групи. Однак, розклад однієї групи не є незалежною частиною розкладу занять, оскільки різні групи в розкладі претендують на одні і ті ж ресурси: навчальні аудиторії, час викладачів і т.д. Крім того, в розкладі занять необхідно враховувати індивідуальні побажання педагогічних працівників, які важко представити у ролі визначених закономірностей.

Все це потребує специфічних підходів до складання розкладів занять окремих груп, які відрізняються від методів планування навчального процесу на рівні спеціальності.

Інформаційна технологія, що пропонується, може бути розглянута як сукупність декількох методів та підходів:

– методу аналізу вхідних даних, що дозволяє здійснювати перевірку допустимості матриць розкладів та можливості їх динамічної оптимізації, в основі якого покладено поняття конфігурації в матриці розкладів;

– методу перманентної декомпозиції, який дозволяє здійснювати генерацію комбінаторних об’єктів згідно модифікованих перманент матриць інцидентності;

– методу генерації матриць розкладів, що ґрунтується на формуванні спеціальної алгебри адитивно-диз’юнктивних форм, яка дозволяє на основі систем різних представників стовпців матриць розкладів синтезувати усі варіанти розкладів у відповідності до заданих додаткових вимог та відрізняються більшою швидкодією у порівнянні з відомими підходами;

– моделювання роботи експерта в процесі формування розкладу із врахуванням пріоритизації вимог стейкхолдерів.

Якщо розглянути інформаційну технологію, що пропонується, у порівнянні з існуючими технологіями (рисунок.2.1.), то слід зауважити, що

низка її складових може працювати у поєднанні з ними. Так, наприклад, критеріальний метод аналізу вхідних даних може бути використаний і в інформаційних технологіях, що ґрунтуються на генетичних алгоритмах. Генетичні алгоритми дуже широко використовуються сьогодні для розв'язання задач складання розкладів. Низка відомих методів стосується генерації комбінаторних об'єктів, різноманітних алгоритмів перебору.

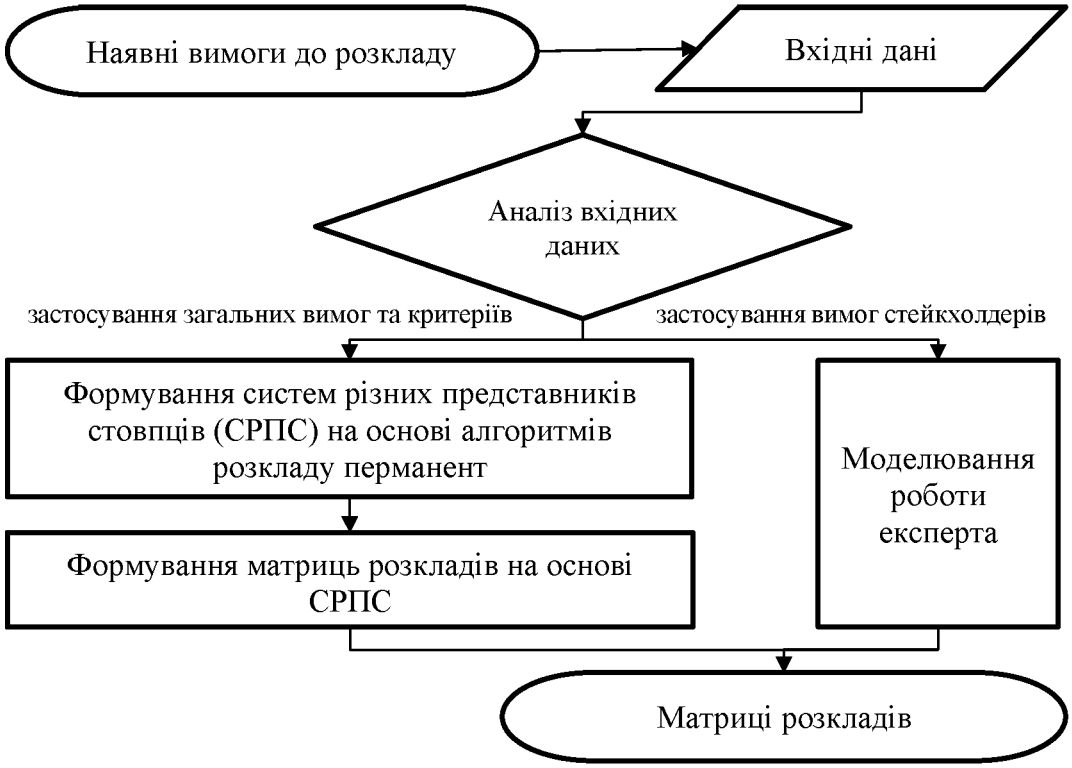

Рис.2.1. Схема застосування інформаційної технології складання розкладів за принципами перманентної декомпозиці

В роботі запропоновані специфічні відношення порядку для матриць розкладів в межах алгоритмів генерації комбінаторних об'єктів, що ґрунтуються на відношеннях порядку. Метод перманентної декомпозиції також генерує результат, який може бути використаний генетичним алгоритмом. Критеріальний метод аналізу вхідних даних базується на поняттях конфігурацій і також може бути використаний як самостійний метод

генерації матриць розкладів у поєднанні з певними евристичними алгоритмами.

Таким чином, розглянуто специфіку задач складання розкладів для закладів вищої освіти та основи інформаційної технології складання розкладів, яка пропонується та дозволить розв'язати низку існуючих проблем. Очевидно, відповідна інформаційна технологія вимагає подальшої деталізації.

2.2. Структуризація вимог та критеріїв для створення розкладів

Як бачимо з попереднього аналізу, формулювання і обґрунтування вимог до складання розкладу занять має фундаментальне значення. Однак, якщо розглянути вимоги до складання розкладів не ізольованих для конкретної групи, а стосовно закладу вищої освіти, то виникає необхідність у більш глибшому аналізі, який ґрунтується на системному підході. В цьому випадку вимоги до розкладу занять трактуються не як проста сукупність, а як система взаємозв'язаних умов зі всіма протиставленнями. Для реалізації такого підходу необхідно перш за все виявити всю множину вимог до розкладу.

Ці формулювання визначають тільки зміст вимог і конкретизуються для кожного окремого випадку.

Ці вимоги служать базою для розробки оптимального розкладу занять навчального закладу, так як вони достатньо повно відображають три головних аспекти навчального процесу: організаційного, психолого-педагогічного і гігієнічного. Кожне заняття характеризується такими вимогами:

– V1 – група, якій проводиться заняття;

– V2 – викладач, який проводить заняття;

– V3 – навчальний предмет, по якому проводиться заняття;

– V4 – аудиторія, в якій проводиться заняття.

Уважне вивчення інструкційно-методичних матеріалів органів освіти, результатів медико-педагогічних досліджень, анкетування керівних і

педагогічних працівників показало, що до розкладу занять у навчальному закладі пред'явлені сформовані вимоги.

Для початку складання розкладу занять повинні бути відомі, у крайньому випадку, перші чотири атрибути (вимоги: V1-V4, рисунок. 2.2.).

Рис.2.2. Вимоги до розкладу ЗВО в залежності від атрибуту (V1–V4)

Проведена вище класифікація вимог до розкладу занять в залежності від зв'язаних атрибутів занять – важлива, але не достатня умова побудови ефективної методики складання розкладів. Необхідно розглянути суттєву властивість вимог до розкладу, покладаючись на вивченні всієї сукупності вимог як цілісної системи. Такою властивістю є внутрішнє протиставлення системи вимог до розкладу.

Якщо врахувати можливість появи нових вимог до розкладу в процесі його формування, то всі атрибути (V) можемо подати у вигляді матриці, що визначатиме умови:

$$\begin{bmatrix} V_{1\,1} & \cdots & V_{1\,N} \\ \vdots & \ddots & \vdots \\ V_{M\,1} & \cdots & V_{M\,n} \end{bmatrix} \tag{2.1}$$

де $V_{i\,j}$–це відповідно, j-та вимога i –тої групи вимог (приклад–рисунок.2.2.), $i = \overline{1,M}, j = \overline{1,N}$.

Дві вимоги вважаються протиставними, якщо виконання однієї з них заважає (або робить неможливим) виконанню другої.

Вивчення практики складання розкладу занять показує, що виконати всі вимоги до розкладу неможливо у силу їх протиставності, тому процес складання розкладу може бути охарактеризований як пошук оптимального компромісу між різноманітними вимогами. Схема пошуку оптимального компромісу повинна бути важливою частиною методики складання розкладу занять. В основі класичного підходу до побудови розкладу занять лежить теорія розкладів, що широко використовується при організації роботи підприємств. Також до даної проблематики залучений і розділ динамічного програмування у теорії управління та теорії обчислювальних систем.

Якщо для задачі формування розкладу занять використовувати формалізм теорії розкладів, то в якості завдань слід розглядати заняття викладача, а в якості обслуговуючих пристроїв, машин – навчальні аудиторії,

де відбуваються заняття. При цьому слід враховувати фактор місткості аудиторій (адже в в одній аудиторії можуть проходити заняття з декількома групами одночасно, наприклад, лекції). Тобто обслуговуючий пристрій матиме додаткове обмеження на клас завдань, які він може обслуговувати.

Модель часу в даному випадку логічно розглядати як дискретну, якщо за одиницю дискретизації часового інтервалу прийняти стандартну тривалість пари. При цьому слід враховувати, що кожного тижня множина занять може змінюватись (зокрема, 1 тиждень / 2 тиждень).

Для того, щоб впорядкувати використання евристик, необхідно ввести відношення часткового порядку як до множини завдань, так і множини часових інтервалів (наприклад, можна вважати, що перша пара будь-якого дня тижня більш краща, ніж друга пара будь-якого дня тижня, оскільки тоді в процесі заповнення таблиці розкладу пари виставлятимуться щільніше – друга пара залишається незаповненою). Оператори часткового порядку являють собою довільну функцію, що впорядковує задані множини у відповідності з обраним критерієм.

В якості вихідної інформації можна розглядати тривимірну матрицю (час, об'єкти, аудиторії). Під об'єктами тут розуміються навчальні групи та викладачі, для яких будується розклад.

Важливим при складанні розкладу є поняття критерію оптимальності.

Однак, це досить складне питання, оскільки побудова універсального критерію ускладняється наявністю слабкоформалізованих та невимірюваних параметрів, протиріччями між різними учасниками процесу та стейкхолдерами. Для навчальних груп і викладачів, наприклад, є чисто суб'єктивне уявлення про зручний розклад. Тому в якості такого універсального критерію можна взяти щільність заповнення заняттями шарів тривимірної матриці як відношення кількості заповнених позицій до кількості усіх елементів.

Практично на будь-які елементи системи можуть накладатися обмеження (критерії: К1-К3, рисунок. 2.3.). Наприклад, всі заняття для будь-якої аудиторії повинні починатися не раніше заданого часу.

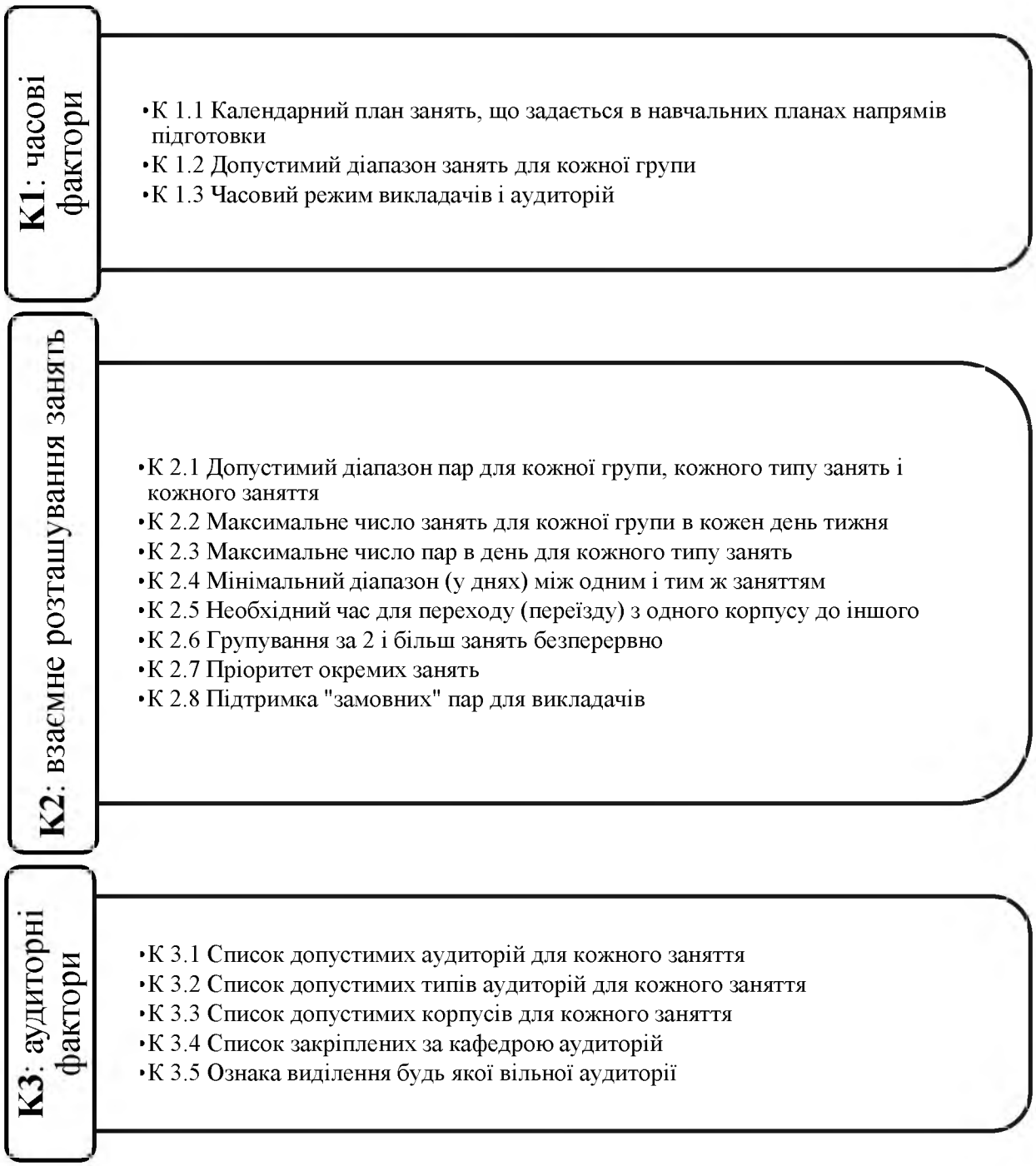

Рис.2.3. Додаткові критерії формування розкладу (К1-К3)

Наступним важливим питанням є способи кількісної оцінки різноманітних варіантів розкладу.

Одним із найпростіших способів кількісної оцінки розкладу занять є спосіб експертних оцінок, який широко використовується на практиці.

Але досвід показує, що цей метод ефективний для аналізу невеликої кількості критеріїв, які мають змістовну інтерпретацію. У випадку аналізу розкладу занять експерти відчувають значні ускладнення з оцінкою розкладу в цілому, хоча легко справляються з оцінкою тих чи інших його локальних частин. Звертання до експертів має ще один суттєвий недолік – забирає багато часу і засобів (підготовка вихідних матеріалів, обробка результатів). Крім того, число експертів, які знають дуже добре специфіку конкретного навчального закладу, зазвичай обмежено, тому важко розраховувати на отримання достатньо об'єктивних оцінок при невеликій кількості експертів.

Для кількісної оцінки розкладів можемо запропонувати підхід, суть якого полягає у наступному. Позначимо всю множину занять, які необхідні призначити до процесу складання розкладу занять через N. Підмножину занять, при призначенні яких представляється вимога j, позначимо Z_j.

Відмітимо, що $Z_j \subseteq N$, де $j = 1,2..,J$; J – загальна кількість вимог до розкладу. Підмножина занять, для яких вимога j дійсно виконана в розкладі занять, позначимо її через $\overline{Z_j}$. Ясно, що вимога може бути виконаною тільки для тих занять, для яких ця вимога була представлена, тобто виконується співвідношення: $\overline{Z_j} \subseteq Z_j \subseteq N$для будь–якого j.

Степенем виконання вимоги j в розкладі занять назвемо величину $S_j = \left|\frac{\overline{Z_j}}{Z_j}\right|$, де $\left|\overline{Z_j}\right|$ – кількість елементів множини $\overline{Z_j}$; $\left|Z_j\right|$ – кількість елементів множини Z_j. В якості кількісної оцінки розкладу занять приймається величина:

$$F = \sum_{j=1}^{J} K_j \cdot S_j. \qquad (2.2)$$

Вираз для оцінки F містить коефіцієнти $K_j (j = 1,2..,J)$. Конкретні числові значення цих коефіцієнтів підбираються експериментально таким чином, щоб вони відображали місце кожної j–вимоги.

Отже, деталізовано основні вимоги та критерії до розкладів занять ЗВО, запропоновано кількісні оцінки якості розкладів, що дозволяє розробляти гнучкі інформаційні технології з максимальним врахуванням вимог. З врахуванням складності та багатокритеріальності відповідних задач виникає необхідність у подальшому аналізі та розвитку методів, які можуть застосовуватись для їх розв'язку, зокрема, еволюційних алгоритмів.

2.3. Формування конфігурацій матриць розкладів з використанням генетичних алгоритмів

Як вже відзначалось вище, одним з найбільш відомих підходів до розв'язання задач теорії розкладів, зокрема, складання розкладу занять, зустрічей тощо є генетичні алгоритми. Генетичний алгоритм – це проста математична модель еволюції в природі, у якій зберігається біологічна термінологія у спрощеному вигляді. Як зазначає в роботі [148] О.С. Бичков, «генетичні алгоритми моделюють природні процеси: відбір, рекомбінацію, мутацію, міграцію, інверсію». Розглянемо, як формалізуються основні біологічні поняття:

– генотип – закодована бітовим рядком хромосома;

– фенотип – інформація, закодована у хромосомі;

– хромосома – бітовий рядок;

– істота – набір хромосом;

– ген – область (суміжні біти) у хромосомі, параметри задачі;

– алель – значення гена в хромосомі;

– придатність – значущість фенотипу;

– епістасис – сильна взаємодія між різними генами;

– алфавіт – мова кодування фенотипу;

– кодування – формальний запис фенотипу;

– популяція – масив генотипів і фенотипів.

В математичному аспекті генетичні алгоритми – це стохастичні пошукові методи, які імітують природну біологічну еволюцію. Генетичні алгоритми діють на популяції потенційних розв'язків із застосуванням принципу природного відбору для вироблення щонайкращих наближень до розв'язку. У кожному поколінні нова популяція (множина наближених розв'язків) створюється шляхом відбору істот згідно з рівнем їхньої придатності та з використанням операторів, «запозичених» у природної генетики. Цей процес детермінує еволюцію популяцій істот, краще пристосованих, ніж ті, з яких вони були створені.

В цілому, завдання складання розкладу навчальних занять можна сформулювати наступним чином: для заданого набору навчальних аудиторій (в даному випадку під навчальною аудиторією розуміється широке коло приміщень, в яких проводяться навчальні заняття (від комп'ютерної аудиторії до спортивного залу)) і заданого набору часових інтервалів (тобто, по суті, уроків або навчальних пар) побудувати такий розподіл навчальних занять для всіх об'єктів (викладачі та навчальні групи), для якого вибраний критерій оптимальності є найкращим.

Для складання розкладів лекцій та іспитів у низці університетів успішно застосовувались генетичні алгоритми. В якості функції відбору вибиралась функція, яка відкидала розклади, які потребують складання декількох іспитів у день або читання лекцій одночасно в декількох місцях, і, водночас, в процесі відбору відбирала варіанти з компактним розміщенням занять, рівномірним розподілом навантаження для студентів.

Дослідження роботи генетичних алгоритмів на тестових прикладах у випадках, коли найкращі розклади були відомі, показала, що результати генетичних алгоритмів відрізнялися від оптимальних лише на частки відсотка,

що практично підтверджує їх ефективність. Відбір визначає, які істоти вибираються для виробництва нових нащадків. Стратегії відбору є складовими ГА і визначають придатних для схрещення істот.

Для застосування генетичних алгоритмів при складанні розкладу занять фундаментальне значення має вибір структур даних. Наприклад, хромосому можна розглянути у вигляді матриці R(m,n), де кожен рядок відповідає викладачу, а кожен стовпець – годині або парі; елементами матриці є аудиторії. Можливо також подання вихідних даних у вигляді тривимірної матриці R(m, n, 2), де елементи 1–го шару матриці – це аудиторії, елементи другого шару – заняття.

До генетичних операторів належать:

– оператор рекомбінації чи кросовера;

– оператор мутації, інверсії;

– оператор відбору.

Генетичні оператори необхідні, щоб застосувати принципи спадковості та мінливості до віртуальної популяції. Вони мають таку властивість, як імовірність використання: описувані оператори не обов'язково застосовуються до всіх схрещуваних істот, що вносить додатковий елемент невизначеності в процес пошуку розв'язків. У даному випадку невизначеність є не негативним чинником, а своєрідним степенем свободи роботи генетичного алгоритму.

Для складання розкладу занять передбачається розробка наступних генетичних операторів [7]:

- мутація k–го порядку (цей оператор виконує наступні дії: з одного і того ж рядка матриці потрібно виділити дві суміжні послідовності розміром k елементів і поміняти їх місцями);

- мутація днів (даний оператор практично ідентичний до попереднього, за винятком того, що замість рядків матриці R він виконується над стовпцями, що представляють собою години, що відносяться до різних днів);

- кросоверу (схрещування).

Розглянемо оператор кросовера детальніше. Нехай є дві матриці хромосом R1 та R2. Нехай задана деяка глобальна функція оптимальності отриманого розкладу, а також деякі локальні функції, що відповідають, наприклад, окремій групі, окремому викладачу або окремій аудиторії. Даний оператор сортує рядки першої матриці в порядку зменшення значення локальної функції оптимальності, що відповідає викладачу. Після цього відбувається так званий процес схрещування, коли формується нова матриця хромосом. Перші (кращі b рядків) в ній беруться з відсортованої матриці R1, а залишилися m–b рядків беруться від другого батька – матриці R2. Значення b задається заздалегідь при формуванні локальних функцій оптимальності.

Зауважимо, що наведені вище приклади генетичних операторів стосуються, швидше, одного з оптимізаційних параметрів-викладачів. Якщо ж побудувати узагальнені генетичні оператори, у яких параметрами виступають одночасно всі групи, викладачі, аудиторії і т. д., то параметрів виходить занадто багато і проблематичним буде навіть побудувати якісь коректні генетичні оператори, не говорячи вже про обґрунтування збіжності такого алгоритму. Для успішної, ефективної і швидкої оптимізації необхідно, щоб параметрів у оптимізованої функції було якомога менше, в силу загальної стохастичності. У таких складних умовах найкращий вихід – це саме гібридний алгоритм. На першому етапі будуються якісь варіанти розкладу, наближені до оптимального. Наступним етапом проходить обґрунтування розкладу звичайним комбінаторним методом із використанням посилених критеріїв. Наприклад, можна застосувати алгоритми комбінаторної оптимізації, що деталізовані нижче. На останньому етапі може бути залучений експерт, який зможе зробити якусь корекцію, перестановку занять. Процес формування розкладу з пріоритетами, на цьому етапі, більше нагадує методику складання розкладу диспетчером-методистом: викладачі ранжуються за навантаженням (першими розподіляються заняття найбільш завантажених викладачів), враховуються ще якісь додаткові умови (рисунок 2.4.).

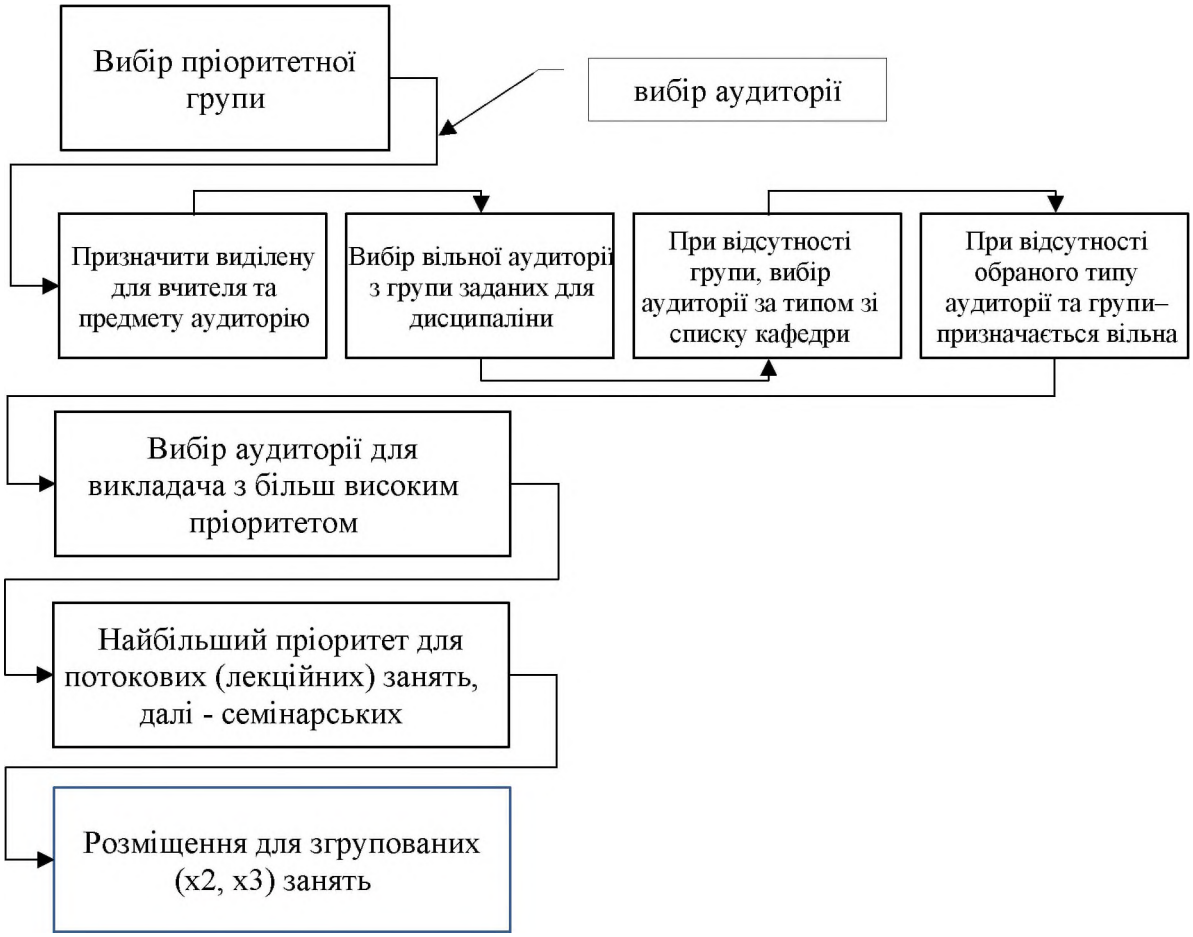

Рис.2.4. Процес формування розкладу з пріоритетами

Розглянуто структуру та особливості генетичних алгоритмів і запропоновано можливі генетичні оператори та стратегії відбору для задач складання розкладів занять.

В процесі роботи генетичного алгоритму виникає, зокрема, проблема аналізу отриманих матриць розкладів на предмет можливості їх подальшої оптимізації. Адже в процесі відбору може бути відкинутою матриця розкладу, яка представлена в неоптимізованому вигляді, однак легко може бути оптимізована шляхом перестановки елементів стовпців. А тому виникає необхідність у розробці більш глибоких методів аналізу матриць розкладів.

2.4. Критеріальний метод для задач складання розкладів з використанням конфігураційної структури матриць розкладів

В процесі генерації матриць розкладів часто виникають ситуації, коли матриця може бути недопустимою або її взагалі неможливо перетворити так, щоб вона задовольняла відповідні вимоги. А тому важливо мати якісь критерії, які могли б виявляти такі ситуації перед тим, як застосовуються відповідні перетворення. Відповідні властивості та критерії розглядаються в даному розділі.

Вище розглянуті вимоги, яким повинен відповідати розклад занять, зокрема: студенти не повинні мати «вікон», викладачі також не повинні мати «вікон», кількість робочих днів викладача мусить бути мінімальною, побажання викладачів варто враховувати максимально тощо [1].

Очевидно, що оптимальний у певному значені розклад – оптимальний для кожного дня навчального тижня. Тому обмежимося розглядом одного робочого дня.

Якщо матриці розкладу повністю заповнені ненульовими елементами (номерами викладачів), то автоматично забезпечується виконання очевидної умови не наявності і «вікон у студентів», кількість пар на день для кожної навчальної групи повинна дорівнювати 3.

2.4.1. Властивості матриць розкладу, що містять лише «ненульові» елементи

Розглянемо згаданий вище конфігураційний підхід та наведемо основні результати, включаючи і випадок частково заповнених матриць денного розкладу [1].

Кожному викладачу поставимо у відповідність деяке натуральне число (ідентифікатор, вага, тощо). Тоді матриця денного розкладу являє собою матрицю розмірності $3 \times n$, де n – кількість груп (підгруп), для яких

складається розклад (тут розглядаємо найпростіший випадок трьох пар на день). Будемо називати таку матрицю матрицею денного розкладу і позначати R. Аналогічно можемо ввести поняття матриці тижневого, місячного розкладу тощо. Зауважимо, що в такій інтерпретації тут не враховуємо інформацію про аудиторії чи конкретні назви дисциплін в розкладі – якщо коректно задана така матриця, то на основі неї можна вже побудувати розклад з конкретними дисциплінами – це окреме завдання.

Тернарною конфігурацією розкладу, утвореною елементом матриці розкладу d, будемо називати множину елементів $\{a_{ij}, a_{kl}, a_{mp}\}$ матриці R, таких, що $a_{ij} = a_{kl} = a_{mp} = d, i,k,m \in \{1,2,3\}, 1 \leq j,l,p \leq n$.

Аналогічно можемо ввести поняття бінарних та унарних конфігурацій (бінарна конфігурація не може бути елементом тернарної).

Спочатку обмежимось розглядом ситуації, коли в матриці денного розкладу немає потоків. Тоді вона має вигляд сукупності тернарних, бінарних, унарних конфігурацій та нулів (нуль означає, що в групи немає ніякої пари).

Матрицю розкладу будемо називати допустимою, якщо в рядку немає двох однакових елементів. Тоді для тернарної конфігурації виконується умова $i \neq k \neq m$ (викладач не може проводити кілька пар одночасно). Така ж умова є доцільною і для бінарної конфігурації.

Очевидно, що набір конфігурацій повинен бути таким, щоб матрицю розкладу взагалі можна було утворити (в одній групі повинно бути не більше 3 пар на день).

Відзначимо, що початковий процес формування конфігурацій може бути або автоматичним з міркувань мінімальної кількості робочих днів викладача, або бути результатом домовленості з викладачем (наприклад, деяким викладачам важко проводити три пари за один день, тоді можна утворити відповідні бінарні чи навіть унарні конфігурації, якщо такі усіх влаштовують). За такого підходу вирішується проблема мінімізації «вікон» викладачів вже на

початковому етапі розв'язку задачі: матриця розкладу представляє набір вже оптимізованих (таких, що не мають вікон) конфігурацій.

Розглянемо спочатку базові вимоги, яким має відповідати матриця розкладу:

K_1 – викладач не має «вікон», тобто якщо для будь-якої бінарної конфігурації $\{a_{ij}, a_{kl}\}$ виконується умова: $|i-k|=1$;

K_2 – студенти не мають «вікон», тобто в матриці денного розкладу немає стовпця, що містить два ненульові елементи та нуль в другому рядку.

Зрозуміло, що різні матриці розкладів можна утворювати шляхом довільної перестановки елементів стовпців (при цьому, зазвичай, можуть виникати недопустимі матриці).

Нехай маємо довільну матрицю денного розкладу, що не містить нульових елементів. Розглянемо декілька властивостей матриць розкладів, які стосуються їх допустимості виходячи з конфігураційної структури.

Нехай маємо деяку множину елементів $\Omega = \{x_1, x_2, \dots, x_n\}$. Нехай $(R_1, R_2, \dots, R_m)$ деяка впорядкована сукупність підмножин Ω. Системою різних представників (СРП) даної сукупності будемо називати вектор виду: $(a_1, a_2, \dots, a_n)$ такий, що $a_1 \in R_1, a_2 \in R_2, \dots, a_n \in R_n, a_i \neq a_j, i \neq j$.

Матрицею інцидентності СРП будемо будувати називати матрицю виду:

$$A = \begin{array}{c} \\ R_1 \\ R_2 \\ \dots \\ R_m \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & \dots & x_n \end{array} \\ \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \end{array}, \tag{2.3}$$

$$\alpha_{ij} = \begin{cases} 1, & \text{коли} \quad x_i \in R_j \\ 0, & \text{в противному випадку} \end{cases} \tag{2.4}$$

Елементи $x_1, x_2, \dots, x_n$ будемо називати ідентифікаторами стовпців матриці інцидентності.

Розглянемо фрагмент типового розкладу занять в Рівненському коледжі НУБіП України (рисунок 2.5.).

41-ІП	42-ІП	43-ІП
Організація комп. мереж *Бабич*	**Диф.рівняння та матем.логіка** *Петрівська*	**Вебдизайн та основи frontend** *Ушаков*
Основи економіки та IT бізнесу *Бондарчук*	**Вебдизайн та основи frontend** *Ушаков*	**Диф.рівняння та матем.логіка** *Петрівська*
Диф.рівняння та матем.логіка *Петрівська*	**Основи економіки та IT бізнесу** *Бондарчук*	**Організація комп. мереж** *Бабич*

Рис. 2.5. Фрагмент розкладу коледжу

Присвоюючи викладачам ідентифікатори 1 – Бабич, 2 – Петрівська, 3 – Ушаков, 4 – Бондарчук) отримуємо матрицю розкладу:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 3 & 2 \\ 2 & 4 & 1 \end{pmatrix} \tag{2.5}$$

Тоді матриця інцидентності матиме вигляд:

$$\begin{matrix} \\ 41-\text{ІП} \\ 42-\text{ІП} \\ 43-\text{ІП} \end{matrix} \begin{matrix} \begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix} \\ \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix} \end{matrix} \tag{2.6}$$

Очевидною є наступна властивість.

Властивість 2.0. Система різних представників існує тоді і тільки тоді, коли перманент матриці інцидентності відмінний від нуля.

Обґрунтування. Дійсно, нехай існує СРП. В такому випадку можемо обрати відповідні стовпчики матриці (2.1), де стоять одинички, та отримати їх добуток при обрахунку перманента, який рівний 1. Нехай перманент відмінний від 0. В такому випадку існує принаймні один добуток елементів матриці інцидентності, вибраних по одному з кожного рядка та різних стовпчиків, відмінний від 0. Розглядаючи елементи множини відповідні стовпчики, отримуємо СРП.

Позначимо через P_1 умову, коли викладач не може мати більше одного практичного заняття в одній групі на день.

Властивість 2.1. Нехай виконується умова P_1. Тоді матриця денного розкладу, що складається лише з тернарних конфігурацій, завжди має допустиму форму.

Обґрунтування. Розглянемо довільну матрицю розкладу, що складається лише з тернарних конфігурацій. На початковому етапі формування розкладу рядки такої матриці можуть містити однакові елементи. Представимо стовпчики матриці розкладу: $R_1, R_2, \ldots, R_n$ як відповідні множини елементів. Зазначимо, що у разі, коли матрицю розкладу складають лише тернарні конфігурації розкладу, матриця інцидентності, складена за множинами $R_1, R_2, \ldots, R_n$, буде квадратною: матриця містить всього 3*n елементів, кількість тернарних конфігурацій – n. Матрицю інцидентності будемо будувати наступним способом:

$$A = \begin{array}{c} \\ R_1 \\ R_2 \\ \ldots \\ R_n \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & \ldots & x_n \end{array} \\ \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{pmatrix} \end{array}, \tag{2.7}$$

$$\alpha_{ij} = \begin{cases} 1, & \text{коли} \quad x_i \in R_j \\ 0, & \text{в противному випадку} \end{cases} \tag{2.8}$$

де x_i – іденифікатори викладачів.

Будемо формувати допустиму матрицю поетапно, утворюючи 1, 2, 3 рядки шляхом перестановки елементів стовпців. Для того, щоб був утворений перший рядок допустимої матриці розкладу, необхідно, аби сукупність множин (R_1, R_2, …, R_n) містила систему різних представників. Втім, за Властивістю 2.0, система різних представників наявна лише тоді, коли перманент матриці інцидентності відмінний від нуля. Отже, будемо розглядати перманент матриці A. Зазначимо, що кожен рядок матриці містить рівно три одиниці (множини R_i складаються з трьох елементів), та кожен стовпчик має рівно три одиниці (бо лише тернарні конфігурації). Припустимо, що перманент матриці A дорівнює 0. Тоді є s таких рядків та t стовпців, s+t=n+1, на перетині яких стоять нулі [156]. Без обмеження загальності можемо вважати, що матриця має вигляд:

$$A=\begin{pmatrix} 0 & 0 & \dots & 0 & \alpha_{1t+1} & \dots & \alpha_{1n} \\ 0 & 0 & \dots & 0 & \alpha_{2t+1} & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & \alpha_{st+1} & \dots & \dots \\ \alpha_{s+11} & \alpha_{s+12} & \dots & \alpha_{s+1t} & \dots & \dots & \alpha_{s+1n} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \alpha_{n1} & \alpha_{n2} & \dots & \dots & \dots & \dots & \alpha_{nn} \end{pmatrix}. \quad (2.9)$$

Але тоді, за структурою побудови матриці, в області

$$\begin{pmatrix} \alpha_{s+11} & \alpha_{s+12} & \dots & \alpha_{s+1t} \\ \alpha_{s+21} & \alpha_{s+22} & \dots & \alpha_{s+2t} \\ \dots & \dots & \dots & \dots \\ \alpha_{n1} & \alpha_{n2} & \dots & \alpha_{nt} \end{pmatrix} \quad (2.10)$$

в кожному стовпчику міститься по три одиниці, а також в області

$$\begin{pmatrix} \alpha_{1t+1} & \alpha_{1t+2} & \dots & \alpha_{1n} \\ \alpha_{2t+1} & \alpha_{2t+2} & \dots & \alpha_{2n} \\ \dots & \dots & \dots & \dots \\ \alpha_{s1} & \alpha_{s2} & \dots & \alpha_{sn} \end{pmatrix} \quad (2.11)$$

в кожному рядку міститься по три одиниці. Отже, загальна кількість одиниць в матриці A більша 3s+3t=3(n+1), що є неможливим, оскільки ця кількість, очевидно, 3n.

Отже, набір множин $\{R_1,R_2,\ldots,R_n\}$ містить систему різних представників. Така система різних представників може утворити перший рядок допустимої матриці розкладу. Очевидно, це деяка перестановка з елементів $x_1, x_2, \ldots, x_n$.

Для того, щоб сформувати другий рядок допустимої матриці, необхідно, аби була наявною система різних представників конфігурацій множин $\{R_1\backslash a_{11},R_2\backslash a_{12},\ldots,R_n\backslash a_{1n}\}$, де $a_{11},a_{12},\ldots,a_{1n}$ – перший рядок допустимої матриці розкладу.

Отже, залишається два рядка матриці розкладу, що містять *n* елементів та складаються лише з бінарних конфігурацій. Абсолютно аналогічні міркування (в аналогічних областях матриць стоятиме лише по дві одиниці) дозволяють отримати відповідне протиріччя, що доводить наявність системи різних представників.

Нехай набір конфігурацій є коректним (в кожному стовпчику є рівно три елементи). Покажемо, що в цьому випадку існує система різних представників стовпчиків матриці. Тоді можемо переставити їх, наприклад, в перший рядок.

Розглядаємо множини елементів, що повинні утворити стовпчики матриці розкладів. Послідовно вибираємо з кожного стовпчика деякий елемент так, щоб він не співпадав з вибраними раніше, і формуємо перший рядок матриці розкладів.

Припустимо, що маємо таку ситуацію, коли вибрати елемент, для якого не буде співпадіння з вибраними раніше, неможливо:

$$\begin{matrix} a_1 & a_2 & \ldots & a_k \\ * & * & \ldots & * \\ * & * & \ldots & * \end{matrix} \qquad (2.12)$$

а наступний стовпчик утворює множина елементів $\{a_{i_1}, a_{i_2}, a_{i_3}\}$, причому $i_1 \leq k, i_2 \leq k, i_3 \leq k$. Відмітимо, що в області матриці розкладу, відміченої зірочками, може бути максимум по 2 елементи з множини $\{a_1, a_2, \ldots, a_k\} \setminus \{a_{i_1}, a_{i_2}, a_{i_3}\}$ та по одному елементу з множини $\{a_{i_1}, a_{i_2}, a_{i_3}\}$. Отже, у згаданій області завжди є мінімум три елементи, що не співпадають з жодним з елементів множини $\{a_1, a_2, \ldots, a_k\}$. Тоді в області * вибираємо стовпчики, де стоять ці елементи. Якщо допустити ситуацію, коли в стовпчику може стояти два однакових елементи (викладач може мати дві пари (практичні) в одній групі в день, що навряд чи доцільно робити з методичної точки зору), то таких стовпчиків принаймні 2. В іншому випадку – рівно 3. Нехай це стовпчики з номерами j_1, j_2. Тоді ставимо замість $a_{i_1}, a_{i_2}, a_{i_3}$ у перший рядок інший елемент. Якщо можна поставити так, щоб не було співпадіння, то можемо поставити без співпадіння і якийсь елемент з стовпчика k+1. Якщо ж без співпадіння елемента вибрати не можемо, то вибираємо довільний. Відбувається співпадіння з якимось елементом в рядку. Тоді, переходячи в стовпчик співпадіння, вибираємо в ньому інший елемент в перший рядок. Процес вибору продовжуємо. Якщо немає трьох однакових елементів в стовпчику, то можемо здійснити перехід в інший. Відмітимо, що без обмеження загальності можемо вважати, що завжди можемо здійснити перехід у стовпчик, в якому раніше не були. Бо якщо такого переходу зробити не можна, то відповідна підсистема містить всі тернарні конфігурації елементів, що в неї входять. Тоді можемо виділити мінімальні замкнуті підсистеми, для яких завжди можна зробити перехід в інший стовпчик. Такі переходи здійснюємо доти, доки не потрапимо у стовпчик j_1 чи j_2, у яких вибираємо елемент, що не співпадає з жодним, що стоять в першому рядку.

Для утворення другого рядка міркуємо аналогічно. Вибираємо послідовно елементи так, щоб не було співпадіння. Якщо на якомусь кроці маємо ситуацію неможливості вибору:

$$\begin{matrix} a_1 & a_2 & \dots & a_k \dots \\ b_1 & b_2 & \dots & b_k \\ * & * & \dots & * \end{matrix} \quad (2.14)$$

і наступний елемент повинен бути вибраним з множини $\{b_{i_1}, b_{i_2}\}, i_1 \leq k, i_2 \leq k$. Тоді у відміченій зірочками області матриці повинно бути рівно 2 елементи, що не співпадають з жодним з елементів множини $\{b_{i_1}, b_{i_2}\}$. Тоді ставимо довільний елемент цієї множини у третій рядок і за допомогою аналогічного підходу утворюємо систему різних елементів другого рядка. Третій рядок, очевидно, утворюється автоматично.

Отже, матриця завжди може бути оптимізована за критерієм К1.

Як бачимо, в останньому обґрунтуванні не використовувалась умова Р1.

Тоді можемо сформулювати наступну властивість.

Властивість 2.2. Матриця розкладу, що складається лише з тернарних конфігурацій, завжди може бути перетворена шляхом перестановок елементів стовпчиків так, що задовольнятиме умову K_1.

Властивість 2.3. Матриця розкладу, що складається лише з бінарних конфігурацій, не може бути перетворена шляхом перестановок елементів стовпчиків так, щоб задовольнялась умова K_1.

Обґрунтування. Припустимо, що матриця розкладу, що складається лише з бінарних конфігурацій, задовольняє умову K_1. Звідси випливає, що кожен представник бінарної конфігурації міститься у другому рядку. Але тоді кількість всіх елементів матриці – 2*n, що є очевидним протиріччям коректного набору конфігурацій.

Властивість (критерій) 2.4. Матриця розкладу може бути перетворена шляхом перестановок елементів стовпчиків так, щоб задовольнялась умова K_1 тільки тоді, коли немає такої підсистеми векторів–стовпців, для якої виконується умова: $k_{bin} > k_{st} - k_{ter}$, де k_{bin} – кількість бінарних конфігурацій у підсистемі, k_{st} – кількість стовпців підсистеми, k_{ter} – кількість тернарних конфігурацій підсистеми.

Доведення. Необхідність: нехай матриця розкладу задовольняє умові К1. Тоді у другому рядку стоїть по одному представнику всіх бінарних та тернарних конфігурацій. Якщо виконується умова $k_{bin} > k_{st} - k_{ter}$, то отримуємо очевидне протиріччя.

Достатність: нехай виконується умова критерію. Покажемо, що матриця може бути перетворена так, щоб для неї виконувався критерій К1.

Дійсно, побудуємо допустиму матрицю за алгоритмом властивості 2.1 шляхом утворення СРПК. Це завжди можемо зробити (наприклад, з бінарних та унарних конфігурацій можемо штучно утворити тернарні і застосувати алгоритм).

Для подальших міркувань нам знадобиться наступний алгоритм. Опишемо алгоритм η_1 (будемо називати його алгоритмом «активних елементів»). Алгоритм застосовується для побудови матриці розкладу, що відповідає умові K_1.

Виділимо основні кроки алгоритму:

1. Здійснюється пошук бінарної конфігурації, що не задовольняє умову K_1. Якщо таких конфігурацій немає, кінець алгоритму.

2. Робиться «активним» довільний елемент бінарної конфігурації, що не задовольняє умову K_1.

3. Здійснюється перестановка «активного» елемента та елемента другого рядка цього ж стовпчика, «активний» елемент перестає бути «активним».

4. Розглядається елемент a, що став на місце «активного» на попередньому кроці.

При цьому виникає дві ситуації. Якщо a співпадає з певним елементом у рядку, то останній стає «активним».

Якщо ж в процесі останньої перестановки бінарна конфігурація по a «розірвалася», то активним стає елемент «розірваної» конфігурації, що не брав участі в останній перестановці.

Здійснюється перехід до кроку 3.

Якщо описаних вище ситуацій не виникає, то здійснюється перехід до кроку 1.

5. Алгоритм закінчує роботу, якщо неоптимізованих бінарних конфігурацій не залишилось.

Застосуємо алгоритм активних елементів для деякої неоптимізованої бінарної конфігурації. Покажемо, що у випадку виконання умови твердження цей алгоритм оптимізує відповідну конфігурацію, зменшивши кількість неоптимізованих конфігурацій на 1. Дійсно, припустимо, що алгоритм не закінчує свою роботу за скінченну кількість кроків (зациклюється). Розглянемо підсистему стовпчиків, що використовуються під час роботи алгоритму. Оскільки алгоритм не закінчує роботу, то активними елементами стають представники бінарних та тернарних конфігурацій, які переходять в другий рядок. Але тоді їх кількість у даній підсистемі принаймні більша на 1 від кількості стовпчиків (кількість стовпчиків + бінарна конфігурація, з представника якої стартував алгоритм).

Розглянемо приклади. В прикладах 1 та 2 розглядається одна і та ж матриця розкладу, що не задовольняє умову K_1. Порушує умову лише одна бінарна конфігурація. Активними елементами вибираються по черзі елементи цієї конфігурації. В процесі роботи алгоритму активні елементи виділені.

Приклад 1:

$$\begin{pmatrix} 1 & 3 & 2 \\ 3 & 4 & 1 \\ \mathbf{2} & 1 & 3 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 3 & 2 \\ 2 & 4 & 1 \\ 3 & 1 & \mathbf{3} \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 3 & 2 \\ 2 & 4 & 3 \\ 3 & \mathbf{1} & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 3 & 2 \\ 2 & 1 & 3 \\ 3 & 4 & 1 \end{pmatrix} \qquad (2.15)$$

Приклад 2:

$$\begin{pmatrix} 1 & 3 & \mathbf{2} \\ 3 & 4 & 1 \\ 2 & 1 & 3 \end{pmatrix} \rightarrow \begin{pmatrix} \mathbf{1} & 3 & 1 \\ 3 & 4 & 2 \\ 2 & 1 & 3 \end{pmatrix} \rightarrow \begin{pmatrix} 3 & \mathbf{3} & 1 \\ 1 & 4 & 2 \\ 2 & 1 & 3 \end{pmatrix} \rightarrow \begin{pmatrix} 3 & 4 & 1 \\ 1 & 3 & 2 \\ 2 & 1 & 3 \end{pmatrix} \qquad (2.16)$$

В прикладі 3 ілюструється ситуація, коли в процесі переходу активного елемента в другий рядок «розривається» бінарна конфігурація.

Приклад 3:

$$\begin{pmatrix} 5 & 3 & \mathbf{2} \\ 3 & 4 & 1 \\ 2 & 1 & 3 \end{pmatrix} \rightarrow \begin{pmatrix} 5 & 3 & 1 \\ 3 & 4 & 2 \\ 2 & \mathbf{1} & 3 \end{pmatrix} \rightarrow \begin{pmatrix} 5 & 3 & 1 \\ 3 & 1 & 2 \\ 2 & 4 & 3 \end{pmatrix} \quad (2.17)$$

Покажемо, що за умови існування матриці розкладу, що задовольняє умову K_1, агоритм активних елементів дозволить її побудувати.

Спочатку розглянемо довільну бінарну конфігурацію, що не задовольняє умову K_1 і покажемо, що алгоритм завершить роботу за скінченну кількість кроків. Зауважимо, що алгоритм «активних елементів», стартуючи з довільної бінарної конфігурації, що не задовольняє умову K_1, проходить замкнуту підсистему стовпчиків. Це випливає безпосередньо з самого алгоритму. Виділимо у окрему множину всі стовпчики матриці розкладів, де виникали «активні» елементи в процесі роботи алгоритму. Зауважимо, що «активний» елемент, згідно алгоритму, є обов'язково представником бінарної чи тернарної конфігурації. Причому в процесі роботи алгоритму виділяється лише один такий представник у кожному стовпчику.

Припустимо, що алгоритм «зациклюється». Це, зокрема, означає, що через певну кількість кроків знову розривається бінарна конфігурація, з якої починалась робота алгоритму. Така ситуація може виникнути, якщо в процесі роботи у одному з стовпчиків, де розміщені представники початкової бінарної конфігурації, «активним» став представник бінарної чи тернарної конфігурації, відмінний від тої, представник якої був переміщений на початку. Таким чином, отримуємо, що $k_{bin} + k_{ter} = k_{st} + 1$, де де k_{bin} – кількість бінарних конфігурацій у виділеній множині, k_{st} – кількість стовпців, k_{ter}– кількість тернарних конфігурацій у виділеній множині. А за критерієм 2.4 це означає, що матриця розкладу не може бути перетворена шляхом

перестановок елементів стовпчиківу в таку, що задовольняє умову K_1. Отримане протиріччя доводить, що алгоритм завершить роботу за скінченну кількість кроків.

Зауважимо, що в процесі роботи алгоритму ліквідовується «розрив» початкової конфігурації, а також усі розриви, що виникають у процесі роботи алгоритму. Отже, на кожній ітерації кількість розірваних бінарних конфігурацій зменшується на 1, що доводить можливість побудови матриці, що задовольняє умову K_1. Таким чином, отримали низку властивостей та критерій, які визначають прості умови, виконання яких дають можливість стверджувати, чи можна матрицю розкладу шляхом перестановок елементів стовпчиків перетворити у таку, яка задовольняє умову K_1. Окрім того, запропоновано алгоритм, що дозволяє це зробити. Однак, всі ці властивості мають місце лише у випадку, коли матриця розкладів заповнена повністю, тобто відсутніми є нульові елементи, які означають відсутність заняття. А тому необхідно розглянути окремий випадок, коли нульові елементи в матриці розкладів допускаються.

2.4.2. Властивості матриць розкладу, що містять нульові елементи

Вимога відсутності нулів у матриці розкладів є досить жорсткою і на практиці згадана ситуація цілком можлива (наприклад, викладач захворів і заміна неможлива в силу певних причин). А тому необхідно вдосконалити відповідні результати та алгоритми для випадку, коли нульові елементи в матриці розкладів допускаються.

Отже, розглянемо випадок, коли матриця розкладу може містити нулі. Очевидно, що при виконанні властивості 2.4, таку матрицю можна перетворити так, щоб виконувалась умова K_1, застосовуючи алгоритм активних елементів. Нулі можна просто вважати елементами унарних конфігурацій.

Отже, надалі будемо вважати, що маємо матрицю розкладу, що задовольняє умові $К_1$.

Введемо наступні позначення. Нехай Ψ_1 – множина стовпців матриці R, які містять нуль у другому рядку та рівно два ненульових елементи (неоптимізованих за критерієм $К_1$), $\widetilde{\Psi}_1$ – множина стовпців матриці R, які містять нуль у другому рядку та рівно два ненульових елементи – представники бінарних чи тернарних конфігурацій, Ψ_2 – множина стовпців матриці R, які не містять нуль у другому рядку та містять рівно два нульових елементи, Ψ_3 – множина стовпців матриці R, що містять унарну конфігурацію у першому чи третьому рядку і представника бінарної чи тернарної у другому.

Розглянемо алгоритм η_2 (будемо називати його алгоритмом «витіснень»). Алгоритм застосовується для перетворень матриці, що задовольняє властивості K_1 так, щоб вона задовольняла умову K_2 .

Вибираємо стовпчик матриці розкладу, який не є оптимізований за критерієм K_2 (що належить множині Ψ_1) і переміщуємо в другий рядок довільний ненульовий елемент з першого чи третього рядка.

Якщо в першому чи третьому рядку стоїть представник унарної конфігурації, то ситуація є тривіальною. Переміщуючи його в другий рядок, зразу отримуємо стовпчик, оптимізований за K_2. Якщо більше елементів множини Ψ_1 немає, то йдеться про кінець алгоритму. Інакше розглядаємо наступний елемент множини Ψ_1. Якщо в першому чи третьому рядку немає представника унарної конфігурації, то переставляємо в другий рядок довільний елемент з першого чи третього стовпця (довільний – лише на першому етапі). Очевидно, що після перестановки елемент, що перейшов у другий рядок, співпаде з елементом матриці, що стоїть в якомусь стовпчику та другому рядку. Останній елемент завжди переходить в той рядок, звідки перейшов елемент, що його «витіснив» (незалежно від того, чи це представник тернарної чи бінарної конфігурації). При цьому елемент, що перейшов у другий рядок, знову може витіснити інший елемент і т. д. (рисунок.2.6.).

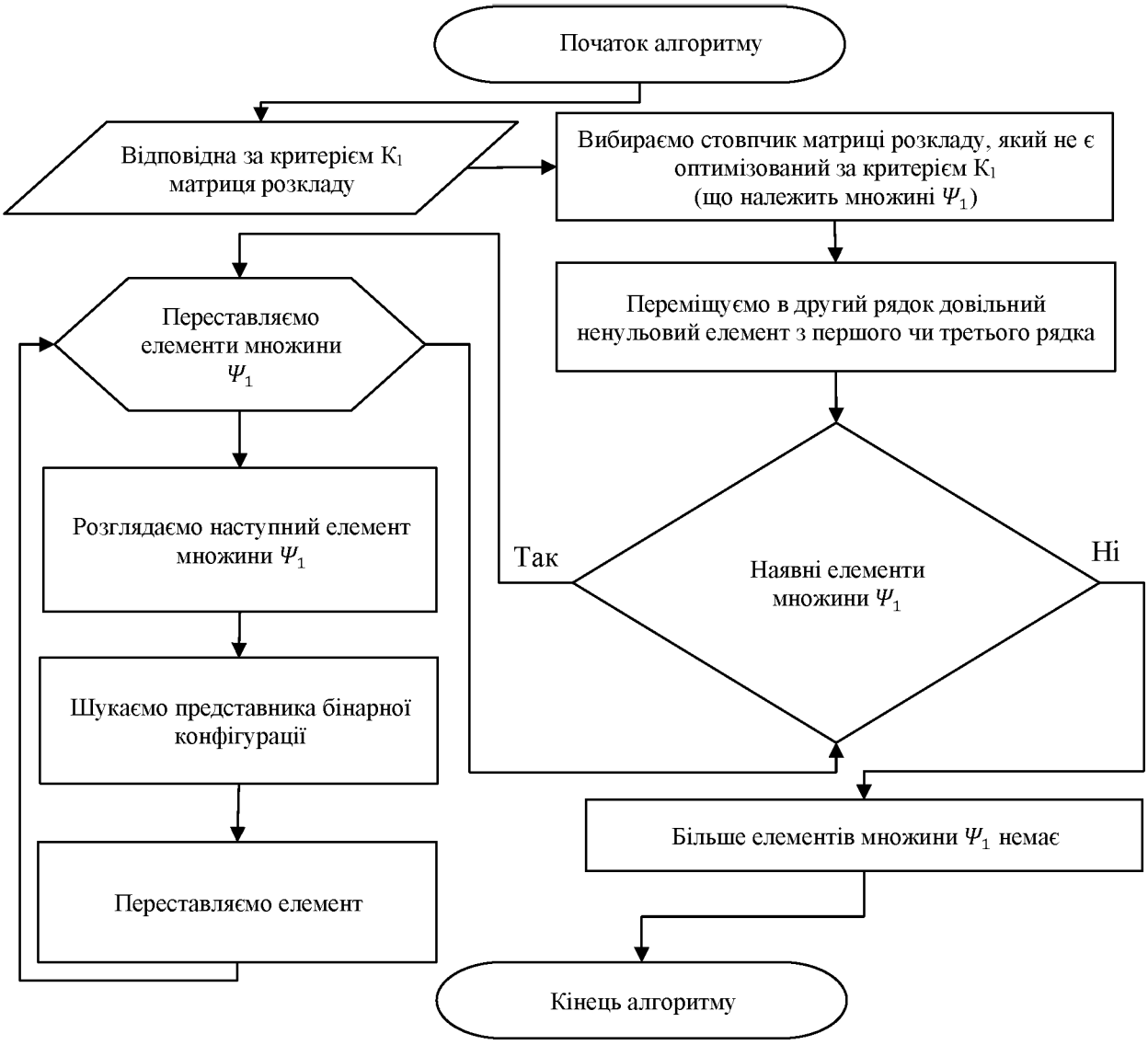

Рис.2.6. Алгоритм η_1 перетворень матриць розкладу

Якщо в другий рядок перейшов нульовий елемент стовпця, у якому є два ненульових елементи, то зразу ж в другий рядок переставляється другий ненульовий елемент стовпця.

Алгоритм, що стартував з певного неоптимізованого за K_2 стовпця, завершить роботу, якщо в другий рядок перейде представник унарної конфігурації, який не може породити подальше витіснення, або нульовий елемент стовпця, де є два нульові елементи (в цьому разі вважаємо, що стовпчик задовольняє умову K_2 за будь–якого розміщення ненульового елемента).

Розглянемо певні особливості алгоритму η_2, що безпосередньо випливають з його визначення.

1. Алгоритм η_2 не розширює множину Ψ_1.

2. Алгоритм η_2 не порушує виконання властивості K_1.

3. Алгоритм працює в одній з груп рядків 1 – 2 чи 2 – 3 , постійно змінюючи стовпчики матриці до того моменту, коли в другий рядок перейде нуль стовпця, що містить два ненульові елементи. Тоді алгоритм переходить в іншу групу.

4. Повернутись в стовпчик, що був задіяний раніше, алгоритм може лише після переходу в іншу групу з використанням елементу тернарної конфігурації.

5. Після повернення у задіяний раніше стовпчик у другий рядок буде переставлятись елемент стовпця, відмінний від того, що переставлявся туди раніше. Остання властивість зумовлює наступну.

6. Якщо у певний замкнутій підсистемі стовпців, що містить стовпчик множини ψ_1, з якого стартує алгоритм, є єдиний стовпчик множини ψ_3 то алгоритм обов'язково закінчить роботу, переставляючи у другий рядок представника унарної конфігурації.

Властивість 2.5. Матриця розкладу, що задовольняє умову K_1 та містить лише тернарні конфігурації та нулі, може бути перетворена так, щоб задовольнялась умова K_2 тоді і тільки тоді, коли немає замкнутої підсистеми стовпців, для якої кількість елементів множини $\widetilde{\Psi}_1$ більша за кількість елементів множини Ψ_2.

Властивість 2.6. Матриця розкладу, що задовольняє умову K_1 та містить лише бінарні конфігурації та нулі, завжди може бути перетворена шляхом перестановок елементів стовпчиків так, щоб задавольнялась умова K_2.

Властивість 2.7. Довільна матриця розкладу, що задовольняє умову K_1, може бути перетворена так, щоб задовольнялась умова K_2 тоді і тільки тоді,

коли немає замкнутої підсистеми стовпців, для якої кількість елементів множини $\widetilde{\Psi}_1$ більша за кількість елементів множини $\Psi_2 \cup \Psi_3$.

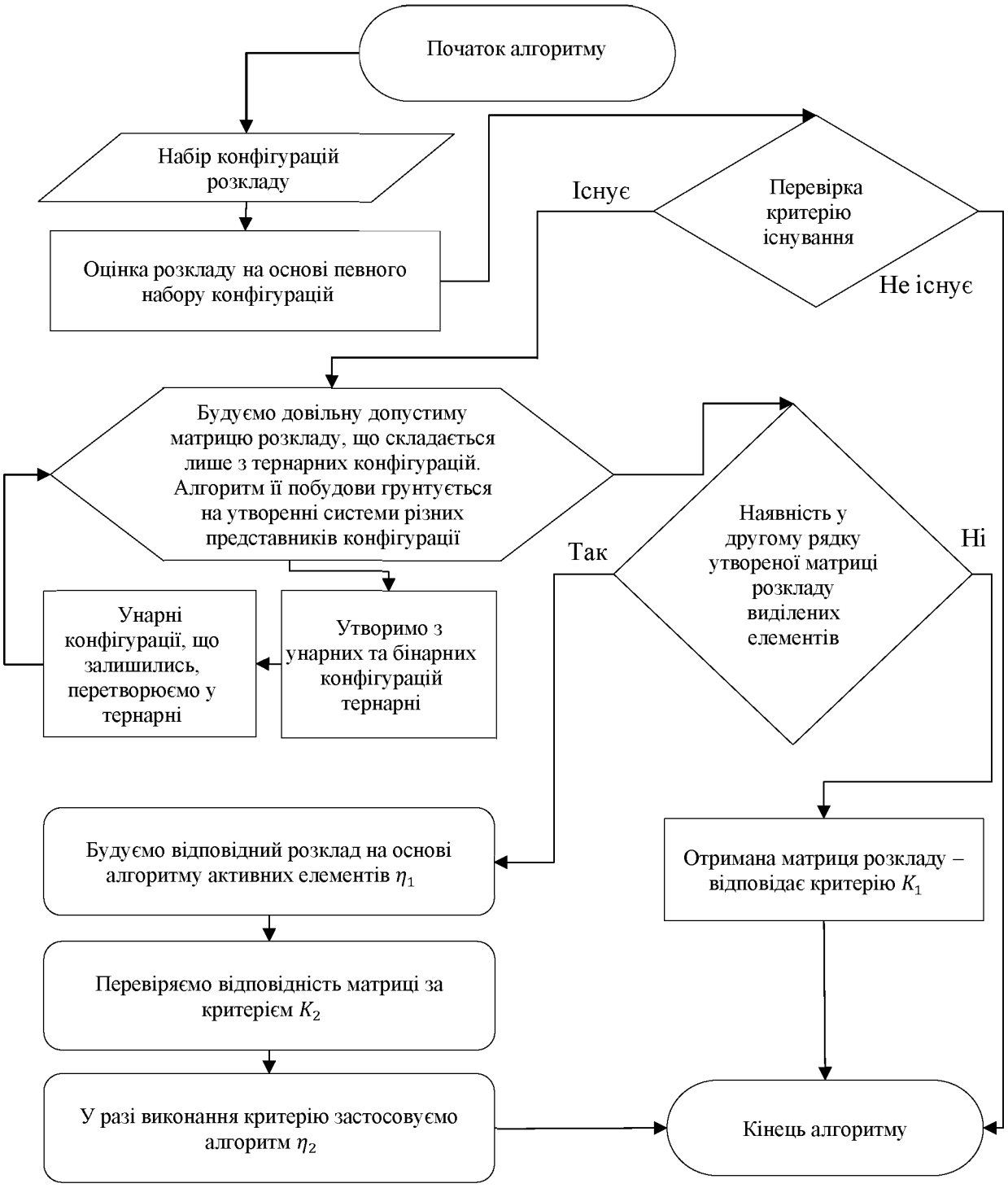

Рис.2.7. Алгоритм η_2 перетворень матриць розкладу

Для побудови оптимального розкладу на основі певного набору конфігурацій розкладу використаємо алгоритм на рисунку 2.7.

Розглянемо деякі приклади, які демонструють ефективність критеріального методу для перевірки коректності вхідних даних.

Приклад 1: покажемо, що існує приклад матриці розкладу, яка оптимізована за критерієм K_1 і не може бути оптимізована за критерієм K_2. Переконаємось, що відповідній умові задовольняє матриця наступного виду:

$$A = \begin{pmatrix} 1 & 2 & 0 \\ 0 & 2 & 1 \\ 2 & 0 & 1 \end{pmatrix} \tag{2.18}$$

Яким би чином не переставляти елементи цієї матриці, її неможливо оптимізувати за критерієм K_2.

Таким чином, умова відповідної властивості (критерію) 2.7. є коректною в тому розумінні, що існують випадки, коли вона виконується та існують, коли не виконується.

Для подальшої демонстрації ефективності критерію 2.7. наведемо інший приклад матриці розкладу, яка задовольняє K_1, містить представників різних типів конфігурацій і може бути перетворена так, щоб виконувалась умова K_2.

Приклад 4:

$$\text{A}= \begin{pmatrix} 1 & 2 & 0 & 0 \\ 0 & 1 & 2 & 4 \\ 2 & 0 & 4 & 1 \end{pmatrix},$$

$$\tilde{\Psi}_1= \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix}, \tag{2.19}$$

$$\Psi_2 \cup \Psi_3 = \emptyset .$$

матриця А не містить замкнутих підсистем.

У наступному прикладі умови критерію 2.7. виконуються і матриця легко може бути за алгоритмом η_2 перетворена так, щоб виконувалась умова K_2.

Приклад 5:

$$A=\begin{pmatrix}1 & 2 & 0 & 0\\ 0 & 1 & 2 & 3\\ 2 & 3 & 4 & 5\end{pmatrix},$$

$$\tilde{\Psi}_1=\begin{pmatrix}1\\ 0\\ 2\end{pmatrix}, \qquad (2.20)$$

$$\Psi_2 \cup \Psi_3=\begin{pmatrix}0\\ 3\\ 5\end{pmatrix},$$

$$\begin{pmatrix}1 & 2 & 0 & 0\\ 0 & 1 & 2 & 3\\ 2 & 3 & 4 & 5\end{pmatrix}\rightarrow\begin{pmatrix}1 & 2 & 0 & 0\\ 2 & 1 & 2 & 3\\ 0 & 3 & 4 & 5\end{pmatrix}\rightarrow\begin{pmatrix}1 & 2 & 0 & 0\\ 2 & 1 & 4 & 3\\ 0 & 3 & 2 & 5\end{pmatrix}.$$

На основі властивості 2.2. перевірка можливості перетворень матриці так, щоб виконувались відповідні умови, не вимагає ніяких додаткових ресурсів і може бути дуже легко здійснена на основі структури матриці розкладів – якщо кожен викладач має рівно три пари для даного варіанту матриці розкладів. Аналогічно якщо виявиться, що є лише бінарні комбінації, ніякі перестановки не забезпечують виконання умов.

Отже, маємо критерії:

1. Матриця, що складається лише з тернарних конфігурацій, завжди може бути перетворена так, що задовольнятиме умову K_1.

2. Матриця, що складається лише з бінарних конфігурацій, не може задовольняти K_1.

3. Загальний критерій можливості перетворень матриці розкладів так, щоб вона задовольняла умову K_1, де враховується кількість бінарних

конфігурацій у підсистемі, кількість стовпців підсистеми, кількість тернарних конфігурацій підсистеми.

4. Матриця, яка задовольняє умову K_1 та містить лише бінарні конфігурації та нулі, завжди може бути пертворена так, щоб задовольняла K_2.

5. Загальний критерій відповідності умові K_2 матриці, що задовольняє умову K_1 та який враховує кількість конфігурацій у підсистемах стовпців.

Бачимо, що разом з наявністю алгоритмів η_1 та η_2, проблема формування матриці розкладів розв'язується повністю за умови наявності інформації про усі конфігурації по окремих днях. Якщо така інформація, наприклад, регламентується додатковими умовами, то проблема формування матриці розкладу розв'язується повністю.

Якщо ж денні конфігурації не визначені, то критеріальний підхід дозволить здійснювати детальніший аналіз даних у поєднанні з іншими підходами.

Розглянемо довільний метод послідовної генерації матриць розкладів. Прикладом такого методу можуть бути, наприклад, алгоритми генерації на основі відношень часткового порядку. Тоді виконання описаних вище критеріїв на якійсь ітерації дозволяє прийняти поточний варіант матриці розкладу та зупинити виконання алгоритму генерації (за необхідності перейти на алгоритми η_1 та η_2). При цьому не потрібно проводити генерацію доти, доки не з'явиться варіант матриці розкладів, що задовольняє відповідні вимоги. Економія при цьому становитиме кількість кроків генерації від виконання критеріальних умов до підходящої конфігурації розкладу. Розглянемо цю характеристику детальніше.

Нехай кількість усіх можливих варіантів матриць розкладів є N, довільний алгоритм генерації дозволяє послідовно отримувати усі ці варіанти. Будемо аналізувати ймовірності того, що матриця розкладу на ітерації i задовольнятиме критерії K_1 та K_2. Тоді трудомісткість генерації в середньому рівна

$$T = 1\frac{1}{\mathrm{N}} + 2\frac{1}{\mathrm{N}} + \cdots + N\frac{1}{\mathrm{N}} = \frac{1+\mathrm{N}}{2}. \quad (2.21)$$

Нехай i – ітерація, на якій генератор завершив свою роботу, $l(i)$ – це кількість кроків роботи генератора від моменту спрацювання критерію до отримання відповідної матриці розкладу. Тоді:

$$\mathrm{T} = \big(1 - \mathrm{l}(1) + \mathrm{l}(1)\big)\frac{1}{\mathrm{N}} + \big(2 - \mathrm{l}(2) + \mathrm{l}(2)\big)\frac{1}{\mathrm{N}} + \cdots + \big(\mathrm{N} - \mathrm{l}(\mathrm{N}) + \mathrm{l}(\mathrm{N})\big)\frac{1}{\mathrm{N}} = \frac{1}{\mathrm{N}}\sum_{\mathrm{i}=1}^{\mathrm{N}}\big(\mathrm{i} - \mathrm{l}(\mathrm{i})\big) + \frac{1}{\mathrm{N}}\sum_{\mathrm{i}=1}^{\mathrm{N}}\mathrm{l}(\mathrm{i}) = \tilde{\mathrm{T}}_{\mathrm{K}} + \frac{1}{\mathrm{N}}\sum_{\mathrm{i}=1}^{\mathrm{N}}\mathrm{l}(\mathrm{i}) \quad (2.22)$$

Отже, виграш від застосування критеріального методу в середньому є $\frac{1}{N}\sum_{i=1}^{N} l(i)$.

Зауважимо, що в загальному випадку, якщо розглядати довільний метод генерації, критерій може спрацювати на довільній ітерації, причому найгірша оцінка $l(i) = i - 1$, а найкраща $l(i) = 1$. Отримати якісь точні розподіли тут дуже складно.

Виділимо, наприклад, клас методів послідовної генерації матриць розкладів, для яких в середньому $l(i) = i/2$ або виконується співвідношення:

$$\frac{1}{N}\sum_{i=1}^{N} l(i) = \frac{1}{2N}\sum_{i=1}^{N} i = \frac{N+1}{4} \quad (2.23)$$

В такому випадку:

$$\frac{\tilde{T}_K}{T} = \frac{1}{2} \quad (2.24)$$

Таким чином, застосування критеріального методу разом з методами генерації матриць розкладів, для яких середнє значення відстані (кількості ітерацій) від ітерації, де спрацювали відповідні критерії, до ітерації, яка дозволяє безпосередньо отримати відповідну форму матриці розкладів, в середньому рівна половині пройдених ітерацій, дозволяє зменшити трудомісткість таких методів генерації на 50%.

Отже, наведені вище критерії, що є елементами відповідного методу, дозволяють здійснювати аналіз структури матриці розкладів і зразу ж відкидати ті матриці, які не можуть бути оптимізовані за відповідними критеріями, що дозволяє суттєво зменшити обчислювальну складність загальних алгоритмів генерації, оскільки їх робота за відсутності такої перевірки може бути безрезультатною на відповідних ітераціях.

2.5. Висновки до другого розділу

Сформовано систему вимог до розкладу занять закладу вищої освіти, визначено низку принципових особливостей розкладу занять. Для розкладу занять в цілому не вдається визначити який-небудь ведучий критерій оцінки.

Між окремими одиницями планування (заняттями) при складанні навчального розкладу важко виявити змістовні взаємозв'язки в рамках навчального тижня, в той час як для тем предметів – основних одиниць при розробці тематичного плану – такі взаємозв'язки відіграють важливу роль.

Вимоги до розкладу занять трактуються не як проста сукупність, а як система взаємозв'язаних умов із низкою протиставлень. Побудовано формальні критерії можливості оптимізації матриць розкладів за певними класами вимог, що має важливе значення для процедур аналізу вхідних даних при використанні будь-яких алгоритмів генерації розкладів та дозволяє покращити обчислювальну складність на 50%.

Запропоновано алгоритми «витіснень» та «активних елементів» оптимізації матриць розкладу, які можуть бути використані як елементи окремої інформаційної технології генерації розкладу, так і елемент вдосконалення генетичних алгоритмів, для яких початковими істотами будуть не довільні матриці, а вже допустимі матриці розкладів.

Основні результати розділу опубліковані у працях [1–4].

РОЗДІЛ 3
МЕТОД ПЕРМАНЕНТНОЇ ДЕКОМПОЗИЦІЇ ТА МЕТОД, ЩО ҐРУНТУЄТЬСЯ НА ОСНОВІ АЛГЕБРИ АДИТИВНО-ДИЗ'ЮНКТИВНИХ ФОРМ В ЗАДАЧАХ СКЛАДАННЯ РОЗКЛАДІВ

3.1. Узагальнена структура інформаційної технології синтезу систем різних представників конфігурацій на основі методу перманентної декомпозиції

Як бачимо, задача формування розкладу є надзвичайно складною за умов наявності відповідної множини додаткових вимог. В процесі проектування тут добре підходить парадигма об'єктно-орієнтованого проектування, оскільки система має бути достатньо гнучкою і повинна легко адаптуватись до ситуації, коли якісь нові обмеження з'являються на етапі самої реалізації, тоді можна будувати відповідні ієрархії класів з використанням механізмів наслідування. Однак, окремі умови та вимоги до розкладу мають суттєво різну вагу у контексті накладання обмежень на структури даних, що можуть використовуватися в системі.

Розглянемо, наприклад, умову допустимості потоків. Така умова серйозно ускладнює задачу формування матриці розкладу та породжує низку протиріч і проблем навіть на рівні структур даних для представлення відповідної інформації. Дійсно, з одного боку очевидною є умова неможливості організації зустрічей однієї особи з різними групами слухачів у той самий час, а з іншого така зустріч можлива, якщо ці групи утворюють «потік». Інформація про потоки має бути представлена десь додатково, наприклад, в базі даних. Але це суттєво ускладнить роботу відповідних алгоритмів синтезу, оскільки доведеться постійно звертатись до цих баз даних в процесі роботи таких алгоритмів та здійснювати, зокрема, перевірку «поточності» елементів. В задачах складання розкладів структури даних для

представлення інформації відіграють фундаментальну роль, що зумовлено складністю таких задач та великим розмаїттям можливостей представлення вхідних даних.

В основі інформаційної технології (рисунок 3.1.), що пропонується в даному розділі, покладено поняття системи різних представників множин, утворених елементами стовпців матриці розкладу (СРПС), особливі способи представлення вхідної інформації на основі алгебраїчних структур – спеціальних матриць інцидентності (без використання додаткових баз даних), а також алгоритми генерації СРПС, що ґрунтуються на процедурах декомпозиції спеціальним чином модифікованих перманент матриць інцидентності.

Особливістю запропонованої інформаційної технології є те, що матриці інцидентності вже містять у собі додаткові обмеження, наприклад, інформацію про потоки, а алгоритми знаходження їх модифікованих перманент дозволяють побудувати конструктивні способи запису у пам'ять відповідних СРПС. Таким чином, окремі проблеми тут вирішуються автоматично лише за рахунок відповідної організації структур даних, як, наприклад, проблема «потоків», що дозволяє уникнути громіздких додаткових обчислювальних процедур. Метод перманентної декомпозиції адаптований до роботи в «парі» з відповідною матрицею інцидентності. В той же час, алгоритм розкладу перманента дозволяє безпосередньо здійснювати запис у потрібну комірку пам'яті необхідного значення, що дозволяє зекономити навіть на пошуку позиції в структурах даних для запису елемента-адреса тут генерується безпосередньо в процесі рекурентної процедури декомпозиції. Таким чином, бачимо, що тут мова йде про цілісну систему, елементи якої тісно пов'язані та адаптовані для оптимальної роботи один до одного.

Ідея застосування перманента як основи алгоритмів генерації, що розглянуті в даній роботі, виникла за рахунок розгляду певних алгебраїчних підходів, зокрема, процедур розкладу перманент та визначників матриць, саме

з інформаційної точки зору, з точки зору структур даних. В самому простому виразі вона полягає у тому, що до стандартної процедури розкладу перманента за рядком з метою обчислення самого значення перманента додається пам'ять, що запам'ятовує номери стовпців в процесі розкладу (рисунок 3.1).

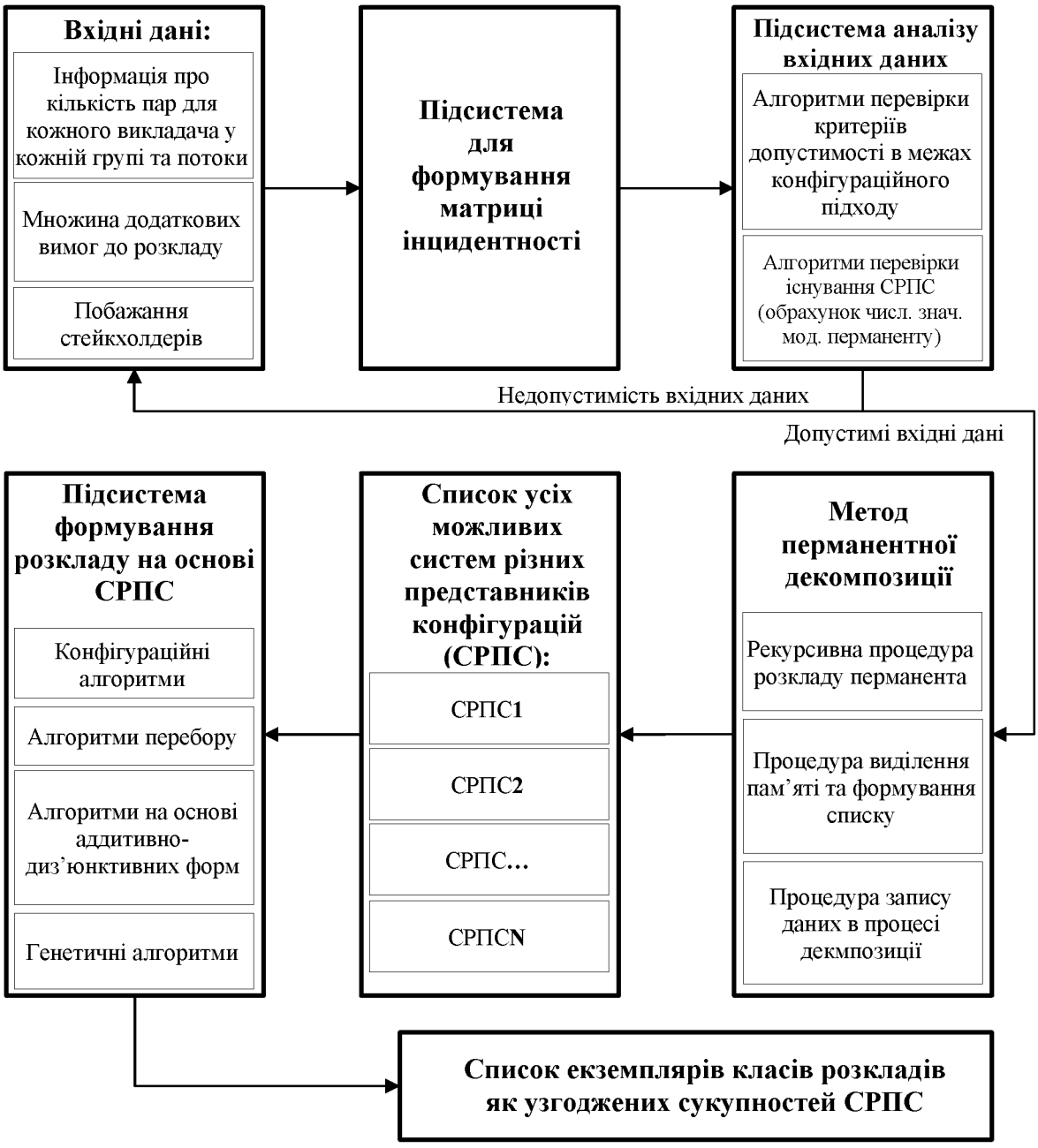

Рис.3.1. Структура інформаційної технології складання розкладів згідно перманентної декомпозиції

Метод дозволяє отримати не саме значення перманента, яке нас тут зовсім не цікавить, а алгоритм, що дозволяє, наприклад, згенерувати усі перестановки індексів стовпців у лексикографічному порядку (якщо є квадратна матриця інцидентності, що складається лише з одиничок). Сам алгоритм тут досить жорстко диктує вимоги до структур даних, які можуть бути використані для програмної реалізації: має забезпечуватись можливість прямого запису поточного індексу викладача у необхідну комірку пам'яті. Як розглядалось вище, це реалізовано через списки, а також динамічні процедури їх рекурсивного формування.

Таким чином, сформовано загальну архітектуру інформаційної технології, що дозволяє генерувати СРПС на основі вхідних даних та методу перманентної декомпозиції, який відрізняється швидкодією та зручністю представлення даних в процесі генерації. В основі методу перманентної декомпозиції покладено специфічні поняття матриць інцидентності та їх модифікованих перманент. Отже, деталізуємо ці поняття.

3.2. Модифікація перманент для матриць інцидентності

Розглянемо загальний підхід побудови систем різних представників конфігурацій стовпців. Під системою різних представників стовпців (СРПС) будемо розуміти множину, утворену елементами стовпців матриці розкладу, у якій всі елементи стовпців різні за винятком потокових елементів. Вибір потокового елемента з якогось стовпця тягне за собою автоматичний його вибір з усіх інших стовпців, де знаходиться відповідний потік.

Очевидно, що у випадку наявності потоків матриця розкладу може бути недопустимою. Розглянемо, наприклад, матрицю виду:

$$\begin{pmatrix} 1 & 1 & 1 & 2 & 3 & 5 \\ 2 & 3 & 1 & 4 & 4 & 4 \\ 3 & 1 & 4 & 5 & 2 & 4 \end{pmatrix} \qquad (3.1)$$

Будемо вважати, що одиниці в першому рядку та четвірки у другому відповідають потокам. Ця матриця немає допустимої форми. Дійсно, дана матриця має чотири тернарні конфігурації, утворені елементами 1, 2, 3, 4 та одну бінарну, утворену елементом 5. У допустимій формі обов'язково має бути по одному представнику кожної тернарної конфігурацій у кожному рядку, крім того, два рядка мають по одному представнику тернарних та бінарної. Всього стовпців – 6. Але в даному прикладі взагалі неможливо утворити рядок, який містив би елементи 1, 2, 3, 4 та 5. Дійсно, якщо одиниця чи четвірка є поточними, то три з шести елементів – «зайняті». А необхідно записати ще чотири різних елементи. Якщо ж вибрати всі представники, що не є «поточними» (така ситуація для даної матриці можлива, можна сформувати рядок, наприклад, 3, 1, 4, 5, 2), то не можемо уникнути збігу, оскільки в матриці немає жодного унарного елемента. Сформулюємо критерій існування допустимої форми матриці денного розкладу. Очевидною є наступна проста властивість.

Властивість 3.0. Матриця денного розкладу R має допустиму форму тоді і тільки тоді, коли існує така система різних представників стовпців α, що $R\backslash\alpha$ має деяку СРПС β, і $R\backslash\alpha\backslash\beta$ є СРПС (операція теоретико–множинного віднімання реалізується «по стовпцях»).

Дійсно, нехай матриця А має допустиму форму. Тоді немає збігу жодних двох елементів в рядках матриці. Але це означає, що всі рядки є СРПС і умова Властивість виконується. Нехай виконується умова Властивість. Але тоді можемо утворити матрицю, рядками якої є α, β та $A\backslash\alpha\backslash\beta$. Очевидно, що ця матриця збігається з вихідною за множинами елементів стовпців. В подальшому нам необхідно модифікувати матрицю інцидентності, а також ввести поняття модифікованого перманента.

Для представлення матриці розкладу, що містить потоки, введемо поняття матриці інцидентності конфігурацій. Нехай маємо довільну матрицю денного розкладу, розмірністю 3xn. Розглянемо стовпчики матриці розкладу

$R_1, R_2, \dots R_n$. Модифікована матриця інцидентності будується наступним способом. По горизонталі зображуються номера викладачів, по вертикалі – групи, у яких проводяться заняття. Кожному викладачу ставиться у відповідність стовпчик матриці, у якому записуються нулі та одиничі в залежності від того, чи має викладач пари у відповідних групах. Причому, якщо якийсь викладач x має поточну пару, то виділяємо йому окремий стовпчик матриці інцидентності, позначивши його x^p (можна використовувати індекс, що є кількістю елементів потоку). Таким чином, утворюється матриця виду:

$$A = \begin{array}{c} \\ R_1 \\ R_2 \\ \dots \\ R_m \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & \dots & x_n \end{array} \\ \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \end{array}, \tag{3.2}$$

$$\alpha_{ij} = \begin{cases} 1, & \text{коли } x_i \in R_j, \\ 0 & \text{в інакшому випадку.} \end{cases} \tag{3.3}$$

Наприклад, для матриці розкладів виду

$$\begin{pmatrix} 1 & 1 & 1 & 2 & 3 & 5 \\ 2 & 3 & 1 & 4 & 4 & 4 \\ 3 & 1 & 4 & 5 & 2 & 4 \end{pmatrix} \tag{3.4}$$

отримуємо наступну матрицю інцидентності:

$$\begin{array}{c} \\ R_1 \\ R_2 \\ R_3 \\ R_4 \\ R_5 \\ R_6 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 1^p & 2 & 3 & 4 & 4^p & 5 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \end{array} \tag{3.5}$$

де 1^p та 4^p – індекси відповідних викладачів, що мають «поточні» пари.

В основі методу перманентної декомпозиції покладено поняття перманент та їх властивостей. Причому певні модифікації класичного поняття перманента лягли в основу оригінальних алгоритмів формування розкладів.

Розглянемо класичну процедуру обрахунку перманента матриці розмірності m x n шляхом розкладу за рядком [1]. Очевидно, що для неї необхідно $A_n^m - 1$ додавань та $A_n^m(m-1)$ добутків.

Зауважимо, що рекурсивне обчислення перманента шляхом розкладу за рядком дозволить суттєво скоротити кількість арифметичних операцій. Дійсно, нехай $x_{m,n}$ $y_{m,n}$ – кількість операцій відповідно множення та додавання при вказаному способі обчислення перманента матриці розміром $m \times n$. Тоді, враховуючи процедуру розкладу перманента, маємо: $y_{m,n} = n - 1 + ny_{m-1,n-1}$ (*n–1* додавань компонент розкладу за рядком, а також наявність *n* компонент розкладу).

Звідси:

$$y_{m,n} + 1 = n(y_{m-1,n-1} + 1) = n(n-1)(y_{m-2,n-2} + 1) = \ldots$$
$$= n(n-1)(n-2)\ldots(n-m+2)(y_{1,n-m+1} + 1) \quad (3.6)$$

Враховуючи, що $y_{1,k} = k - 1$ маємо:

$$y_{m,n} + 1 = n(y_{m-1,n-1} + 1) = n(n-1)(y_{m-2,n-2} + 1) = \ldots$$
$$= n(n-1)(n-2)\ldots(n-m+2)(n-m+1) = A_n^m \quad (3.7)$$

Бачимо, що кількість операцій додавання таке саме, як при безпосередньому обчисленні перманента. Оцінимо кількість операцій множення $x_{m,n}$.

Маємо: $x_{m,n} = n + nx_{m-1,n-1}$. Звідси $x_{m,n}/n! = 1/(n-1)! + x_{m-1,n-1}/(n-1)!$

Далі:

$$\frac{x_{m,n}}{n!} = \frac{1}{(n-1)!} + \frac{x_{m-1,n-1}}{(n-1)!} = \frac{1}{(n-1)!} + \frac{1}{(n-2)!} + \ldots + \frac{1}{(n-m+1)!} + \frac{x_{1,n-m+1}}{(n-m+1)!} \quad (3.8)$$

Оскільки $x_{1,k} = 0$, маємо:

$$x_{m,n} = \frac{n!}{(n-1)!} + \frac{n!}{(n-2)!} + \cdots + \frac{n!}{(n-m+1)!} = A_n^1 + A_n^{21} + \ldots + A_n^{m-1} \quad (3.9)$$

Для квадратної матриці:

$$A_n^1 + A_n^{21} + \ldots + A_n^{n-1} \approx n!\,(e-1), \qquad n!\,(e-1) \leq n!\,(n-1) \quad (3.10)$$

Отже, кількість операцій множення суттєво менше, ніж при безпосередньому обчисленні. Ця обставина є дуже важливою і буде використана нами при введенні понять модифікованих перманент та розробці алгоритмів перманентної декомпозиції та їх модифікацій.

*Означення***.** Модифікованим перманентом матриці інцидентності будемо називати суму всіх можливих добутків елементів матриці, кожен з яких містить по одному елементу з кожного рядка та з різних стовпців, причому елемент потокового стовпця (стовпця, що відповідає потоковому елементу) не може бути в добутку разом з елементами інших рядків, що відповідають цьому ж потоку.

Зауважимо, що за відсутності потокових елементів модифікований перманент є звичайним перманентом.

Для знаходження модифікованого перманента матриці інцидентності використаємо наступний алгоритм розкладу за рядком, який назвемо Δ–алгоритмом. Ненульовий елемент рядка множиться на модифікований

перманент матриці, утвореної за наступними правилами – якщо елемент рядка належить потоковому стовпчику, то матриця утворюється з вихідної викреслюванням стовпця, де цей елемент знаходиться, всіх рядків, що відповідають всім елементам даного потоку, а також стовпчик, що відповідає такому ж непоточному елементу. Якщо елемент рядка не належить потоковому стовпчику, то матриця утворюється шляхом викреслювання рядка та стовпця, де стоїть цей елемент, а також всіх потокових стовпців, на перетині яких з цим рядком стоять ненульові елементи.

Розглянемо як приклад процедуру обчислення модифікованого перманента для матриці розкладів (3.4). Для ілюстрації алгоритму розкладу в матриці інцидентності запишемо ідентифікатори стовпців. Маємо:

$$A = \begin{pmatrix} 1 & 1^p & 2 & 3 & 4 & 4^p & 5 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{3.11}$$

При обрахунку перманента будемо будувати розклад за першим рядком (потокові стовпці – 2-й та 6-й):

$$permodA = 1 * permod \begin{pmatrix} 1 & 2 & 3 & 4 & 4^p & 5 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} + 1 * permod \begin{pmatrix} 1 & 3 & 4 & 4^p & 5 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} + 1 * permod \begin{pmatrix} 1 & 2 & 4 & 4^p & 5 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}. \tag{3.12}$$

Далі процедура розкладу здійснюється аналогічно.

Очевидною є наступна властивість.

Властивість 3.1. Δ-алгоритм дозволяє знайти модифікований перманент матриці інцидентності.

Дійсно, згідно з процедурою розкладу за рядком кожен ненульовий елемент множиться на перманент матриці, яка утворюється викреслюванням не лише стовпчика, де знаходиться цей елемент (що забезпечує його присутність у добутку лише один раз), але й усіх рядків та стовпчиків таким чином, щоб виключити одночасну присутність в окремому добутку потокових та непотокових елементів. Таким чином, отримуємо суму всіх можливих добутків елементів матриці, кожен з яких містить по одному елементу з кожного рядка та з різних стовпців, причому елемент стовпця, що відповідає потоковому елементу, не може бути в добутку разом з елементами інших рядків, що відповідають цьому ж потоку.

Властивість 3.2. Матриця розкладів має СРПС тоді і тільки тоді, коли модифікований перманент матриці інцидентності відмінний від нуля.

Нехай маємо матрицю розкладів розмірності 3 x n. Нехай існує СРПС матриці розкладів $x^1, x^2, \ldots, x_n$. Тоді, побудувавши матрицю інцидентності, бачимо, що перманент матриці відмінний від нуля, оскільки перманент матриці, утвореної стовпцями, що відповідають елементам $x^1, x^2, \ldots, x_n$ є одиницею:

$$\begin{pmatrix} x_1 & x_2 & \ldots & \ldots & x_n \\ 1 & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 0 & \ldots & 1 \end{pmatrix} \qquad (3.13)$$

Наявність потокових елементів не погіршує ситуацію. Якщо, наприклад, x^1=x^2, а інші елементи різні, то маємо матрицю інцидентності:

$$\begin{pmatrix} x_1^p & x_2 & \dots & \dots & x_n \\ 1 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix} \tag{3.14}$$

модифікований перманент якої теж, очевидно, є одиницею.

Нехай модифікований перманент матриці інцидентності відмінний від нуля. Тоді в його розкладі є одиничний елемент, тобто перманент деякої матриці, утвореної підсистемою стовпців, дорівнює 1. Але якщо розглянути елементи, що відповідають стовпцям цієї підсистеми, то неважко побачити, що вони утворюють СРПС. Властивість доведена.

Отже, для модифікованої матриці інцидентності з врахуванням потоків вперше запропоновано поняття модифікованого перманента та процедуру його обчислення. Отримано критерій існування СРПС на основі перманент. В подальшому виникає необхідність у модифікації процедури декомпозиції перманента з можливістю формування відповідних структур даних.

3.3. Метод перманентної декомпозиції формування системи різних представників стовпців матриці розкладу

Для побудови процедури генерації усіх можливих СРПС будемо використовувати процедуру перманентної декомпозиції. Зауважимо, що кожній СПРС відповідає підматриця матриці інцидентності з одиничним перманентом. Крім того, саму СПРС можна отримати, якщо ввести процес «запам'ятовування» елемента стовпця, що відповідає одиниці, яка використовується при побудові добутку в даний момент та рядка, в якому ця одиниця стоїть у вихідній матриці інцидентності. У зв'язку з цим у процесі обчислення перманента будемо кожній одиничці дописувати два індекси: верхній – елемент, що відповідає стовпчику, де стоїть ця одиничка, нижній – номер рядка, в якому ця одиниця стоїть у вихідній матриці.

Нехай, наприклад, маємо матрицю розкладу виду:

$$R = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 3 & 1 \\ 3 & 1 & 4 \end{pmatrix} \tag{3.15}$$

Тоді матриця інцидентності має вигляд:

$$\begin{pmatrix} & 1 & 1^n & 2 & 3 & 4 \\ R_1 & 0 & 1 & 1 & 1 & 0 \\ R_2 & 1 & 1 & 0 & 1 & 0 \\ R_3 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \tag{3.16}$$

Побудуємо процес розкладу модифікованого перманентаз «запам'ятовуванням» за першим рядком:

$$per\ mod \begin{pmatrix} 1 & 1^n & 2 & 3 & 4 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix} = 1_1^{1^p} * 1 + 1_1^2 * per\ mod \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} + 1_1^3 *$$

$$per\ mod \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix} = 1_1^{1^n} * 1 + 1_1^2$$

$$* \left(1_2^1\ per\ mod\ (0 \quad 1) + 1_2^3\ per\ mod\ (1 \quad 1)\right) + \cdots +$$

$$1_1^3 1_2^1 per\ mod\ (0 \quad 1) = 1_1^{1^n} + 1_1^2 1_2^1 1_3^4 + 1_1^2 1_2^3 1_3^1 + 1_1^2 1_2^3 1_3^4 + 1_1^3 1_2^1 1_3^4. \tag{3.17}$$

Бачимо, що модифікований перманент рівний 5 (і це є кількість всіх можливих СРПС). Крім того, відомі й самі СРПС. Вони записуються як верхні індекси «одиничок». Причому, якщо певні нижні індекси відсутні, то відповідну кількість разів повторюється верхній елемент (ситуація потокового елементу): *111, 214, 231, 234, 314.*

Узагальнимо описану вище процедуру формування СРПС та покажемо, що для довільної матриці інцидентності процедура перманентної декомпозиції забезпечує формування усіх можливих СРПС.

Дійсно, розглянемо частковий випадок матриці інцидентності, що містить усі 1 за додаткової умови відсутності потокових елементів:

$$\begin{matrix} \\ R_1 \\ R_2 \\ \dots \\ R_n \end{matrix}\begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ \dots & \dots & \dots & \dots & \dots \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{3.18}$$

Властивість 3.3. Результатом перманентної декомпозиції за будь-яким рядком «з запам'ятовуванням» матриці (3.18) є усі можливі перестановки елементів $a_1, a_2, \dots, a_n$.

Обґрунтування. Будемо доводити методом математичної індукції по n. При *n=2* твердження очевидне. Нехай твердження справедливе при деякому *k=n–1*. Покажемо, що воно виконується при *k=n*.

Маємо:

$$per\begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ \dots & \dots & \dots & \dots & \dots \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} = 1_1^{a_1} * per\begin{pmatrix} a_2 & a_3 & \dots & a_n \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ \dots & \dots & \dots & \dots \\ 1 & 1 & 1 & 1 \end{pmatrix} +$$

$$+1_1^{a_2} * per\begin{pmatrix} a_1 & a_3 & \dots & a_n \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ \dots & \dots & \dots & \dots \\ 1 & 1 & 1 & 1 \end{pmatrix} + \cdots 1_1^{a_n} * per\begin{pmatrix} a_2 & a_3 & \dots & a_{n-1} \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ \dots & \dots & \dots & \dots \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

(3.19)

Бачимо, що в процесі декомпозиції кожен доданок суми, згідно з методом розкладу, буде формувати множини СРПС, що починаються відповідно елементами $a_1,a_2,\ldots, a_n$, причому кожна матриця розкладу матиме розмірність *n–1* x *n–1* а отже, згідно припущення індукції, генеруватиме усі можливі перестановки відповідних індексних елементів. Таким чином, отримуємо усі можливі перестановки елементів, що починаються відповідно $a_1, a_2,\ldots, a_n$, що і доводить властивість 3.3.

Зауважимо, що у випадку послідовного запису усіх отриманих в результаті перманентної декомпозиції перестановок у пам'ять із використанням відповідних структур даних, усі перестановки будуть записані в лексикографічному порядку, якщо впорядкованими є елементи $a_1,a_2,\ldots, a_n$.

Розглянемо випадок матриці інцидентності, що містить усі 1 за додаткової умови відсутності потокових елементів та розмірності *m*x*n*, *m*<*n*:

$$\begin{array}{c} \\ R_1 \\ R_2 \\ \ldots \\ R_m \end{array} \begin{pmatrix} a_1 & a_2 & a_3 & \ldots & a_n \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \qquad (3.20)$$

Властивість 3.4. В процесі перманентної декомпозиції за будь-яким рядком «із запам'ятовуванням» матриці (3.20) отримуємо усі можливі розміщення з *n* елементів $a_1, a_2,\ldots, a_n$ по *m*.

Обґрунтування даної властивості легко зробити аналогічно до попередньої, розглядаючи індукцію по *n*.

Розглянемо матрицю інцидентності розмірності *m* x *n*, за умови відсутності потокових елементів, що може містити нульові елементи або одиниці.

$$\begin{array}{c} \\ R_1 \\ R_2 \\ \dots \\ R_m \end{array} \begin{array}{c} \begin{array}{cccc} a_1 & a_2 & \dots & a_n \end{array} \\ \begin{pmatrix} \aleph_{11} & \aleph_{12} & \dots & \aleph_{1n} \\ \aleph_{21} & \aleph_{22} & \dots & \aleph_{2n} \\ \dots & \dots & \dots & \dots \\ \aleph_{m1} & \aleph_{m2} & \dots & \aleph_{mn} \end{pmatrix} \end{array} \tag{3.21}$$

$$\aleph_{ij} = \begin{cases} 1, & \text{коли } a_j \in R_i, \\ 0 & \text{в інакшому випадку.} \end{cases} \tag{3.22}$$

В такому випадку, очевидно, наявність нуля в *k*–тому рядку означає, що відповідний індексний елемент не може бути присутнім на *k*–тому місці в жодній з перестановок, адже згідно з визначенням процедури декомпозиції розглядаються лише ненульові елементи.

Таким чином, має місце властивість:

Властивість 3.5. Результатом перманентної декомпозиції за будь-яким рядком «з запам'ятовуванням» матриці (3.14) є усі можливі розміщення з *n* елементів $a_1, a_2, \dots, a_n$ по *m* виду $(a_{i_1}, a_{i_2}, \dots, a_{i_m}), i_k \in \{1,2,,,n\}, k = \overline{1,m}$

$$i_k \neq i_l, k \neq l, k \tag{3.23}$$

для яких виконується умова:

$$\forall a_{i_k}, k = \overline{1,m}, \aleph_{ki_k} = 1 \tag{3.24}$$

Розглянемо випадок наявності поточних елементів.

Властивість 3.6. Результатом декомпозиції модифікованого перманента матриці (3.21) за умови наявності потоків є усі можливі розміщення з *n* елементів $a_1, a_2, \dots, a_n$ по *m* виду з врахуванням наявності поточних елементів $(a_{i_1}, a_{i_2}, \dots, a_{i_m}), i_k \in \{1,2,,,n\}, k = \overline{1,m}$,$i_k \neq i_l, k \neq l, k$ для яких виконується умова:

$$\forall a_{i_k}, k = \overline{1,m}, \aleph_{ki_k} = 1 \qquad (3.25)$$

Дійсно, розглянемо процедуру розкладу за рядком матриці інцидентності (обчислення модифікованого перманента з врахуванням потоків). Вона відрізняється від попередньої лише тим, що в процесі розкладу, якщо біжучим є поточний елемент, то крім стовпчика викреслюються усі рядки, що відповідають ненульовим елементам потоку (забезпечує примусове включення усіх поточних елемнтів), а також стовпчик, що має аналогічний до потокового ідентифікатор (виключається співпадіння потокового та звичайного елемента). Якщо викреслюється стовпчик та рядок для непотокового елемента, то одночасно викреслюються усі стовпчики, для яких в цьому рядку стоять потокові елементи (виключається одночасне включення елемента, а також некоректного фрагмента потоку). Така модифікація процесу розкладу забезпечує лише коректний запис потокових елементів, а також виключає співпадання ідентифікаторів у одному розміщенні. Очевидно, вона не впливає на формування усіх можливих розміщень, що доводилось вище.

При наявності всіх можливих СРПС, є допустимим вирішення задач різних класів. Наприклад, якщо нас цікавить допустима форма матриці розкладу, то вона існує лише за умови, коли у множині всіх СПРС є такі три СПРС, що не містять в собі на однакових місцях однакових елементів, окрім елементів потоків. Для даного прикладу існує тільки один набір СРПС: *111, 314, 231.*

Отже, будемо формувати допустиму матрицю поетапно, утворюючи перший, другий, третій рядки (і т.д.), шляхом перестановок елементів стовпців. Для того, щоб був сформований перший рядок допустимої матриці розкладу, необхідно, щоб конфігурація множин стовпців $\{R_1, R_2, \dots, R_n\}$ містила систему різних представників (СРП). Але СРП існує тоді і тільки тоді, коли перманент матриці інцидентності конфігурації відмінний від нуля. Отже, будемо розглядати перманент матриці А. Якщо він відмінний від нуля, то існує

СРП: $a11, a12, \ldots, a1n$. Дана СРП утворює перший рядок допустимої матриці розкладу. Для того, щоб сформувати другий рядок допустимої матриці, необхідно, щоб існувала СРП конфігурацій множин $\{R1/a11, R2/a12, \ldots Rn/a1n\}$, де $a11, a12, \ldots, a1n$ – перший рядок допустимої матриці розкладу. Відповідні міркування продовжуємо.

Відмітимо, що початковий процес формування конфігурацій може бути або автоматичним, з міркувань мінімальної кількості робочих днів викладача, або являти собою результат домовленості з викладачем (наприклад, деяким викладачам важко проводити три пари в один день, тоді можна утворити відповідні бінарні чи навіть унарні конфігурації, якщо такі усіх влаштовують). При такому підході вирішується проблема мінімізації «вікон» викладачів вже на початковому етапі розв'язку задачі: матриця розкладу являє собою набір вже оптимізованих (що не мають вікон) конфігурацій.

Маючи всі можливі СПРС, очевидно, можемо вирішувати різні класи задач. Наприклад, якщо нас цікавить допустима форма матриці розкладу, то вона в даному випадку існує лише у тому випадку, коли в множині всіх СПРС є такі три СПРС, що не мають на однакових місцях однакових елементів, за винятком елементів потоків.

В останньому прикладі такий набір СПРС єдиний: *111*, *314*, *231*.

Відповідну умову можна узагальнити.

Властивість 3.7. Допустима форма матриці розкладу існує тоді і тільки тоді, коли у множині всіх СРПС існує така підмножина, що множини елементів, утворені першими, другими,..., k–тими елементами всіх СРПС цієї підмножини співпадають з множинами елементів стовпців матриці розкладів (к–кількість рядків матриці розкладів).

Ця властивість є очевидною, оскільки набір СРПС утворює допустиму форму матриці розкладів.

Відмітимо, що *Властивість 3.3* носить загальний характер. При побудові розкладу, очевидно, необхідно розглядати всі дні тижня (досі

розглядались лише матриці денного розкладу). Тоді будемо мати загальну матрицю розкладу, що має 15 рядків (5 робочих днів по три пари в день). Стовпчики, очевидно, можуть містити елементи, що повторюються. Тоді при побудові матриці інцидентності необхідно записувати таку кількість стовпців, що відповідають повторюваному елементу, скільки разів він зустрічається у стовпчику. Останню обставину необхідно враховувати і у випадку побудови матриці інцидентності денного розкладу, якщо допускається можливість проведення двох практичних у тій самій групі в день, тобто наявність двох однакових непоточних елементів у стовпчику матриці.

Приклад:

$$R = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 3 & 1 \\ 2 & 1 & 4 \end{pmatrix} \tag{3.26}$$

Тоді матриця інцидентності має вигляд:

$$\begin{matrix} \\ R_1 \\ R_2 \\ R_3 \end{matrix} \begin{pmatrix} 1 & 1^{\text{п}} & 2 & 2 & 3 & 4 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.27}$$

Означення модифікованого перманента та механізм розкладу за рядком такої матриці аналогічний розглянутому вище випадку.

Оскільки описаний вище підхід дозволяє побудувати всі можливі СПРС, маємо всю інформацію про можливі рядки матриці розкладів, то, очевидно, на етапі вибору певних СРП можемо вирішувати різні задачі, що випливають з певного поняття оптимальності. Умова, що розглядається у теоремі 2, дозволяє побудувати лише допустиму матрицю розкладу. Очевидно, можемо розглянути ряд інших критеріїв, яким повинен задовольняти розклад. Розглянемо, наприклад, критерій K_2, який розглядався в роботі [4]. Його можна

розглянути як додаткову умову при знаходженні підмножини СПРС, враховуючи наявність відповідної перестановки СПРС, що автоматично забезпечить розбиття розкладу «по днях». Очевидно, що можна розглянути і інші умови [4].

Таким чином, побудовано новий метод перманентної декомпозиції, в процесі якого формуються усі СРПС в лексикографічному порядку, що дозволяє розробляти ефективні засоби програмної реалізації. Однак, вимагає детальнішого дослідження складність відповідних алгоритмів. Для порівняння розглянемо детальніше підхід до генерації матриць розкладів, що базується на основі відношень порядку.

3.4. Алгоритми генерації матриць розкладів на основі відношень порядку

Відомими є низка алгоритмів генерації кортежів та перестановок, які добре описані ще в роботах Д. Кнута [156]. Найпопулярнішими серед них є алгоритми, що використовують спеціальні відношення порядку, зокрема, лексикографічного. Однак, безпосередньо роботи, які стосувалися б генерації матриць розкладу саме в такому контексті, як пропонується вище за наявності потоків, автору є невідомими. А тому, для порівняння ефективності методу перманентної декомпозиції з іншими методами слід провести певну деталізацію методу генерації матриць розкладу на основі відношень порядку. Алгоритми, що при цьому виникають, в головних рисах не відрізняються від добре відомих, а тому будемо використовувати їх як основу для порівняння з методом перманентної декомпозиції.

3.4.1. Особливості інформаційної технології

У практиці розв’язання широкого класу задач, які стосуються генерації повної сукупності комбінаторних об’єктів, часто застосовується підхід, який

полягає у введенні спеціального відношення порядку на множині об'єктів та знаходження на кожній ітерації наступного об'єкта, найближчого до біжучого в контексті відповідного відношення порядку.

Якщо відношення порядку введене коректно, то таким чином гарантовано буде згенерована уся сукупність об'єктів.

У цьому розділі розглянемо інформаційну технологію складання розкладів, що ґрунтується на використанні такого підходу (рисунок 3.2.).

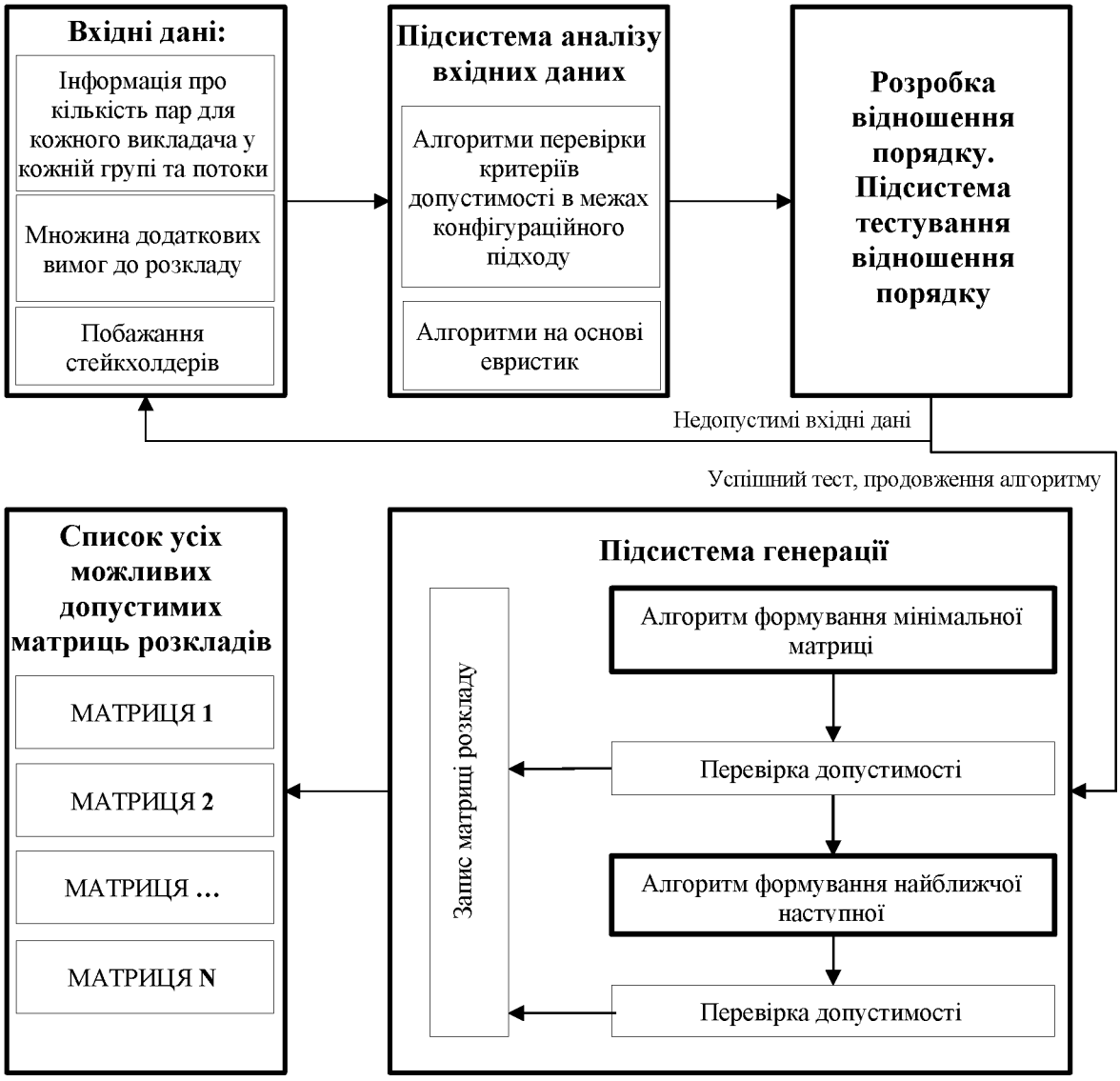

Рис.3.2. Структура інформаційної технології складання розкладів згідно перманентної декомпозиції, що ґрунтується на використанні відношення порядку

Як бачимо з рисунку 3.2., фундаментальну роль в технології, що пропонується, відіграє відношення порядку. Воно повинно бути коректним (зокрема, виключати рівність для будь-яких двох згенерованих матриць). В підсистемі генерації важливою є коректність алгоритму знаходження найближчої матриці в сенсі відношення порядку, не повинна порушуватись інформаційна структура матриці в розумінні кількості конфігурацій. Правильність алгоритму потрібно доводити, необхідною є підсистема тестування.

Звернемо увагу на підсистему перевірки коректності вхідних даних. На відміну від технології перманентної декомпозиції, в даному випадку форма представлення вхідних даних може бути довільною, адже тут не будуються матриці інцидентності. А тому необхідно використовувати набір евристик та вимог до розкладу.

3.4.2. Відношення порядку та алгоритми генерації для випадку відсутності потоків

Розглянемо задачу формування розкладів з точки зору відношень порядку. Для побудови алгоритму складання розкладу необхідно ввести відношення порядку на множині матриць розкладів.

Введемо відношення порядку на множині матриць розкладу. Нехай маємо дві матриці розкладу A, B. Позначимо через $a_1, a_2, \dots, a_m$ та $b_1, b_2, \dots, b_m$ рядки матриць A та B відповідно, $a^1, a^2, \dots a^n, b^1, b^2, \dots b^n$ – стовпчики. Будемо вважати, що одна матриця розкладу $A \prec_1 B$ якщо існує такий індекс i при якому $a^i <_L b^i$, де $<_L$– відношення лексикографічного порядку на можині стовпиків матриць, $a^k = b^k, 1 \le k < i$.

Очевидно, що введене таким чином відношення є, фактично, відношенням лексикографічного порядку для рядків, утворених шляхом послідовного об’єднання транспонованих стовпців матриць розкладів.

Аналогічно можемо розглянути інше відношення порядку для матриць розкладів, яке буде еквівалентне відношенню лексикографічного порядку для рядків, утворених шляхом послідовного об'єднання (конкатенації) рядків матриць розкладів:

$A \prec_2 B$ якщо існує такий індекс *i* при якому $a_i <_L b_i$, де $<_L$ – відношення лексикографічного порядку на множині рядків матриць розкладів, $a_k = b_k, 1 \leq k < i$.

Тепер, маючи відношення порядку на множині матриць розкладів, можемо використовувати алгоритм генерації усіх можливих матриць розкладу від найменшої до найбільшої, на кожному кроці шукаючи найближчу більшу матрицю до біжучої. При цьому пошук найближчої більшої матриці буде суттєво залежним від вибраного нами відношення порядку. Окрім того зауважимо, що в процесі генерації матриць міняти місцями, сортувати можна лише елементи в стовпця. Розглянемо цю процедуру детальніше.

Очевидно, що «найменшою» матрицею розкладу стосовно обох відношень порядку буде матриця, утворена наступним чином: спочатку формується перший рядок, що складається з мінімальних елементів стовпців, тоді другий рядок, що складається з мінімальних елементів стовпців, які залишились після першого вибору і т.д..

Розглянемо відношення порядку $\prec_1$. Тоді генерація найближчої більшої матриці здійснюється на основі наступного алгоритму.

1. Переглядаються усі стовпчики в зворотному порядку починаючи з останнього, знаходиться перша ситуація, коли один елемент стовпця менший за сусідній нижній $a_{ij} < a_{i+1j}$.

2. Знаходиться мінімальний елемент в біжучому стовпчику починаючи з позиції $i + 1$, який більший за елемент a_{ij}. Елемент a_{ij} міняється місцями зі знайденим.

3. Всі елементи у біжучому стовпчику $a_{i+1j}, a_{i+2j}, \ldots, a_{mj}$, а також у стовпцях $a^{j+1}, a^{j+2}, \ldots, a^n$ сортуються у порядку зростання.

Очевидно, що описаний вище алгоритм є правильним та дає на виході найближчу більшу за біжучу матрицю розкладів, оскільки знаходить найближчу позицію «з кінця» матриці розкладів, у якій матриця може бути збільшена на мінімальну величину.

3.4.3. Уточнення лексикографічного алгоритму при наявності «потоків»

Зауважимо, що при наявності потоків необхідно модифікувати описаний вище алгоритм. Якщо більшим виявився звичайний елемент, але над ним стоїть потік, то в процесі сортування потік має ігноруватись, бо тоді потік може зіпсувати положення самого біжучого елемента.

Розглянемо приклад:

$$\begin{pmatrix} 1^n & 1^n & 1^n & 2 \\ 2 & 3 & 4^n & 4^n \\ 4 & 5 & 1 & 2 \\ 3 & 4 & 5 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1^n & 1^n & 1^n & 1 \\ 2 & 3 & 5 & 2 \\ 4 & 5 & 1 & 2 \\ 3 & 4 & 4^n & 4^n \end{pmatrix} \tag{3.28}$$

Якщо по останньому стовпчику сортувати з поточним елементом, то поточний елемент стає в останній рядок, витісняючи і поточний елемент 1 та 5, що його замінив. Отже, можемо отримати гіршу ситуацію, ніж без врахування поточного елемента. В той же час, загалом поточний елемент при сортуванні може не змінювати поточний і навіть покращити ситуацію. Отже, наявність потокових елементів у стовпчику, що сортується, призводить до серйозного ускладнення: в процесі сортування відповідного стовпця необхідно перевіряти оптимальне розміщення потокових елементів.

А ось алгоритм, коли при сортуванні ігнорується потоковий елемент:

$$\begin{pmatrix} 1^n & 1^n & 1^n & 2 \\ 2 & 3 & 4^n & 4^n \\ 4 & 5 & 1 & 2 \\ 3 & 4 & 5 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1^n & 1^n & 1^n & 1 \\ 2 & 3 & 4^n & 4^n \\ 4 & 5 & 5 & 2 \\ 3 & 4 & 1 & 2 \end{pmatrix} \quad (3.29)$$

Розглянемо інший аспект. Якщо мінімально більшим за поточний виявився потоковий елемент (див. останню матрицю попереднього прикладу, останній стовпчик), то заміна біжучого елемента потоковим може бути не оптимальною, якщо попередні елементи вище потоку теж потокові і в попередніх стовпцях наверх спливають більші елементи:

$$\begin{pmatrix} 1^{\text{п}} & 1^{\text{п}} & 1^{\text{п}} & 1 \\ 2 & 3 & 4^{\text{п}} & 4^{\text{п}} \\ 4 & 5 & 5 & 2 \\ 3 & 4 & 1 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 2 & 3 & 4^n & 4^n \\ 1^n & 1^n & 1^n & 1 \\ 4 & 5 & 5 & 2 \\ 3 & 4 & 1 & 2 \end{pmatrix} \quad (3.30)$$

Останній варіант, розглянутий тут з сортуванням по останньому стовпчику.

Однак, легко бачити, що можна перевірити ще ситуації під потоком, де також можна збільшити матрицю. В наступному прикладі розділено окремо перестановку та сортування. Елемент 5 витісняє потоковий 4^n.

$$\begin{pmatrix} 1^{\text{п}} & 1^{\text{п}} & 1^{\text{п}} & 1 \\ 2 & 3 & 4^{\text{п}} & 4^{\text{п}} \\ 4 & 5 & 5 & 2 \\ 3 & 4 & 1 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 1^{\text{п}} & 1^{\text{п}} & 1^{\text{п}} & 1 \\ 2 & 3 & 5 & 2 \\ 4 & 5 & 4^{\text{п}} & 4^{\text{п}} \\ 3 & 4 & 1 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 1^n & 1^n & 1^n & 1 \\ 2 & 3 & 5 & 2 \\ 4 & 5 & 1 & 2 \\ 3 & 4 & 4^n & 4^n \end{pmatrix} \quad (3.31)$$

Бачимо, що отримали матрицю більшу за попередню, але суттєво меншу за отриману в попередньому прикладі. Таким чином, якщо першим більшим за поточний виявився потоковий елемент, необхідно перевіряти можливості збільшення в області матриці під потоком, а також здійснювати сортування в усій області під потоком у випадку, якщо все ж доводиться переставляти

поточний елемент. Остання позиція зміни у стовпцях та величина зміни мають бути мінімальними. Отримуємо:

$$\begin{pmatrix} 1^{\text{п}} & 1^{\text{п}} & 1^{\text{п}} & 2 \\ 2 & 3 & 4^{\text{п}} & 4^{\text{п}} \\ 4 & 5 & 1 & 2 \\ 3 & 4 & 5 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1^{\text{п}} & 1^{\text{п}} & 1^{\text{п}} & 1 \\ 2 & 3 & 4^{\text{п}} & 4^{\text{п}} \\ 4 & 5 & 5 & 2 \\ 3 & 4 & 1 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 1^{\text{п}} & 1^{\text{п}} & 1^{\text{п}} & 1 \\ 2 & 3 & 5 & 2 \\ 4 & 5 & 1 & 2 \\ 3 & 4 & 4^{\text{п}} & 4^{\text{п}} \end{pmatrix} \rightarrow$$

$$\begin{pmatrix} 1^{\text{п}} & 1^{\text{п}} & 1^{\text{п}} & 1 \\ 2 & 3 & 5 & 2 \\ 4 & 5 & 4^{\text{п}} & 4^{\text{п}} \\ 3 & 4 & 1 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 1^{\text{п}} & 1^{\text{п}} & 1^{\text{п}} & 1 \\ 2 & 3 & 4^{\text{п}} & 4^{\text{п}} \\ 4 & 5 & 5 & 2 \\ 3 & 4 & 1 & 2 \end{pmatrix} \rightarrow \begin{matrix} \text{сортування} \\ \text{під} \\ \text{потоком} \end{matrix} \rightarrow \begin{pmatrix} 1^{n} & 1^{n} & 1^{n} & 1 \\ 2 & 3 & 4^{n} & 4^{n} \\ 4 & 5 & 1 & 2 \\ 3 & 4 & 5 & 2 \end{pmatrix} \tag{3.32}$$

Розглянемо відношення порядку $\prec_2$. Тоді генерація найближчої більшої матриці здійснюється на основі наступного алгоритму.

1. Переглядаються усі рядки, починаючи з останнього, справа наліво і знаходиться перша ситуація, коли один елемент рядка менший за елемент стовпця, що знаходиться нижче (мінімальна ситуація, коли матрицю можна збільшити). Нехай $a_{ij} < a_{i+1j}$.

2. Знаходиться мінімальний елемент в біжучому стовпчику починаючи з позиції $i + 1$, який більший за елемент a_{ij}. Елемент a_{ij} міняється місцями, зі знайденим.

3. Всі елементи у біжучому стовпчику $a_{i+1j+1}, a_{i+2j+!}, \ldots, a_{n,j+1}$, а також у стовпцях $a^1, \ldots, a^n$ сортуються у порядку зростання, починаючи з позиції i.

Зауважимо, що при наявності потоків алгоритм має бути модифікованим.

Дійсно, якщо мінімально більшим виявився поточний елемент при перегляді справа наліво, то він може при заміні некоректно витісняти інші потокові елементи, потрібно перевіряти принцип мінімального індексу заміни.

$$\begin{pmatrix} 1^n & 1^n & 1^n & 2 \\ 2 & 3 & 4^n & 4^n \\ 4 & 5 & 3 & 2 \\ 3 & 4 & 1 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 2 & 3 & 4^n & 4^n \\ 1^n & 1^n & 1^n & 2 \\ 4 & 5 & 1 & 2 \\ 3 & 4 & 3 & 2 \end{pmatrix} \quad (3.33)$$

Тут індекс заміни – 11.

$$\begin{pmatrix} 1^n & 1^n & 1^n & 2 \\ 2 & 3 & 4^n & 4^n \\ 4 & 5 & 3 & 2 \\ 3 & 4 & 1 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 1^n & 1^n & 1^n & 2 \\ 2 & 5 & 1 & 2 \\ 4 & 3 & 3 & 2 \\ 3 & 4 & 4^n & 4^n \end{pmatrix} \quad (3.34)$$

Індекс заміни – 22. Індекс заміни має бути мінімальним при гарантованому збільшенні матриці. Якщо мінімально більшим виявився поточний елемент при перегляді справа наліво, то при витісненні меншого елемента справа можливе витіснення більшого зліва і заміна його меншим потоковим, що призводить до недопустимого зменшення матриці і зациклення. Після заміни мінімального елемента поточним можливе зменшення величини індикатора заміни, що тягне за собою можливість зациклення. Індикатор заміни – це елемент, положення якого максимально наближене до верхнього лівого кутка матриці в процесі замін. Індекс індикатора заміни – це положення в матриці.

$$\begin{pmatrix} 1^{\text{п}} & 1^{\text{п}} & 1^{\text{п}} & 2 \\ 2 & 3 & 5 & 1 \\ 4 & 5 & 4^{\text{п}} & 4^{\text{п}} \\ 3 & 4 & 1 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 1^n & 1^n & 1^n & 2 \\ 2 & 3 & 4^n & 4^n \\ 4 & 5 & 1 & 1 \\ 3 & 4 & 5 & 2 \end{pmatrix} \quad (3.35)$$

А тому необхідно відслідковувати зменшення індикатора заміни при пересуванні потокових елементів. Розглянемо приклад.

$$\begin{pmatrix} 1^{\text{п}} & 1^{\text{п}} & 1^{\text{п}} & 2 \\ 2 & 3 & 4^{\text{п}} & 4^{\text{п}} \\ 4 & 5 & 1 & 2 \\ 3 & 4 & 5 & 1 \end{pmatrix} \rightarrow 33 \rightarrow \begin{pmatrix} 1^{\text{п}} & 1^{\text{п}} & 1^{\text{п}} & 2 \\ 2 & 3 & 4^{\text{п}} & 4^{\text{п}} \\ 4 & 5 & 5 & 1 \\ 3 & 4 & 1 & 2 \end{pmatrix} \rightarrow 34 \rightarrow \begin{pmatrix} 1^{\text{п}} & 1^{\text{п}} & 1^{\text{п}} & 2 \\ 2 & 3 & 4^{\text{п}} & 4^{\text{п}} \\ 4 & 5 & 5 & 2 \\ 3 & 4 & 1 & 1 \end{pmatrix} \rightarrow 23$$

$$\rightarrow \begin{pmatrix} 1^{\text{п}} & 1^{\text{п}} & 1^{\text{п}} & 2 \\ 2 & 3 & 5 & 1 \\ 4 & 5 & 1 & 2 \\ 3 & 4 & 4^{\text{п}} & 4^{\text{п}} \end{pmatrix} \rightarrow 33 \rightarrow \begin{pmatrix} 1^{\text{п}} & 1^{\text{п}} & 1^{\text{п}} & 2 \\ 2 & 3 & 5 & 1 \\ 4 & 5 & 4^{\text{п}} & 4^{\text{п}} \\ 3 & 4 & 1 & 2 \end{pmatrix} \rightarrow 23! \rightarrow \begin{pmatrix} 1^{n} & 1^{n} & 1^{n} & 2 \\ 2 & 3 & 4^{n} & 4^{n} \\ 4 & 5 & 1 & 1 \\ 3 & 4 & 5 & 2 \end{pmatrix}$$

ПОМИЛКА

$$\begin{pmatrix} 1^{\text{п}} & 1^{\text{п}} & 1^{\text{п}} & 2 \\ 2 & 3 & 5 & 1 \\ 4 & 5 & 4^{\text{п}} & 4^{\text{п}} \\ 3 & 4 & 1 & 2 \end{pmatrix} \rightarrow 22 \rightarrow \begin{pmatrix} 1^{n} & 1^{n} & 1^{n} & 2 \\ 2 & 5 & 1 & 1 \\ 4 & 3 & 4^{n} & 4^{n} \\ 3 & 4 & 5 & 2 \end{pmatrix} \rightarrow 33 \text{ (був 4, зросло на 5)}$$

$$\rightarrow \begin{pmatrix} 1^{n} & 1^{n} & 1^{n} & 2 \\ 2 & 5 & 1 & 1 \\ 4 & 3 & 5 & 2 \\ 3 & 4 & 4^{n} & 4^{n} \end{pmatrix} \rightarrow 33! \quad \begin{matrix} \text{зменшення} \\ \text{зациклення} \\ \text{по мінімальному!} \end{matrix} \quad \rightarrow 32 \begin{pmatrix} 1^{\text{п}} & 1^{\text{п}} & 1^{\text{п}} & 2 \\ 2 & 5 & 1 & 1 \\ 4 & 4 & 4^{\text{п}} & 4^{\text{п}} \\ 3 & 3 & 5 & 2 \end{pmatrix} \rightarrow 23 \rightarrow$$

$$\begin{pmatrix} 1^{n} & 1^{n} & 1^{n} & 2 \\ 2 & 5 & 4^{n} & 4^{n} \\ 4 & 4 & 1 & 1 \\ 3 & 3 & 5 & 2 \end{pmatrix} \rightarrow 21 \rightarrow \ldots \qquad (3.36)$$

Таким чином, процес генерації матриць розкладу має вигляд:

$$\begin{pmatrix} 1^{\text{п}} & 1^{\text{п}} & 1^{\text{п}} & 2 \\ 2 & 3 & 4^{\text{п}} & 4^{\text{п}} \\ 4 & 5 & 1 & 2 \\ 3 & 4 & 5 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1^{\text{п}} & 1^{\text{п}} & 1^{\text{п}} & 2 \\ 2 & 3 & 4^{\text{п}} & 4^{\text{п}} \\ 4 & 5 & 5 & 1 \\ 3 & 4 & 1 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 1^{\text{п}} & 1^{\text{п}} & 1^{\text{п}} & 2 \\ 2 & 3 & 4^{\text{п}} & 4^{\text{п}} \\ 4 & 5 & 5 & 2 \\ 3 & 4 & 1 & 1 \end{pmatrix} \rightarrow$$

$$\begin{pmatrix} 1^{\text{п}} & 1^{\text{п}} & 1^{\text{п}} & 2 \\ 2 & 3 & 5 & 1 \\ 4 & 5 & 1 & 2 \\ 3 & 4 & 4^{\text{п}} & 4^{\text{п}} \end{pmatrix} \rightarrow \begin{pmatrix} 1^{\text{п}} & 1^{\text{п}} & 1^{\text{п}} & 2 \\ 2 & 3 & 5 & 1 \\ 4 & 5 & 4^{\text{п}} & 4^{\text{п}} \\ 3 & 4 & 1 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 1^{\text{п}} & 1^{\text{п}} & 1^{\text{п}} & 2 \\ 2 & 3 & 5 & 1 \\ 4 & 5 & 4^{\text{п}} & 4^{\text{п}} \\ 3 & 4 & 1 & 2 \end{pmatrix} \rightarrow$$

$$\begin{pmatrix} 1^{n} & 1^{n} & 1^{n} & 2 \\ 2 & 3 & 4^{n} & 4^{n} \\ 4 & 5 & 1 & 1 \\ 3 & 4 & 5 & 2 \end{pmatrix} \qquad (3.37)$$

Остаточний алгоритм виглядає так: генеруються усі можливі матриці розкладів у лексикографічному порядку та вибираються ті з них, де немає співпадінь по рядках. Таким чином, можуть бути згенеровані усі можливі

конфігурації розкладу. Також в процесі генерації можуть бути враховані додаткові умови.

Таким чином, запропоновано метод генерації матриць розкладів, що відрізняється використанням спеціальних відношень порядку, які гарантують отримання усіх можливих комбінацій розкладів. Дослідимо складність даного підходу та методу перманентної декомпозиції.

3.5. Дослідження ефективності методу перманентної декомпозиції у порівнянні з підходом на основі узагальненого відношення порядку для задач генерації комбінаторних об'єктів

Розглянемо детальніше ефективність методу перманентної декомпозиції у порівнянні з відомими підходами, наприклад, підходом на основі узагальненого відношення порядку для задач генерації комбінаторних об'єктів. Для отримання числових оцінок виберемо деяку тестову задачу, наприклад задачу генерації перестановок, та проаналізуємо кількість арифметичних операцій при застосуванні методу перманентної декомпозиції та відомих алгоритмів, наприклад, алгоритму Нараяни [158].

Зауважимо, що класи складності таких алгоритмів є відомими, однак тут нам доведеться здійснювати більш глибокий аналіз з точністю до константи, а не просто аналіз асимптотичного класу складності. При такому аналізі для отримання достовірного результату необхідно застосовувати аналогічну методику оцінювання в обох випадках.

Розглянемо детальніше алгоритм генерації всіх перестановок на основі лексикографічного порядку, відомий як підхід Нараями [158]. Як відомо, відношення лексикографічного порядку $\prec_L$ на множині рядків визначається наступним чином:

$$s = (s_1, s_2, \dots, s_m) \prec_L p = (p_1, p_2, \dots, p_m), \tag{3.38}$$

якщо

$$\exists i: s_i < p_i, s_k = p_k \; \forall 1 \leq k < i. \tag{3.39}$$

Знайдемо кількість арифметичних операцій. Введемо поняття точки обміну, після якої здійснюється сортування частини масиву. В такому випадку після досягнення точки обміну k точки змінюються наступним чином:

1 2 3 4	2	sort 1	3 1 2 4	2	sort 1
1 2 4 3	3	sort 2	3 1 4 2	3	sort 2
1 3 2 4	2	sort 1	3 2 1 4	2	sort 1
1 3 4 2	3	sort 2	3 2 4 1	3	sort 2
1 4 2 3	2	sort 1	3 4 1 2	2	sort 1
1 4 3 2	4	sort 3	3 4 2 1	4	sort 3
2 1 3 4	2	sort 1	4 1 2 3	2	sort 1
2 1 4 3	3	sort 2	4 1 3 2	3	sort 2
2 3 1 4	2	sort 1	4 2 1 3	2	sort 1
2 3 4 1	3	sort 2	4 2 3 1	3	sort 2
2 4 1 3	2	sort 1	4 3 1 2	2	sort 1
2 4 3 1	4	sort 3	4 3 2 1	end	sort 4

Тут справа записаний індекс першого елемента, який менший за правий. Аргумент sort – це кількість елементів підмасиву, що сортуються (після вибору мінімально більшого за поточний елемент, відповідного обміну та формування «хвоста» для сортування).

Як бачимо, послідовності в лексикографічному порядку можна розділити на n блоків (в прикладі по 6 послідовностей) в залежності від того, який елемент стоїть в послідовності першим. Тоді кількість сортувань, що містить n–1 елемент sort(n–1) рівне кількості елементів, що більші за елемент 1 в першій послідовності – таких рівно n–1. Розглянемо, скільки сортувань з

n–2 елементів (sort 2 у прикладі). Очевидно, що кожен з n послідовних блоків містить однакову кількість послідовностей, що вимагають сортувань n–2 елементів. Причому перша послідовність блоку є впорядкованою по зростанню. Тоді в процесі роботи алгоритму така послідовність породжуватиме n–2 послідовності, що вимагають n–2 сортування в процесі генерації (перший елемент з n використаний, далі повне сортування відбувається лише у випадку, коли другий елемент послідовності блоку замінюється на більший, а таких ситуацій n–2). Отже, маємо кількість послідовностей – n(n–2). Розглядаємо всі послідовності, що вимагають n–3 сортування. Маємо n блоків, в кожному блоці можна виділити підблоки в залежності від других елементів – їх можлива кількість n–1. А тоді кількість послідовностей, що вимагають сортування, що включає рівно n–3 елементи, рівна n–3 (третій елемент впорядкованої послідовності міняється на більший). Отже, маємо кількість n(n–1)(n–3). Продовжуючи міркування, отримуємо формулу для кількості операцій:

$$Q = (n-1)Qsort(n-1) + n(n-2)Qsort(n-2) + n(n-1)(n-3)Qsort(n-3) + \cdots + n(n-1)\ldots(n-k)(n-k-2)Qsort(n-k-2) + \cdots + n(n-1)(n-2)\ldots 3Qsort(1) + 1 = (n-1)Qsort(n-1) + \sum_{k=0}^{n-3}(\prod_{s=0}^{k}(n-s)) * (n-k-2)Qsort(n-k-2) + 1 \quad (3.40)$$

$Qsort$ – кількість арифметичних операцій при сортуванні + пошук мінімуму.

Якщо застосувати швидкий алгоритм сортування, то він має в середньому *nlogn* кількість порівнянь та nlogn/6 кількість обмінів (з коефіцієнтом 2ln2, Virth). Тоді:

$$Q = 2\ln 2((n-1)^2(ln(n-1)+1) + n(n-2)^2(ln(n-2)+1) + n(n-1)(n-3)^2 ln(n-3) + \cdots + n(n-1)\ldots(n-k)(n-k-2)^2(ln(n-k-2) +$$

$$1) + \cdots + n(n-1)(n-2)\ldots 3) + 1 \geq 2\ln 2(n(n-1) + n(n-1)(n-2) + n(n-1)(n-2)(n-3) + \cdots + n(n-1)(n-2)\ldots 3) = 2\ln 2 n!\left(\frac{1}{(n-2)\ldots 32} + \frac{1}{(n-3)\ldots 32} + \cdots + \frac{1}{2}\right) = 2\ln 2 n!\left(\frac{1}{(n-2)!} + \frac{1}{(n-3)!} + \cdots + \frac{1}{2!}\right) \approx n!\, 2\ln 2(e-2). \quad (3.41)$$

У попередніх оцінках використано, що

$$\begin{aligned} &(n-1)\,(ln(n-1)+1) > n, \\ &(n-1)\,ln(n-1) \geq 1, \\ &n \geq 2.763222834, \\ &\alpha \geq 1.278464543. \end{aligned} \quad (3.42)$$

Бачимо, що отримали клас складності $O(n!)$. Покажемо, що клас складності алгоритму практично не залежить від алгоритму сортування частини масиву. Виберемо, наприклад, обмінний алгоритм сортування фрагменту масиву, який має меншу ефективність у порівнянні зі швидкими методами. Тоді отримуємо:

$$\begin{aligned} &(n-1)^2(n-1) + n(n-2)^2(n-2) + n(n-1)(n-3)^2(n-3) + \cdots \\ &\quad + n(n-1)\ldots(n-k)(n-k-2)^2(n-k-2) + \cdots \\ &\quad + n(n-1)(n-2)\ldots 3 + 1 \\ &\quad \geq n(n-1)(n-2) + n(n-1)(n-2)(n-3) + \cdots \\ &\quad + n(n-1)\ldots(n-k)(n-k-1)\quad (n-k-2)(n-k-3) + \cdots \\ &+ n(n-1)(n-2)\ldots 3 + 1 = n!\left(\frac{1}{(n-3)!} + \frac{1}{(n-4)!} + \cdots + \frac{1}{2!}\right) \end{aligned}$$

$$(n-1)\,ln(n-1) \geq n$$

$$(n-1)\,(ln(n-1)+1) \geq 2n + 1 = 2(n-1) + 3$$

$$Q = 2\ln 2((n-1)^2(ln(n-1)+1) + n(n-2)^2(ln(n-2)+1)$$
$$+ n(n-1)(n-3)^2 ln(n-3) + \cdots$$
$$+ n(n-1)\dots(n-k)(n-k-2)^2(ln(n-k-2)+1) + \cdots$$
$$+ n(n-1)(n-2)\dots 3) + 1$$
$$\geq 2\ln 2\big((n-1)(2n+1) + n(n-1)(2n-1)$$
$$+ n(n-1)(n-2)(2n-3) + \cdots + n(n-1)(n-2)\dots 3\big) =$$
$$2\ln 2\big((n-1)(2n+1) + n(n-2)(2n-1) + n(n-1)(n-3)(2n-3) + \cdots$$
$$+ n(n-1)\dots(n-k)(n-k-2)\,(2(n-k-2)+3)$$
$$+ n(n-1)(n-2)3*1*5\big) =$$
$$= 2\ln 2\big((n-1)(n+n+1) + n(n-2)(n-1+n-1+1)$$
$$+ n(n-1)(n-3)(n-2+n-2+1) + \cdots$$
$$+ n(n-1)(n-2)\dots 3*1*5\big)$$
$$= 4\ln 2 n!\left(\frac{1}{(n-2)\dots 32} + \frac{1}{(n-3)\dots 32} + \cdots + \frac{1}{2} + 1\right) +$$
$$2\ln 2\big((n-1)(1) + n(n-2)(1) + n(n-1)(n-3)(1) + \cdots$$
$$+ n(n-1)(n-2)\dots 3\big)$$
$$= 4\ln 2 n!\left(\frac{1}{(n-2)!} + \frac{1}{(n-3)!} + \cdots + \frac{1}{2!} + 1\right)$$
$$+ 2\ln 2 n!\left(\frac{n-1}{(n)!} + \frac{n-2}{(n-1)!} + \frac{n-3}{(n-2)!}\dots + \frac{1}{2!}\right) =$$
$$\approx n!\,4\ln 2(e-1) + 2\ln 2 n!\left(\tfrac{1}{(n-1)!} + \tfrac{1}{(n-2)!} + \tfrac{1}{(n-3)!}\dots + \tfrac{1}{2!} - \tfrac{1}{(n)!} - \tfrac{1}{(n-1)!} =\right.$$
$$\left.\tfrac{1}{(n-2)!}\dots - \tfrac{1}{3!}\right) = n!\,4\ln 2(e-1) + 2\ln 2 n!\left(e - 2 - \left(e - 2 - \tfrac{1}{2}\right)\right) =$$
$$n!\,(4\ln 2(e-1) + \ln 2) \qquad (3.43)$$

Отримали аналогічний результат із попереднім, кількість арифметичних операцій відрізняється лише на константу.

Тепер розглянемо метод перманентної декомпозиції. Маємо формування n матриць, аналог множення–запис у список відповідного елемента, аналог

додавання–розбиття біжучої послідовності на дві частини, розгалуження. Арифметичні операції можуть бути відсутні. Але тоді матимемо копіювання елементів у списки. Отже, кількість арифметичних операцій може бути записана за допомогою наступного рекурентного співвідношення:

$$Q(n) = 2n - 1 + \mathrm{nQ(n-1)} = 2\mathrm{n} - 1 + \mathrm{n}\big(2(\mathrm{n}-1) - 1 + (\mathrm{n}-1)\mathrm{Q}(\mathrm{n}-2)\big)$$
$$= 2\mathrm{n} - 1 + 2\mathrm{n}(\mathrm{n}-1) - \mathrm{n} + \mathrm{n}(\mathrm{n}-1)\mathrm{Q}(\mathrm{n}-2) =$$
$$= 2\mathrm{n} - 1 + 2\mathrm{n}(\mathrm{n}-1) - \mathrm{n} + \mathrm{n}(\mathrm{n}-1)\big(2(\mathrm{n}-2) - 1 + (n-2)Q(n-3)\big) =$$
$$= 2\mathrm{n} - 1 + 2\mathrm{n}(\mathrm{n}-1) - \mathrm{n} + 2\mathrm{n}(\mathrm{n}-1)(n-2) - n(n-1) + \mathrm{n}(\mathrm{n}-1)(n-2)Q(n-3) = 2\mathrm{n} - 1 + 2\mathrm{n}(\mathrm{n}-1) - \mathrm{n} + 2\mathrm{n}(\mathrm{n}-1)(n-2) - n(n-1) + \mathrm{n}(\mathrm{n}-1)(n-2)\big(2(n-3) - 1 + (n-3)Q(n-4)\big) = 2\mathrm{n} - 1 + 2\mathrm{n}(\mathrm{n}-1) - \mathrm{n} + 2\mathrm{n}(\mathrm{n}-1)(n-2) - n(n-1) + 2\mathrm{n}(\mathrm{n}-1)(n-2)(n-3) - \mathrm{n}(\mathrm{n}-1)(n-2) + \mathrm{n}(\mathrm{n}-1)(n-2)(n-3)Q(n-4) = \cdots =$$
$$= 2\mathrm{n} + 2\mathrm{n}(\mathrm{n}-1) + 2\mathrm{n}(\mathrm{n}-1)(n-2) + \cdots + 2\mathrm{n}(\mathrm{n}-1)(n-2) *..* 2 - (1 + n + n(n-1) + n(n-1)(n-2) + \cdots + n(n-1)(n-2) *..* 3) + n! =$$
$$(n + n(n-1) + n(n-1)(n-2) + \cdots + n(n-1)(n-2) *..* 2) + n! + n! - 1 = n!\left(\frac{1}{(n-1)!} + \frac{1}{(n-2)!} + \cdots + \frac{1}{2!} + 1\right) + 2n! - 1 \approx n!\,(e+1) - 1. \qquad (3.44)$$

Тоді для порівняння складності обох методів маємо:

$$\frac{(4\ln 2(e-1) + \ln 2)}{e+1} = \frac{5.457}{3.718} = 1.46 \qquad (3.45)$$

Аналізуючи оцінки складності відповідних алгоритмів бачимо, що метод перманентної декомпозиції належить до того самого класу складності – *O(n!)*, що і метод, який базується на відношенні порядку.

Однак, незважаючи на однаковий порядок складності, маємо виграш у складності перманентного алгоритму (за рахунок мультиплікативної константи), особливо зважаючи на той факт, що при оцінці складності класичного підходу до генерації перестановок була використана оцінка знизу. Бачимо, що кількість арифметичних операцій методу перманентної декомпозиції менша від складності порядкового методу принаймні на 46%.

Зауважимо, що результатом застосування методу перманентної декомпозиції є множина СРПС. З метою побудови ефективних алгоритмів генерації матриць розкладів поняття матриць інцидентності та модифікованого перманента вимагають подальшого вдосконалення.

3.6. Модифікація матриці інцидентнтності та адитивно–диз'юнктивні форми

Нехай маємо довільну матрицю денного розкладу, розмірністю m x n. Розглянемо стовпчики матриці розкладу $R_1, R_2, \dots R_n$. Як вже розглядалось вище, модифікована матриця інцидентності при наявності потоків будувалась наступним чином.

По горизонталі зображуються номера викладачів, по вертикалі – групи, у яких проводяться заняття. Кожному викладачу ставиться у відповідність стовпчик матриці, у якому записуються нулі та одиничі в залежності від того, чи має викладач пари у відповідних групах.

Причому, якщо якийсь викладач x має поточну пару, то виділяємо йому окремий стовпчик матриці інцидентності, позначивши його x^p (можна використовувати індекс, що є кількістю елементів потоку).

Таким чином, утворюється матриця виду:

$$A = \begin{array}{c} \\ R_1 \\ R_2 \\ \dots \\ R_m \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & \dots & x_n \end{array} \\ \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \end{array}, \qquad (3.46)$$

$$\alpha_{ij} = \begin{cases} 1, & \text{коли } x_i \in R_j, \\ 0 \text{ в інакшому випадку.} \end{cases} \tag{3.47}$$

Зауважимо, що така матриця інцидентності не дає вичерпну інформацію для побудови розкладу.

Адже у випадку, коли викладач має декілька пар в одній і тій самій групі, чи декілька аналогічних поточних пар, в матриці все одно стоятимуть 1. А тому розглянемо модифікацію матриці інцидентності, що містить інформацію про кількість пар. Для цього вкажемо замість 1 відповідне число. Отже,

$$\alpha_{ij} = \begin{cases} k_i, & \text{коли } x_i \in R_j, \text{. кількість пар } k_i \\ & 0 \text{ в інакшому випадку.} \end{cases} \tag{3.48}$$

Наприклад, для матриці розкладів виду

$$\begin{pmatrix} 1 & 1 & 1 & 2 & 3 & 4 \\ 4 & 3 & 1 & 2 & 3 & 4 \\ 3 & 1 & 4 & 5 & 2 & 4 \end{pmatrix}, \tag{3.49}$$

отримуємо наступну матрицю інцидентності:

$$\begin{pmatrix} & 1 & 1^p & 2 & 3 & 4 & 4^p & 5 \\ R_1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ R_2 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ R_3 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ R_4 & 0 & 0 & 2 & 0 & 0 & 0 & 1 \\ R_5 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ R_6 & 0 & 0 & 0 & 0 & 2 & 1 & 0 \end{pmatrix}, \tag{3.50}$$

де 1^p та 4^p – індекси відповідних викладачів, що мають «поточні» пари.

Нехай, наприклад, маємо матрицю розкладу виду:

$$R = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 3 & 1 \\ 1 & 3 & 4 \end{pmatrix} \tag{3.51}$$

Тоді матриця інцидентності має вигляд:

$$\begin{pmatrix} & 1 & 1^{n} & 2 & 3 & 4 \\ R_1 & 1 & 1 & 1 & 0 & 0 \\ R_2 & 0 & 1 & 0 & 2 & 0 \\ R_3 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \quad (3.52)$$

Побудуємо процес розкладу модифікованого перманента з «запам'ятовуванням» за першим рядком:

$$per\mathrm{mod}_{2\,3}^{1}\begin{pmatrix} 1 & 1^{n} & 2 & 3 & 4 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 2 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix} = 1_1^1 * per\mathrm{mod}_{3}^{2}\begin{pmatrix} 2 & 3 & 4 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} + 1_1^{1^n} * 1 + 1_1^2 *$$

$$per\mathrm{mod}_{3}^{2}\begin{pmatrix} 1 & 3 & 4 \\ 0 & 2 & 0 \\ 1 & 0 & 1 \end{pmatrix} = 1_1^1 * (2_2^3 per\mathrm{mod}_3 \binom{24}{01}) + 1_1^{1^n} * 1 + 1_1^2 *$$

$$(2_2^3 per\mathrm{mod}_3 \binom{14}{11}) = 1_1^1 2_2^3 1_3^4 + 1_1^{1^n} + 1_1^2 2_2^3 1_3^1 + 1_1^2 2_2^3 1_3^4. \quad (3.53)$$

Розглянемо детальніше процедуру розкладу перманента. Зауважимо, що ненульовий елемент, що стоїть у рядку матриці на першому кроці означає обов'язкову присутність відповідного компонента у розкладі. В першій групі обов'язково мають бути присутніми елементи 1, 2, 1^n. Далі в процесі розкладу здійснюється розгалуження і виникають різні ситуації. В процесі обрахунку перманента використовуємо операцію додавання +. Однак, з точки зору побудови розкладу логіка операції додавання тут зовсім інша, вона фактично позначає «диз'юнкцію», можливість вибору одного з варіантів СРПС. Якщо множиться елемент, значення якого більше за 1, то можлива ситуація включення кількох компонент. Так, наприклад у виразі $2_2^3 per\mathrm{mod}_3 \binom{14}{11} = 2_2^3 * 1_3^1 + 2_2^3 * 1_3^4$ додавання означає обов'язкове включення обох компонент, оскільки 3 включається двічі, елемент матриці інцидентності рівний 2. В той

же час у виразі $1_2^3 per \mathrm{mod}_3 \binom{14}{11} = 1_2^3 * 1_3^1 + 1_2^3 * 1_3^4$ додавання означає вибір одного з двох варіантів, оскільки елемент 3 використовується лише 1 раз. Надалі операцію вибору позначатимемо значком $\vee$ та називатимемо диз'юнктивною операцією (диз'юнкцією), а операцію обов'язкового включення + називатимемо адитивною операцією.

Отже, наведені вище приклади показали, що у процесі перманентної декомпозиції модифікованої матриці інцидентності недостатньо однієї бінарної операції додавання, яка, фактично, раніше відігравала двояку роль: з точки зору обрахунку перманента вона була звичайним додаванням, однак з точки зору СРПС її зміст – включення, додавання деякої СРПС у загальну сукупність. Якщо маємо модифіковану матрицю, що містить елементи, які перевищують 1, слід використовувати операцію включення (адитивну операцію) або вибору (диз'юнктивну операцію) в процесі формування СРПС в залежності від величини біжучого елемента матриці інцидентності. Ці операції є логічними з точки зору формування різних варіантів матриць розкладів. Дійсно, будь-яка матриця розкладу є сукупністю рядків (СРПС), яка може бути утворена послідовним застосуванням операції включення в матрицю окремих рядків. В той же час, коли є декілька варіантів матриць розкладів, то цю різноваріантність можна записати з використанням операції вибору.

Очевидно, що якщо побудувати певну множину елементів та операції на цій множині (результат операцій належить множині), то за умови відсутності предикатів отримаємо алгебраїчну систему, яку називають алгеброю [11], а сама множина є її носієм. Для коректної побудови алгебри, що містить операції включення та вибору введемо формальне визначення розкладу.

Нехай маємо n різних елементів $(a_1, a_2, \ldots, a_n)$, що утворюють множини $(W_1, W_2, \ldots, W_m)$, причому елементи можуть у множинах повторюватись. Інформацію про те, які елементи утворюють кожну з множин, зручно записати у вигляді матриці інцидентності виду:

$$\begin{pmatrix} & a_1 & a_2 & \dots & a_n \\ W_1 & n_{11} & n_{12} & \dots & n_{1n} \\ W_2 & n_{21} & n_{22} & \dots & n_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ W_m & n_{m1} & n_{m2} & \dots & n_{mn} \end{pmatrix}, \quad (3.54)$$

де n_{ij}–кількість елементів a_j у множині W_i.

Елементи $(a_1, a_2, \dots, a_n)$ – ідентифікатори стовпців матриці інцидентності. Системою представників (СП) множин $(W_1, W_2, \dots, W_m)$ є вектор виду:

$$(v_1, v_2, \dots, v_m), v_i \in W_i, i = \overline{1, m}. \quad (3.55)$$

З врахуванням специфіки задачі складання розкладу занять, будемо розділяти усі елементи на звичайні та поточні. Поточним будемо називати такий елемент, присутність якого є обов'язковою на визначених позиціях в СП. Очевидно, що інформація про усі можливі потоки має бути заданою. Якщо її записати за допомогою матриці інцидентності, то будемо для поточних елементів виділяти окремі стовпчики (приклади наведені в попередніх розділах):

$$\begin{pmatrix} & a_1 & a_1^p & \dots & a_n & a_n^p \\ W_1 & n_{11} & n_{11}^p & \dots & n_{1n} & n_{1n}^p \\ W_2 & n_{21} & n_{21}^p & \dots & n_{2n} & n_{2n}^p \\ \dots & \dots & \dots & \dots & \dots & \dots \\ W_m & n_{m1} & n_{11}^p & \dots & n_{mn} & n_{mn}^p \end{pmatrix} \quad (3.56)$$

Системою різних представників (СРП) множин $(W_1, W_2, \dots, W_m)$ будемо називати вектор виду: $(v_1, v_2, \dots, v_m)$, $v_i \in W_i$, $i = \overline{1, m}$, та виконується умова: $v_i \neq v_j, i \neq j$, якщо елементи v_i, v_j не належать одному потоку.

Розкладом будемо називати довільний вектор, елементами якого є СРП.

Розклад

$$R = ((v_{11}, v_{12}, \dots, v_{1m}), (v_{21}, v_{22}, \dots, v_{2m}), \dots, (v_{k1}, v_{k2}, \dots, v_{km})) \quad (3.57)$$

будемо називати коректним, якщо виконуються умови:

1. $\forall j \in \{1,2,\dots,m\}: \{v_{1j} \cup v_{2j} \cup \dots \cup v_{kj}\} = \{n_{j1} * a_1 \cup n_{j2} * a_2 \cup \dots \cup n_{jn} * a_n\}, l * a = \{a_1, a_2, \dots, a_l\}, a_i = a, i = \overline{1,l}.$

2. $\forall i \in \{1,2,\dots,k\}: v_{ij} \neq v_{ir}, j \neq r.$, якщо v_{ij}, v_{ir} є непотоковими елементами.

Як бачимо, наведене вище поняття розкладу збіжне з поняттям матриці розкладу, що використовували раніше, якщо всі СРП розглядати як рядки матриці. Очевидно, що бінарна операція включення (додавання) має застосовуватись як мінімум до СРП, що є розкладом згідно визначення.

В такому випадку, розклад

$$R = ((v_{11}, v_{12}, \dots, v_{1m}), (v_{21}, v_{22}, \dots, v_{2m}), \dots, (v_{k1}, v_{k2}, \dots, v_{km})) \quad (3.58)$$

можемо записати як «суму» СРП:

$$R = (v_{11}, v_{12}, \dots, v_{1m}) + (v_{21}, v_{22}, \dots, v_{2m}) + \dots + (v_{k1}, v_{k2}, \dots, v_{km}) \quad (3.59)$$

та дати його визначення через операцію включення.

Адитивно–диз'юнктивною формою будемо називати довільний вираз, що може містити розклади, які коректно поєднані бінарними адитивною чи диз'юнктивною операціями та дужки, які змінюють порядок виконання операцій.

За аналогією з булевою алгеброю, елементарною адитивною формою будемо називати суму СРП, елементарною диз'юнктивною формою

називатимемо диз'юнкцію СРП. Канонічною АДФ будемо називати диз'юнкцію розкладів.

Таким чином, довільна СРП є АДФ, довільний розклад є АДФ. Отже, множина АДФ та описані вище операції включення та диз'юнкції утворюють алгебру, $\boldsymbol{A}(W_1, W_2, \ldots, W_m) = \{A, +, \vee\}$, де A – множина всіх АДФ, породжена системою множин $(W_1, W_2, \ldots, W_m)$ за допомогою методу, що описаний вище.

Нехай $a, b \in A$ – адитивно–диз'юнктивні форми. Тоді мають місце такі властивості:

1) $a \vee a = a$;
2) $a \vee b = b \vee a$;
3) $a + b \neq b + a$, a, b – різні;
4) $a + a \in A$;
5) $a + b \vee c = (a + b) \vee (a + c)$.

Як бачимо, операція вибору є комутативною, адитивна операція – не комутативна. Операція вибору має більший пріоритет. Очевидно, що на основі записаних аксіом можна будувати безліч властивостей. Але зазначених вище властивостей тут цілком достатньо для розробки алгоритмів та інформаційної технології формування матриць розкладів. Окрім того, зауважимо, що для даної задачі необхідно побудувати канонічну АДФ, що містить всі коректні варіанти розкладів. Тоді в ході подальшого аналізу слід відкидати варіанти розкладів, що не задовольняють відповідні додаткові вимоги.

Отже,

$$1_2^3 per \mathrm{mod}_3 \binom{14}{11} = 1_2^3 * 1_3^1 \ \vee 1_2^3 * 1_3^4 \tag{3.60}$$

$$2_2^3 per \mathrm{mod}_3 \binom{14}{11} = 1_2^3 * 1_3^1 + 1_2^3 * 1_3^4 \tag{3.61}$$

$$1_2^3 per\mathrm{mod}_3\binom{14\ 5}{11\ 1} = 1_2^3 * 1_3^1 + 1_2^3 * 1_3^4 + 1_2^3 * 1_3^5 = АДФ = 1_2^3 * 1_3^1 \vee 1_2^3 * 1_3^4 \vee 1_2^3 * 1_3^5 \quad (3.62)$$

$$2_2^3 per\mathrm{mod}_3\binom{14\ 5}{11\ 1} = 2_2^3 * 1_3^1 + 2_2^3 * 1_3^4 + 2_2^3 * 1_3^5 = АДФ == (1_2^3 * 1_3^1 + 1_2^3 * 1_3^4) \vee (1_2^3 * 1_3^1 + 1_2^3 * 1_3^5) \vee (1_2^3 * 1_3^4 + 1_2^3 * 1_3^5) \quad (3.63)$$

$$3_2^3 per\mathrm{mod}_3\binom{14\ 5}{11\ 1} = 3_2^3 * 1_3^1 + 3_2^3 * 1_3^4 + 3_2^3 * 1_3^5 = АДФ = 1_2^3 * 1_3^1 + 1_2^3 * 1_3^4 + 1_2^3 * 1_3^5 \quad (3.64)$$

Окрім того, при використанні для обов'язкового включення відповідного елемента стовпця, якому відповідає ненульовий елемент матриці інцидентності, будемо зменшувати значення елемента матриці інцидентності на 1.

Отже, значення елемента матриці інцидентності рівне кількості включень відповідного індексу у підсумковій АДФ у виразах обов'язкового включення.

Отже у випадку, коли значення елемента–множника менше за кількість ненульових елементів рядка розкладу, маємо C_n^k-варіантів інтерпретації початкової суми у вигляді АДФ.

Тепер розглянемо попередній приклад з точки зору АДФ. Можемо модифікувати процедуру розкладу перманента так, щоб отримати коректну АДФ. Для цього на кожній ітерації декомпозиції перманента аналізуватимемо входження однакових елементів у всі складові обов'язкового включення. Очевидно, що має виконуватись умова – загальна кількість входжень елемента в блоці обов'язкового включення має дорівнювати його індексу.

Якщо виявиться, що ця умова порушена, то блок обов'язкового включення вважатиметься некоректним і має бути видаленим.

$$per\mathrm{mod}_{3}^{1,2}\begin{pmatrix} 1 & 1^{n} & 2 & 3 & 4 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 2 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix} =$$

$$=1_1^1 * per\mathrm{mod}_{3}^{2}\begin{pmatrix} 2 & 3 & 4 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} +1_1^{1^{\text{п}}} * 1+1_1^2 * per\mathrm{mod}_{3}^{2}\begin{pmatrix} 1 & 3 & 4 \\ 0 & 2 & 0 \\ 1 & 0 & 1 \end{pmatrix} =1_1^1 *$$

$$(2_2^3 per\mathrm{mod}_3\binom{24}{01}) +1_1^{1^{\text{п}}} * 1+1_1^2 * (2_2^3 per\mathrm{mod}_3\binom{14}{11}) =\text{АДФ}=1_1^1 *$$

$$(1_2^3 per\mathrm{mod}_3\binom{24}{01}) +1_1^{1^{\text{п}}} * 1+1_1^2 * (1_2^3 per\mathrm{mod}_3\binom{14}{11}) =1_1^1 1_2^3 1_3^4 + 1_1^{1^n} +$$

$$1_1^2 1_2^3 1_3^1 \vee 1_1^2 1_2^3 0_3^4. \qquad (3.65)$$

В даному прикладі у процесі побудови АДФ врахована властивість, що, якщо елемент розкладу вже використаний у виразі обов'язкового включення і він присутній у інших виразах обов'язкового включення, то його інцидентне значення у наступних виразах зменшується на 1.

Таким чином, модифікація процедури розкладу перманента матриці інцидентності шляхом побудови АДФ дозволяє отримати усі можливі коректні варіанти розкладів.

У запропонованій процедурі вирішується не лише проблема співпадіння елементів розкладу у рядках, але і зразу ж отримуємо необхідну кількість однакових елементів у стовпцях у відповідності до вхідних даних, записаних в модифікованій матриці інцидентності.

Серед особливостей програмного забезпечення зауважимо, що при відсутності ситуацій, коли викладач може мати кілька пар в одній групі в процесі побудови АДФ отримуємо АДФ виду:

$$(ДФ)+(ДФ)+\ldots+(ДФ), \tag{3.66}$$

де

$$ДФ = 1_1^{i_1} * 1_2^{i_2} *\ldots* 1_n^{i_n} \vee 1_1^{j_1} * 1_2^{j_2} *\ldots* 1_n^{j_n} \vee \ldots \ . \tag{3.67}$$

У випадку, коли допускається кілька пар в одній групі чи потоках отримуватимемо адитивні конструкції обов'язкового включення (ДАФ)+(ДАФ)+…+(ДАФ), де ДАФ містить конструкції виду:

$$1_1^{i_1} * 1_2^{i_2} * \ldots * 1_n^{i_n} \vee 1_1^{j_1} * 1_2^{j_2} * \ldots * 1_n^{j_n} \vee \left(1_1^{j_1} * 1_2^{j_2} * \ldots * 1_n^{j_n} \vee \ldots\right) +$$
$$+(1_1^{j_1} * 1_2^{j_2} *\ldots* 1_n^{j_n} \vee\ldots) \ldots \tag{3.68}$$

Побудову АДФ будемо здійснювати на базі множини СРПС. Дійсно, якщо значення індексу інцидентності деякого елемента в СРПС більше 1 і рівне k, то в такому випадку матимемо C_l^k варіантів інтерпретації початкової суми у вигляді АДФ.

Причому будь-яка комбінація СРПС з k елементів записуватиметься з операцією обов'язкового включення, між якими стоятиме операція вибору. Така процедура дозволить побудувати АДФ на основі множини усіх СРПС.

3.7. Алгоритм АДФ та загальний алгоритм формування матриці розкладу

Розглянемо алгоритми побудови АДФ детальніше.

Спочатку операція обов'язкового включення ставиться між групами СРПС, що утворені в результаті розкладу за першим рядком матриці інцидентності. Детальніше, групи обов'язкового включення – це всі СРПС, перший елемент в яких є елементом, якому відповідає ненульовий елемент першого рядка матриці інцидентності. Якщо індекс елемента є 1, то в групі стоятимуть всі диз'юнкції, якщо ні – то матимемо диз'юнкції всіх можливих

комбінацій кон'юнктивних блоків обов'язкового включення. Наприклад, якщо індекс є 2, матимемо початкові групи обов'язкового включення. Далі переглядатимемо усі СРПС послідовно по перших елементах (поточний елемент вважається одним), других і т.д.

Якщо перший елемент в якійсь групі має індекс 1, то, очевидно, що всі СРПС в групі використовуються із операцією вибору.

Якщо ж його індекс більший, то матимемо всі можливі комбінації блоків обов'язкового включення з'єднаних операцією вибору. Зауважимо, що результатом такого розкладу є сума диз'юнктивних форм, що гарантовано включає усі необхідні елементи першого стовпця. Далі розглядаємо всі СРПС по елементу, що стоїть другим. Логіка подальших дій – забезпечення гарантованого досягнення необхідної кількості включень другого елемента.

Для цього розглядаємо другий рядок матриці інцидентності та усі його ненульові елементи. Розглядаємо перший елемент. Визначаємо, у яких СРПС він є. Тоді визначаємо можливі варіанти так, щоб він входив у залежності від індексу. Для цього нумеруємо входження, генеруємо комбінації та перевіряємо необхідну кількість входжень з врахуванням операцій обов'язковості включень.

Оскільки в кожній групі лише диз'юнкції, то при індексі, більшому за 1, необхідно вибирати з кількох груп (з однієї лише при співпадінні). Тоді можна вибрати групи, з яких вибирається відповідна СРПС і тоді вона вибирається єдиною. При цьому з інших груп вибираються усі інші СРПС, які не містять даного елемента. Під новими групами ставимо диз'юнкцію. Таким чином, отримуємо диз'юнктивну форму груп, в кожній з яких є відповідні адитивні конструкції обов'язкового включення. А далі для інших елементів окремо працюємо аналогічно для кожної групи, бо входження для кожної групи має бути однакова кількість.

Припустимо, що його індекс більший. Зауважимо, що СРПС з цим елементом може бути в кількох групах обов'язкового включення (див.

приклад). Якщо стоїть лише в одній групі, де скрізь операції вибору, скласти розклад неможливо. Якщо стоїть в кількох різних групах, то з кожної такої групи можна вибрати лише 1 СРПС. Якщо є індексні елементи і можна сформувати розклад з необхідною кількістю входжень даного елемента, то формуються комбінації з СРПС з операціями обов'язкового включення. При цьому з'являтимуться СРПС обов'язкового включення. Далі розглядаємо третій елемент, четвертий тощо.

Отже, покроково опишемо АДФ-лгоритм:

1. Формування початкового загального блоку

1.1. Переглядаємо перший рядок матриці інцидентності та знаходимо всі ненульові елементи та їх ідентифікатори (відповідні елементи нульового рядка матриці інцидентності).

1.2.Список СРПС розбивається на групи, кожна СПРС з одної групи містить першими однакові елементи-ідентифікатори. Між групами ставиться знак обов'язкового включення + включення.

1.3. Якщо індекс відповідного елемента першого рядка 1, то між усіма елементами групи ставиться операція вибору. Інакше формуються усі можливі комбінації з елементів групи, що з'єднані операцією +, кількість СРПС в яких рівна індексу елемента, між якими ставиться операція вибору.

Зауважимо, що результатом такого розкладу є сума диз'юнктивних форм, що гарантовано коректно включає усі необхідні елементи першого стовпця.

2. Аналіз диз'юнктивних форм та формування АДФ

2.1. Розглядаємо другий рядок матриці інцидентності, аналогічно визначаємо ненульові елементи та їх ідентифікатори.

Розглянемо довільний ненульовий елемент та будемо забезпечувати його коректне входження у початкову АДФ в залежності від індексу.

2.2. Для кожного ненульового елемента

2.2.1. Визначаємо, у яких СРПС списку він є. *Тоді визначаємо усі можливі варіанти так, щоб він входив в залежності від його індексу.* Для цього нумеруємо всі СРПС, де другим є даний поточний елемент, розглядаємо всі можливі комбінації з СРПС, що містять другий елемент та кількість яких рівна його індексу та перевіряємо необхідну кількість його входжень із врахуванням операцій обов'язковості включень (генеруємо допоміжний масив комбінацій в лексикографічному порядку). Комбінація є коректною, якщо кількість входжень рівна індексу.

2.2.2. Для кожної вибраної коректної комбінації утворюємо новий загальний блок, що будується наступним способом: з кожного диз'юнктивного блоку, що містить СРПС з нашим другим елементом, включаємо лише обрані в даній комбінації СРПС, а усі інші, що містять другим наш елемент, видаляються.

3. Далі аналогічно аналізується третій рядок, четвертий і т.д. Причому умова коректного входження кожного елемента аналізується для кожного блоку обов'язкового включення.

Приклад 1:

1) Аналіз першого рядка матриці інцидентності:

$$\{[(1,3,4)\,]+[(1,1,1)]+\,[\,(2,3,1)\vee(2,3,4)]\} \qquad (3.69)$$

2) Аналіз другого рядка матриці інцидентності:

$$1^{\pi}:\{[(1,3,4)]+[(1,1,1)]+\,[\,(2,3,1)\vee(2,3,4)]\} \qquad (3.70)$$

$$3:\ \{[(1,3,4)\,]+[(1,1,1)]+\,[\,(2,3,1)]\}\ \vee\ \{[(1,3,4)\,]+[(1,1,1)]+\,[\,(2,3,4)]\} \qquad (3.71)$$

3) Аналіз третього рядка матриці інцидентності:

$$1:\ \{[(1,3,4)\,]+[(1,1,1)]+\,[\,(2,3,1)]\}\ \vee\ \{[(1,3,4)\,]+[(1,1,1)]+\,[\,(2,3,4)]\} \qquad (3.72)$$

$$1^n:\{[(1,3,4)\,]+[(1,1,1)]+[\,(2,3,1)]\}\vee\{[(1,3,4)\,]+[(1,1,1)]+[\,(2,3,4)]\} \quad (3.73)$$

$$4:\ \{[(1,3,4)\,]+[(1,1,1)]+[\,(2,3,1)]\} \quad (3.74)$$

Приклад 2:

$$\begin{gathered}(1_1^1 2_2^3 1_3^4)+(1_1^{1^n})+(1_1^2 2_2^3 1_3^1+1_1^2 2_2^3 1_3^4)=\\(1_1^1 2_2^3 1_3^4)+(1_1^{1^n})+(1_1^2 2_2^3 1_3^1\vee 1_1^2 2_2^3 1_3^4)=2елементи=\\(1_1^1 2_2^3 1_3^4)+(1_1^{1^n})+(1_1^2 2_2^3 1_3^1\vee 1_1^2 2_2^3 1_3^4)=(\\(1_1^1 2_2^3 1_3^4)+(1_1^{1^n})+1_1^2 2_2^3 1_3^1)\vee(\\(1_1^1 2_2^3 1_3^4)+(1_1^{1^n})+1_1^2 2_2^3 1_3^4)=\\=3елементи=(\\(1_1^1 2_2^3 1_3^4)+(1_1^{1^n})+1_1^2 2_2^3 1_3^1)=\\=(\\(1_1^1 2_2^3 1_3^4)+(1_1^{1^n})+1_1^2 2_2^3 1_3^1)\end{gathered} \quad (3.75)$$

Якщо в блоці обов'язкового включення більше елементів, ніж їх індекс, то такий блок видаляється.

Таким чином, модифікація матриці інцидентності, що відрізняється зберіганням вичерпної інформації про розклад, а також процедури перманентної декомпозиції призводить до необхідності виникнення алгебри адитивно-диз'юнктивних форм. На базі такої алгебри запропоновано алгоритми формування матриць розкладів на основі множини СРПС чи безпосередньо в процесі декомпозиції.

3.8. Інформаційна технологія складання розкладів на основі алгебри АДФ

Як бачимо з попереднього розділу, результатом застосування методу перманентної декомпозиції є сукупність комбінаторних об'єктів – перестановок, комбінацій, розміщень. Для задачі складання розкладів у

частині формування матриць розкладів метод дає записану в пам'ять сукупність усіх можливих систем різних представників множин, що являють собою стовпчики матриць розкладів (СРПС). Оскільки алгоритм перманентної декомпозиції дає всі можливі СРПС, то це породжує проблему формування остаточного розкладу на основі СРПС чи усіх можливих варіантів розкладів та вимагає розробки спеціальних алгоритмів. Окремі підходи до розв'язання такої задачі розглянуті в попередньому розділі, однак вони пов'язані зі значною обчислювальною складністю у загальному випадку. Це стосується і підходу на основі відношення порядку на множині матриць розкладу.

В основі інформаційної технології, що пропонується у даному розділі, є подальша модифікація матриць інцидентності та, відповідно, така модифікація методу перманентної декомпозиції, яка, зокрема, дозволяє на виході генерувати готові варіанти матриць розкладів. Описані вище результати дозволяють деталізувати інформаційну технологію для розв'язання широкого кола задач календарного планування, у основі якої покладено використання алгоритмів, що базуються на процедурах декомпозиції модифікованих спеціальним чином перманент матриць інцидентності.

Дана технологія представляє собою сукупність методів та підходів, які включають:

– методи аналізу вхідних даних;

– методи генерації різноманітних комбінаторних об'єктів – перестановок, систем різних представників стовпців, комбінацій з низкою додаткових умов, які ґрунтуються на основі спеціально розроблених алгоритмів декомпозиції перманент;

– методи формування усіх можливих допустимих варіантів матриць;

– розкладів, що включають алгоритмічні рішення, які базуються, зокрема, на використанні спеціально розробленої для таких задач алгебри адитивно–диз'юнктивних форм;

– методи представлення даних в процесі роботи алгоритмів генерації, які адаптовані до цих алгоритмів та максимально враховують їх специфіку.

Таким чином, дана інформаційна технологія являє собою цілісну систему, що дозволяє вирішувати задачі формування розкладів за наявності низки додаткових умов. Методи аналізу вхідних даних, зокрема, допустимості матриць розкладів, як і у випадку базового методу перманентної декомпозиції, базуються в основному на понятті конфігурації – сукупностей взаємопов'язаних елементів матриці розкладів, які стосуються одного виконавця (викладача, тощо). При цьому розглядаються тернарні, бінарні та унарні конфігурації, використовується низка властивостей, які мають, наприклад, матриці, що складаються лише з тернарних конфігурацій, лише з бінарних конфігурацій та поєднання різних конфігурацій. Важливе місце мають твердження про можливість оптимізації за критерієм, що мінімізує кількість «вікон» викладачів та студентів з точки зору створення системи, яка максимально враховує побажання усіх стейкхолдерів. Отримані теоретичні результати дозволяють здійснювати глибокий аналіз вхідних даних, здійснювати висновок про недопустимість матриці розкладів чи неможливість її оптимізації за вибраними критеріями, а також пропонувати різні алгоритмічні рішення, що стосуються формування матриць розкладів.

Зауважимо, що саме ефективна організація структур даних призвела до використання перманента. На основі вхідних даних формується матриця інцидентності. Виходячи з концепції мінімізації структур вхідних даних, матриця інцидентності чи початкова матриця розкладу мала б представляти вичерпну інформацію про розклад.

Однак, вона не дає такої інформації за умов наявності потоків. Проблема допустимості потоків вимагає зберігання додаткової інформації про потоки, їх структуру та кількість. Використання бази даних в цьому випадку, наприклад, вимагатиме постійного звертання до неї у процесі реалізації алгоритму, що є дуже затратним з точки зору швидкодії – задача і так є NP–повною. Наявність

потоків породжує фундаментальне протиріччя: з одного боку умовою допустимості є неможливість присутності двох однакових елементів у рядку матриці розкладів, з другого боку – можливість такої присутності для потоків. Розв'язати таке протиріччя вдається шляхом певної модифікації матриці інцидентності та введення додаткових стовпців в матрицю інцидентності для поточних елементів. Відповідно, зазнав модифікації і алгоритм декомпозиції, як описано вище.

Таким чином, у нас виникає досить важливий момент – додаткова інформація про вимоги до розкладу враховується на етапі формування матриці інцидентності чи алгоритму декомпозиції перманента недопустимі з точки зору ситуації, відкидаються (не генеруються) в процесі декомпозиції (рисунок 3.3.).

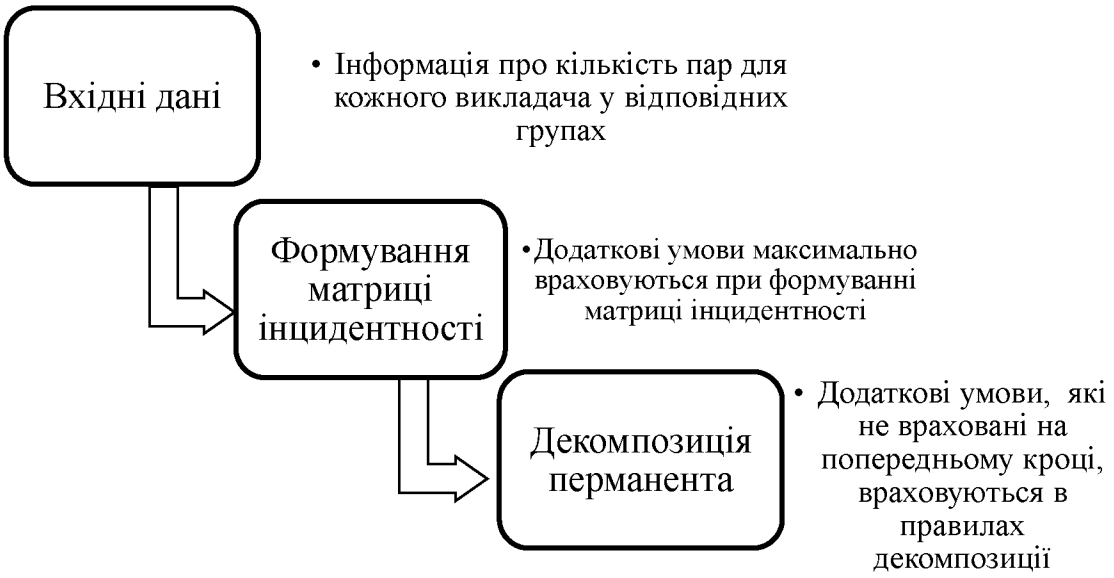

Рис.3.3. Ілюстрація врахування додаткових умов до розкладу на етапі формування матриць інцидентності та процедури декомпозиції перманента

Такий підхід із врахуванням самих оптимізованих процедур прямого запису в пам'ять дозволить збільшити швидкодію програмного комплексу при наявності низки додаткових умов. Проблема остаточної генерації розкладу на основі системи СРПС вимагала розробки принципово нових підходів.

Вирішення цієї проблеми пропонується на основі застосування алгебри адитивно–диз'юнктивних форм (рисунок 3.4.).

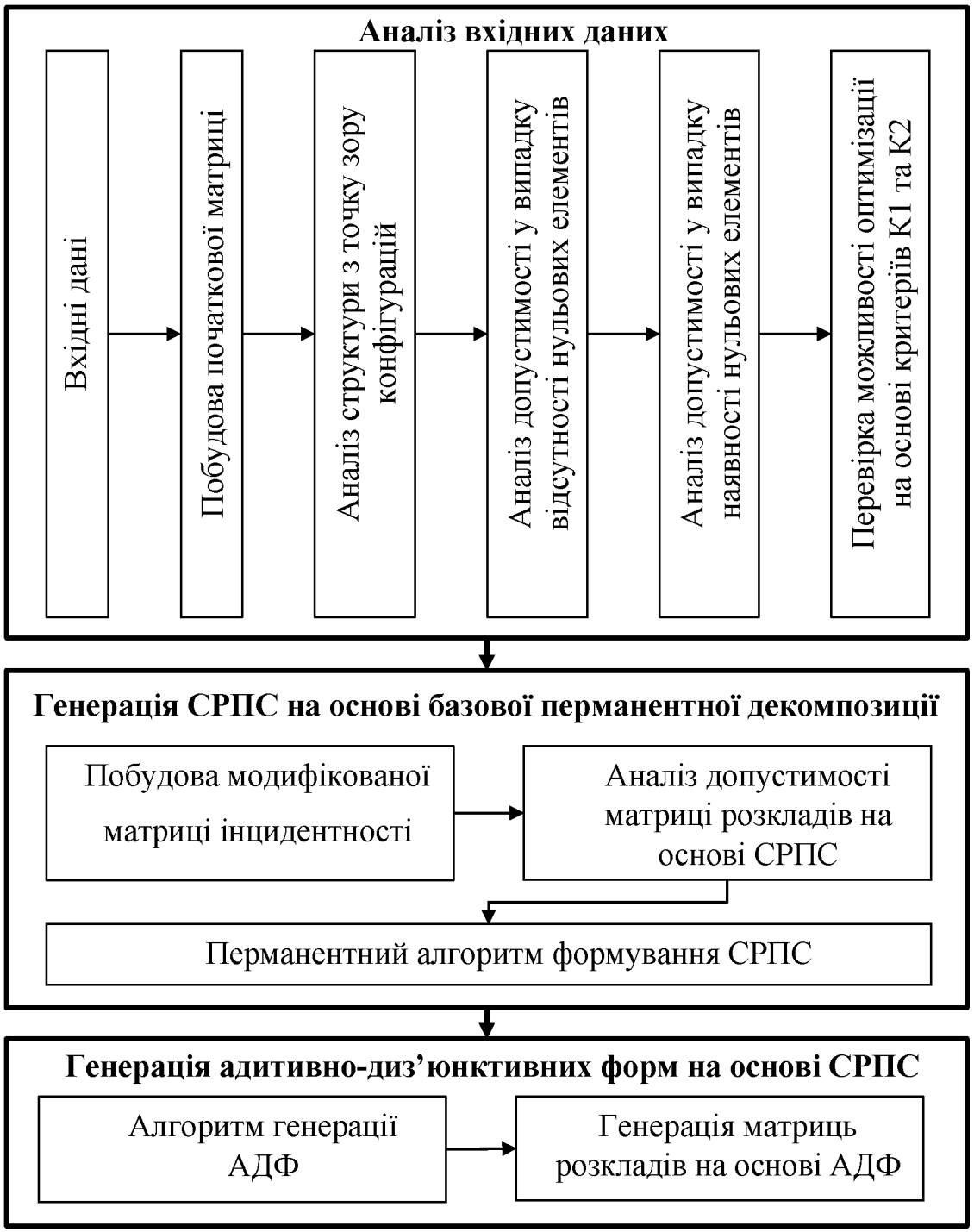

Рис.3.4. Структура інформаційної технології складання розкладів згідно перманентної декомпозиції з використанням АДФ (варіант використання базового перманентного підходу)

Ця алгебра вперше була запропонована в даній роботі і служить єдиній меті – розробці алгоритму генерації усіх можливих допустимих варіантів остаточних матриць розкладів в основі СРПС. Процес вирішення цієї проблеми включає два етапи:

1. Друга модифікація матриці інцидентності, яка дає вичерпну інформацію про кількість пар.

2. Модифікація процедури декомпозиції перманента з точки зору структур даних, в якій враховуються операції алгебри АДФ. Результатом такої процедури є не просто список СРПС, а адитивно–диз'юнктивна форма, яка дозволяє вже безпосередньо генерувати (рисунок 3.5.) варіанти остаточних матриць розкладів (операція диз'юнкції призводить до різних варіантів розкладу, а операція обов'язкового включення дозволяє формувати один конкретний варіант). Таким чином, задача формування матриці розкладу розв'язується повністю.

Зауважимо, що методи, які складають інформаційну технологію, можуть працювати і локально як самостійні модулі в поєднанні з іншими підходами, та використовуються для розв'язання задач теорії розкладів й генерації комбінаторних об'єктів. Так, наприклад, методи конфігураційного підходу можуть суттєво підвищити ефективність генетичних алгоритмів, оскільки даватимуть відразу допустимі матриці, генетичний алгоритм вже працюватиме із допустимими матрицями, а не з довільними.

Аналогічно це стосується і перманентного підходу – після етапу формування СРПС можна застосувати генетичний алгоритм.

При наявності готових допустимих варіантів матриць розкладів можна враховувати і додаткові умови вже після формування матриць шляхом відкидання варіантів, які не задовольняють вимоги. Зауважимо, що перманентні методи можуть бути ефективними і для генерації будь-яких комбінаторних об'єктів, наприклад, генерації перестановок, що є цікавим для задач створення систем захисту інформації.

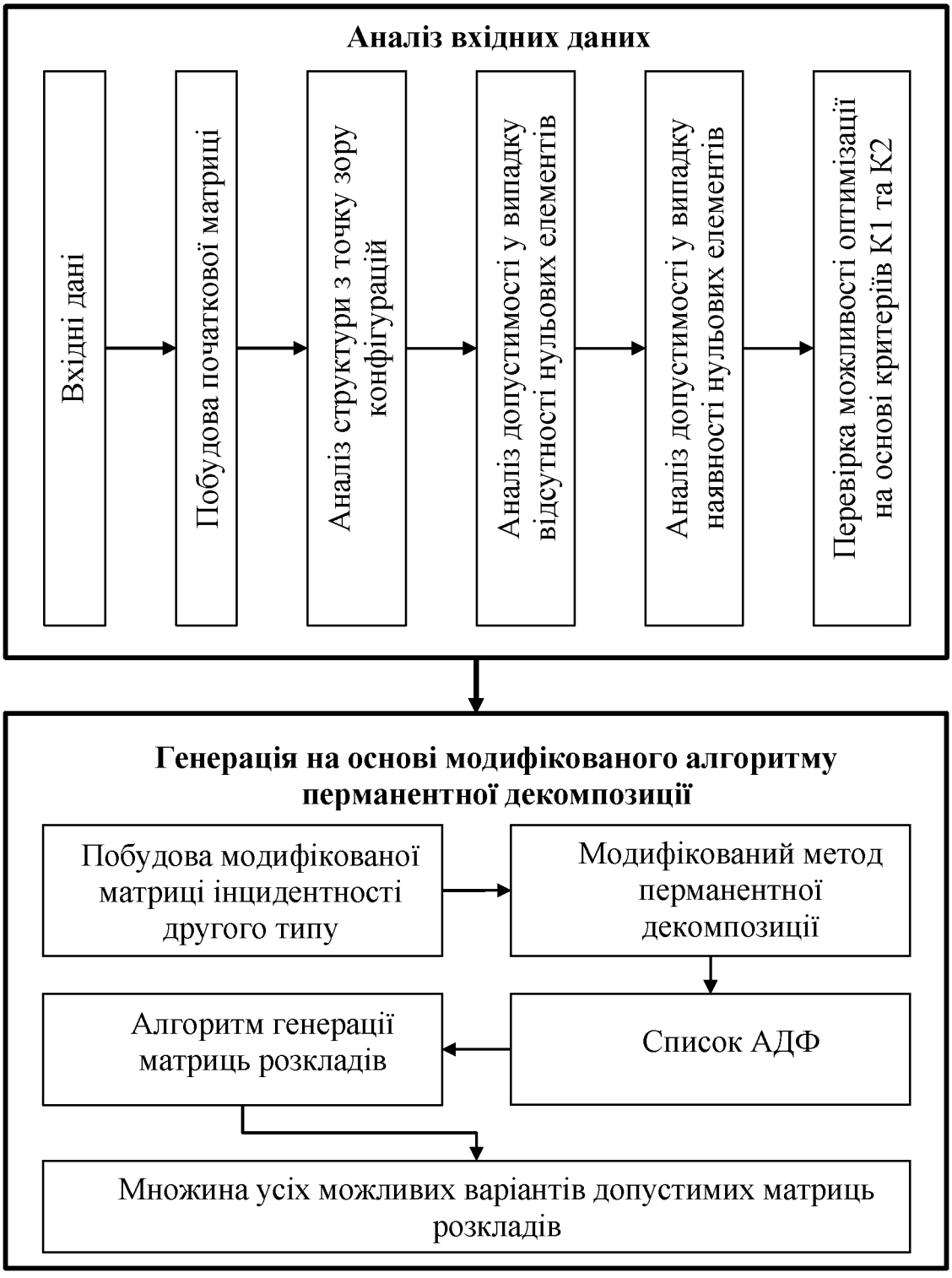

Рис.3.5. Інформаційна технологія складання розкладів згідно перманентної декомпозиції на основі алгебри АДФ (варіант прямого використання модифікованого методу перманентної декомпозиції)

Таким чином, в розділі обґрунтовано необхідність подальшого вдосконалення матриці інцидентності та методу перманентної декомпозиції з

метою побудови ефективних алгоритмів формування загальних матриць розкладів на основі СРПС. Слід відзначити ефективність застосування методики автоматизованої генерації розкладів та формування експертних оцінок, що буде в певних аспектах розглянуто у наступному розділі.

3.9. Висновки до третього розділу

Запропоновано метод перманентної декомпозиції, а також метод, що грунтується на використанні алгебри адиктивно-диз'юнктивних форм. Відповідні методи включають в себе низку алгоритмів.

Запропоновано алгоритм формування модифікованої матриці інцидентності, що дозволяє коректно врахувати інформацію про потоки. Суть модифікації полягає у введенні додаткових стовпців для кожного викладача, що має поточні пари з коректним відображенням інформації про структуру потоку.

Запропоновано визначення модифікованого перманента для матриць інцидентності, яке відрізняється тим, що в сумі всіх можливих добутків елементів матриці, кожен з яких містить по одному елементу з кожного рядка та з різних стовпців, елемент потокового стовпця не може бути в добутку разом з елементами інших рядків, що відповідають цьому ж потоку. Така інтерпретація перманента дає можливість побудувати ефективний алгоритм декомпозиції та алгоритм формування систем різних представників стовпців.

Побудовано алгоритм декомпозиції модифікованого перманента, що включає у себе конструктивну процедуру формування систем різних представників стовпців матриці розкладів та запису індексів стовпців у комірки пам'яті та який виключає, у підсумку, можливість одночасного проведення викладачем кількох пар, і в той же час допускає одночасну присутність викладача у кількох групах, якщо допускаються потоки. Відповідний алгоритм декомпозиції із «запам'ятовуванням» може бути

використаний для розв'язання широкого класу задач генерації комбінаторних об'єктів, зокрема, перестановок (що реалізовано в даному розділі як приклад), комбінацій, підмножин різних елементів деякої системи множин.

Вперше розроблені алгоритми формування матриці розкладів у межах концепції порядку, що відрізняються використанням специфічного відношення порядку, введеного на множині матриць розкладу.

Враховано особливості випадків допустимості потоків та їх відсутності. Перманентний алгоритм генерації перестановок має клас складності, аналогічний класу складності алгоритму, що базується на відношенні порядку, складність його менша принаймні на 46%. Однак, на відміну від класичних алгоритмів, перманентний алгоритм допускає узагальнення для вирішення значно складніших задач теорії розкладів, а класичні алгоритми генерації перестановок є вузькоспеціалізованими для вирішення конкретних задач.

Запропоновано інформаційну технологію для задач складання розкладів на основі використання спеціального відношення порядку на множині матриць розкладів.

Запропоновано другу модифікацію матриці інцидентності, яка відрізняється тим, що, на відміну від попередніх варіантів, містить у собі повну інформацію, необхідну для складання розкладу. Відповідна модифікація дозволила внести суттєві корективи у процедуру декомпозиції перманента цієї матриці та привела до необхідності розробки алгебри адитивно–диз'юнктивних форм.

Запропоновано спеціальну алгебру адитивно–диз'юнктивних форм (АДФ), що містить дві операції – диз'юнкції та обов'язкового включення (додавання) та відрізняється алгоритмічним характером її операцій. Операція диз'юнкції означає вибір і, відповідно, дублювання відповідних списків, що містять диз'юнктивні форми, в процесі роботи рекурсивних процедур генерації. Операція обов'язкового включення означає просте включення відповідної СРПС як наступного рядка у біжучу матрицю розкладу.

На основі АДФ вперше запропоновано два алгоритми формування матриць розкладів:

– перший алгоритм дозволяє формувати матриці розкладу безпосередньо в процесі декомпозиції перманента другої модифікації матриці інцидентності;

– другий алгоритм дозволяє утворювати АДФ на основі СМПР, що сформована як результат алгоритмів, сформульованих у попередньому розділі.

Основні результати розділу опубліковані в роботах [4–9].

РОЗДІЛ 4
ІНФОРМАЦІЙНА СИСТЕМА ТА ДЕЯКІ ОСОБЛИВОСТІ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ. ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ БАЗОВИХ АЛГОРИТМІВ

4.1. Інформаційна система складання розкладів, яка максимально враховує побажання стейкхолдерів

Сучасна парадигма якості освітніх програм передбачає створення комфортного середовища для усіх учасників освітнього процесу, усіх стейкхолдерів. Це стосується, очевидно, і процесу формування розкладу занять. В ідеалі побажання усіх стейкхолдерів мають максимально враховуватись. Це є новий і дуже важливий аспект, який має, без сумніву, певним чином враховуватись у програмних комплексах, що здійснюють автоматизацію процесу складання розкладу занять ЗВО. Очевидно, що така вимога породжує додаткові критерії і так у дуже складній багатокритеріальній задачі та ускладнює відповідні алгоритмічні реалізації. А тому в даному розділі розглянемо певні евристичні підходи, які не дають можливості збудувати оптимальний розклад в строгому його розумінні, однак дають дієві механізми у поєднанні з строго оптимізаційними підходами а також з методом експертних оцінок.

Зауважимо, що задача складання розкладів є багатокритеріальною, якщо враховувати певні додаткові умови. Якщо говорити про розклад занять чи зустрічей, то зразу ж виникає низка критеріїв, які в певному розумінні мають антагоністичний характер. Наприклад, розклад має максимально враховувати інтереси викладачів чи слухачів? Якщо домінанту мають слухачі, то постає проблема мінімізації «вікон» для них, виникають якісь специфічні вимоги, наприклад, хтось не може відвідувати заняття в першій половині дня, в якісь окремі години тощо. З точки зору викладачів часто на практиці виникають

задачі мінімізації кількості робочих днів, виникають якісь категоричні умови, аналогічні як і у слухачів, наприклад, неможливості читання пар в окремі дні.

В такому випадку необхідно розробляти формальні критерії, що містять вимірювані параметри та застосувати певні вагові коефіцієнти для переходу від багатокритеріальної до однокритеріальної задачі. Наявність великої кількості критеріїв суттєво ускладнює задачу, коли розглядається підхід, що ґрунтується на генерації усіх можливих варіантів.

Таким чином, якщо розклад розробляється за умови максимального врахування побажань стейкхолдерів, то за умови категоричного врахування їх побажань скласти його буде часто просто неможливо, оскільки система додаткових умов може породжувати протиріччя. Окрім того, окремі вимоги, що стосуються, наприклад, кількості зустрічей у день, на практиці можуть мати рекомендаційний характер: зустрічей може бути 2 або 3, або може й 4, якщо в цьому виникне необхідність. При цьому проект буде виконуватись в будь-якому випадку. В такому випадку виникають нечіткі додаткові умови, які, знову ж таки, має формувати експерт. З врахуванням зазначених обставин логічним виглядає розробка комбінованого підходу, в якому експертні оцінки поєднуються з процедурами автоматизованого складання розкладу.

Такий підхід реалізується на прикладі модуля Schedule builder 1.0 для складання розкладу, який проходив тестування, зокрема, у Рівненському державному гуманітарному університеті та у Рівненському коледжі НУБіП. Модуль містить функції для максимального врахування побажань викладачів та студентів. Відповідна програмна реалізація відрізняється власною оригінальною низькорівневою системою побітового кодування, що оптимізує представлення даних, реалізацією усіх алгоритмів з використанням побітових операцій та оптимізацією роботи з пам’яттю комп’ютера (в якості мови програмування було обрано C++).

З врахуванням побажань стейкхолдерів до розкладу занять вхідні дані можна поділити на декілька груп.

Перша група – це нормативні дані, що визначаються навчальним планом, який жорстко регламентує мінімально допустиму кількість пар для кожної освітньої компоненти на тиждень.

Друга група – це множина базових критеріїв. Водночас виникають додаткові вимоги – побажання стейкхолдерів, які теж необхідно розділити як мінімум на дві групи за ступенем їх критичності: критичні (неможливість прочитати викладачем якусь пару саме у визначений час у зв'язку з поважними обставинами – хвороба, фізична неможливість дістатись до аудиторії при навчанні офлайн тощо) та некритичні. Некритичні побажання можуть бути проігноровані, якщо алгоритм не може технічно їх реалізувати.

Отже, виникає система пріоритетів для додаткових умов – спочатку умови критичні, а далі – некритичні. Очевидно, що система пріоритетів має регулюватись експертом (диспетчером, відповідальною особою) або автоматизовано, коли стейкхолдери самі виставляють пріоритет.

Структура відповідної системи зображена на рисунку.4.1.

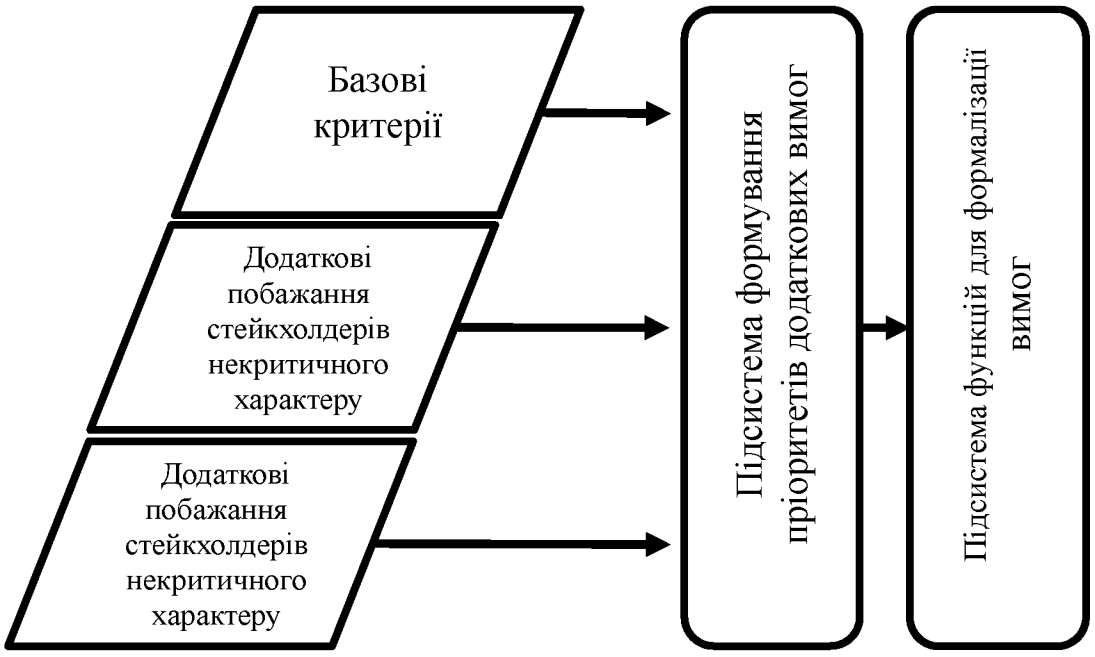

Рис.4.1. Система врахування вимог

Розроблену інформаційну систему використання конфігураційного підходу подано у вигляді діаграми руху потоків даних на рисунку 4.2.

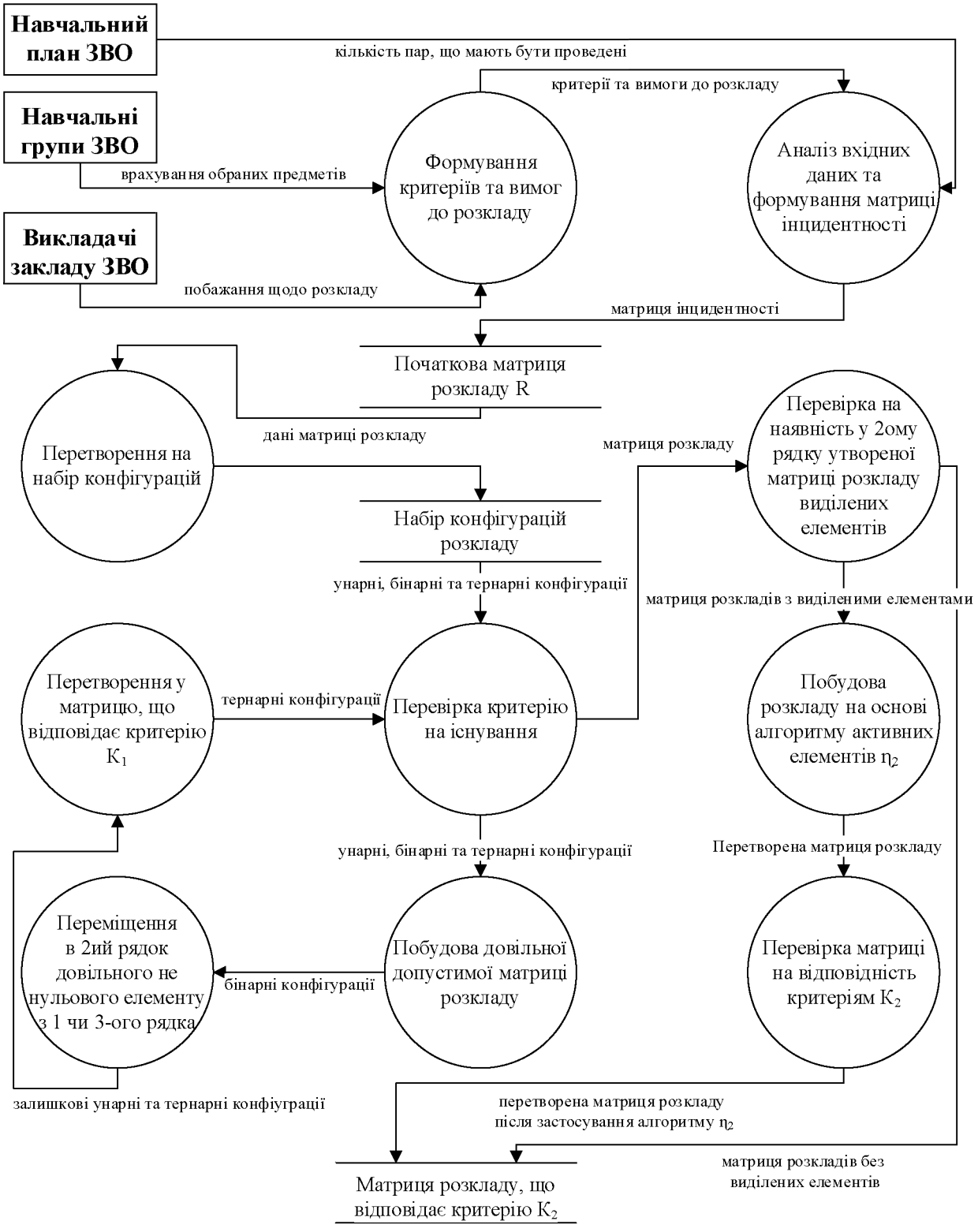

Рис. 4.2. Діаграма руху потоків даних при використанні конфігураційного підходу

Розроблену інформаційну технологію використання перманентного підходу разом з АДФ подано у вигляді діаграми руху потоків даних, що зображено на рисунку 4.3. (при зображенні руху потоків даних на рисунках 4.2. та 4.3 використано нотацію Гейне-Сарсона).

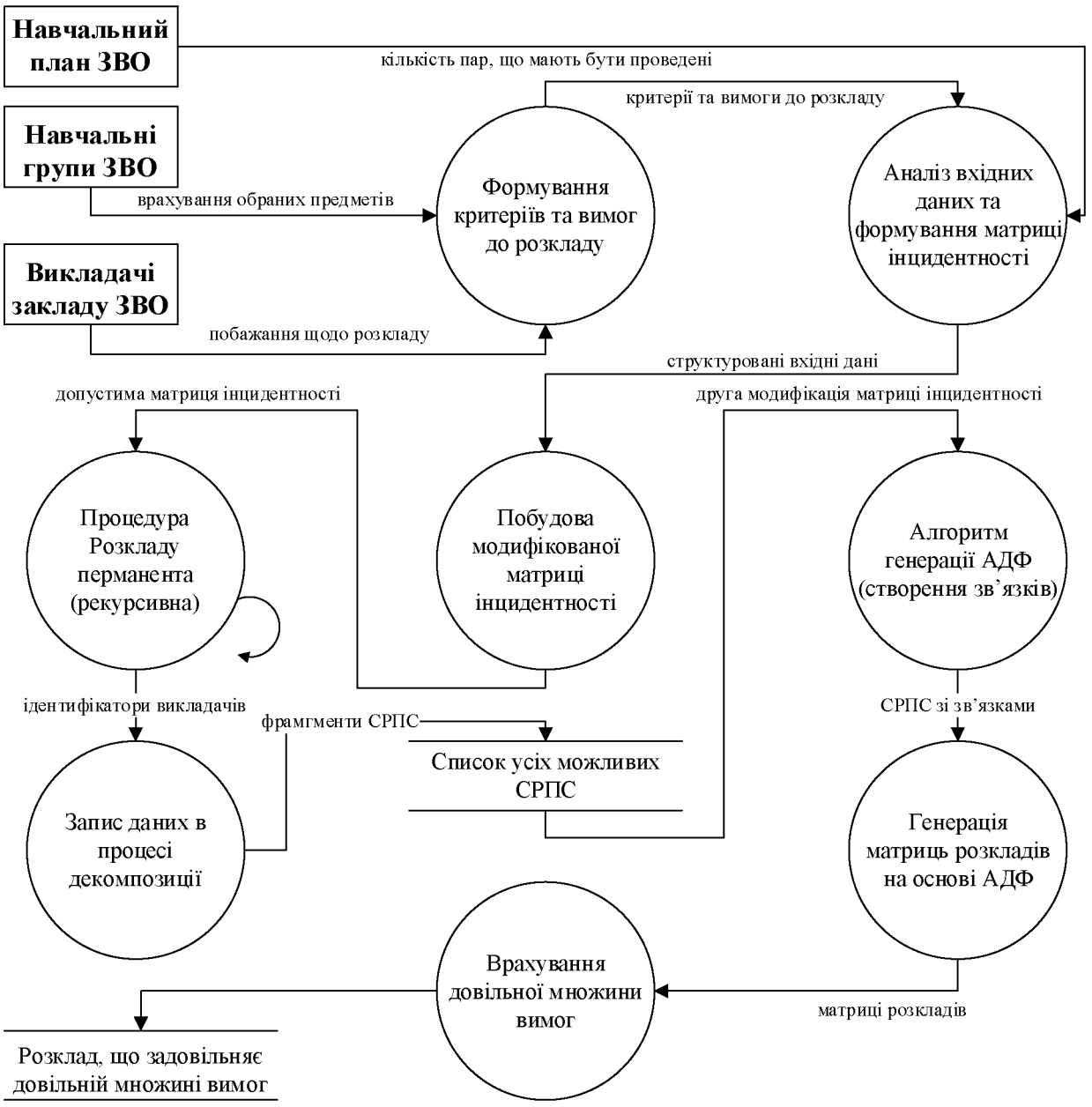

Рис. 4.3. Діаграма руху потоків даних при використанні перманентного підходу з АДФ

4.2. Опис системи кодування та структур даних

Важливим аспектом та особливістю даної інформаційної технології є структури даних та використання побітових операцій (рисунок 4.4.).

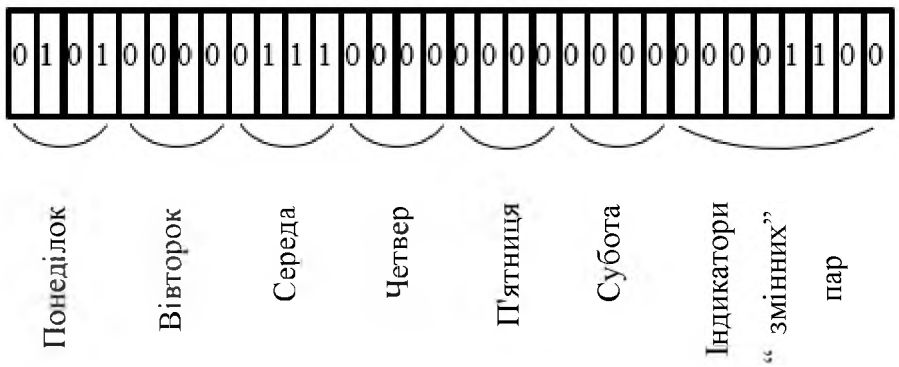

Рис.4.4. Особливості побітового представлення інформації про пари

Розклад являє собою двовимірну матрицю, елементами якого є числа типу int, що займає в пам'яті 4 байти. При цьому старших два байти містять інформацію про ідентифікатор викладача, що читає по чисельнику, а молодші два байти – по знаменнику. При цьому введено спеціальний клас для збереження інформації про викладача.

Однак тут не використовуються бази даних, а інформація про те, які пари має викладач та в яких групах, записується в масиві цілих чисел, кожен елемент якого відповідає групі. При цьому, з 32 бітів, перші 24 біти використовуються для представлення інформації про пари в даній групі (24 – це 4 пари протягом кожного з 6 днів), а інші 8 біт використовуються для представлення інформації про те, чи пара є «змінною», для кожної пари відповідно («змінні» пари – це пари за чисельником та знаменником; 1 – пара є змінною, 0 – ні).

Таким чином, усі додаткові вимоги реалізовані із застосуванням функцій для побітових операцій. Остання інформація вимагає наявності особливої підсистеми для введення вхідної інформації про кількість пар, викладачів,

структуру потоків тощо. В реалізації запропоновано низку технічних рішень для формування файлу вхідних даних. Очевидно, що ця підсистема може бути автоматизованою.

В основі інформаційної системи знаходиться підсистема виставлення пар з врахуванням пріоритетів. Загальна структура інформаційної технології зображена на рисунку.4.3.

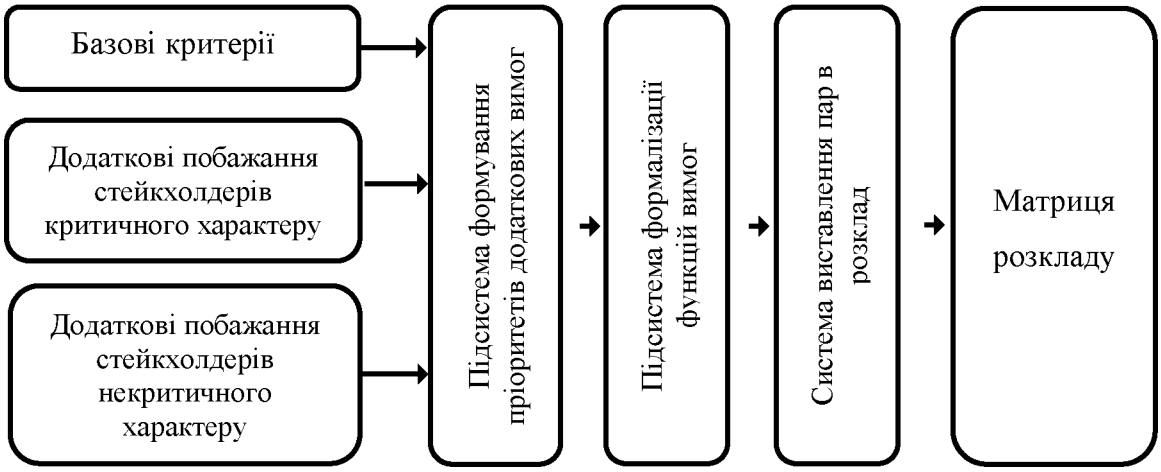

Рис.4.5. Загальна структура інформаційної технології складання розкладів згідно перманентної декомпозиції, що враховує побажання стейкхолдерів

Розглянемо відповідний підхід на прикладі складання розкладу занять у закладі вищої освіти (РДГУ, факультет математики та інформатики). Нехай на факультеті на бакалаврських програмах навчається 20 груп студентів, по 5 груп на кожен курс. Для проведення лабораторних занять кожна група ділиться на 2 або навіть 3 підгрупи. Навчальне навантаження складається таким чином, що заняття з окремих дисциплін проводяться через тиждень. Таким чином, у розкладі одна дисципліна стоїть у чисельнику (непарний тиждень) а інша – у знаменнику (парний тиждень). Очевидно, є заняття, що читаються потоками, причому потоки об’єднують перші три групи кожного курсу, 2 останні групи кожного курсу або усі групи курсу для специфічних дисциплін. Поточні пари також можуть читатись по парних чи непарних тижнях. При цьому низка дисциплін має категоричний пріоритет, бо читається

лекторами для всього університету (БЖД, іноземна мова тощо) і мусить бути розміщена в розкладі у визначений час. Також існують певні обмеження аудиторного фонду, які не дозволяють в окремі дні виставити, наприклад, лише усі лабораторні заняття.

Розглянемо детальніше специфіку структур даних, які застосовують для вирішення запропонованої задачі. Оскільки кожен викладач має ідентифікатор, то, як вже згадувалось вище, розклад є матрицею розмірності $24 \times n$, де n – кількість підгруп. Отже, для збереження розкладу будемо використовувати масив unsigned int *rozk*[24][60]; У такому разі є 20 груп, кожна з яких може поділятися на 3 підгрупи. Крім того, є 6 робочих днів та 4 пари в день.

В задачі допускається, що кожна група ділиться на три або дві підгрупи. Тому і n кратне 3. Якщо якась підгрупа не задіяна, має «вікно», то відповідний елемент масиву *rozk* рівний 0. Крім того, у розкладі допускаються пари «змінні», тобто такі уроки, які проводяться лише в парні чи непарні тижні (чисельник та знаменник). Остання обставина значно ускладнює можливість простого представлення розкладу у вигляді матриці. Щоб врахувати змінні будемо використовувати тип unsigned, а також молодший та старший байт числа типу unsigned int (якщо тип займає 4 байти, то два старші байти не використовуються). В молодший байт будемо записувати номер викладача, пара якого читається по «знаменнику», а в старший – по «чисельнику».

В якості вхідних даних нам необхідно мати інформацію про те, скільки та яких пар кожен викладач проводить у кожній групі.

Для представлення цієї інформації введемо такий клас *vykl*, в якому є поле, яке задає ім’я викладача та масив значень *group*[60] типу unsigned long. Кожне число–елемент масиву відповідає своїй групі. Причому в цьому числі кодується інформація про те, які за порядком пари та в які дні тижня читає певний викладач, а також інформація про те, чи є ці пари «змінними». Кодування проводять таким чином. Як відомо, значення типу unsigned long

займає 4 байти чи 32 біти. Перші 24 біти в представленні числа використовуються для подання інформації, які пари читаються викладачем (всі пари пронумеровані від 1 до 24, на день може бути 4 пари). Наприклад, послідовність 0001 1100 0000 1100 0000 0011 означає, що певний викладач читає 2 перші пари підряд в понеділок, 2 останні пари у середу та п'ятницю та має першу пару в суботу (послідовність тут простежуються справа наліво). Останні 8 біт представлення числа використовуються для подачі інформації про те, чи є відповідні пари «змінними». Якщо у цій послідовності стоїть 1, то відповідна пара є «змінною», 0 – звичайною. Наприклад, для описаної вище послідовності можемо розглянути інформацію про «змінні»: 1100 0001 0001 1100 0000 1100 0000 0011.

Одиничка у 25 біті означає, що перша пара в понеділок є «змінною» парою. Наступні чотири пари на тижні є звичайними, тоді дві останніх – також є «змінними» (тобто четверта пара в п'ятницю та перша пара в суботу). Іншими словами, біти від 25–го по 32–й переглядаються справа наліво. Кожен біт відповідає одиничці, яка стоїть в області числа від першого по 24–й біти. Отже, зрозуміло, що викладач не може мати 8 пар на тиждень в одній групі. Таке обмеження було отримано безпосередньо з представлення даних.

4.3. Специфіка представлення вхідних даних та функції врахування побажань стейкхолдерів

В якості вхідних даних будемо використовувати модифіковану матрицю інцидентності, що надаватиме вичерпну інформацію про те, скільки кожен викладач має пар у кожній групі щотижня, інформацію про потоки та пари–змінні. Інформацію задаватимемо наступним чином. Функція ініціалізації void *init*(int *nomvykl*,char *namevykl*[7],char *namegr*[4],int *nurok*,int *myg*) зчитує інформацію про ID викладача, ім'я викладача, назву групи, номер пари та прапорець, чи є пара «змінною». При цьому ім'я викладача є не обов'язковим

і використовується швидше для зручності роботи з файлом при тестуванні програми.

Приклад фрагменту такого файлу вхідних даних:

1 golin MF21 6 0
1 golin PM11 1 0
1 golin I11 1 0
1 golin PM11 2 0
1 golin I11 2 0
1 golin PM11 3 0
1 golin I11 4 0
1 golin PM11 5 0
1 golin I11 6 0
2 sapil I11 1 1
2 sapil PM11 1 1
2 sapil I11 2 0
2 sapil PM11 2 0
2 sapil PM11 3 0
2 sapil PM11 4 1
53 tymosh I11 1 0
53 tymosh I11 2 1
4 tur PM21 1 0
4 tur I21 1 0
4 tur PM21 2 0
4 tur I21 2 0
4 tur PM21 3 0
4 tur I21 4 0
4 tur PM21 5 0
4 tur I21 5 0
4 tur PM41 1 0
4 tur PM41 2 0
10 kyryk PM11 11 0
68 helen PM12 11 0
27 nikol I11 12 0
26 dydiak I12 12 0
10 kyryk PM22 2 0
10 kyryk I22 3 0
26 dydiak PM23 2 0
5 mor MI11 1 0
5 mor MI12 1 0

Специфічним є тут представлення інформації про номери пар та потоки. Якщо початковий номер для кількох груп є однаковий, то вважається, що задано потік. Тоді номери інших пар цього ж викладача повинні бути будь-якими іншими, але не більші за 24–максимальну кількість пар на тиждень. Наприклад, для умовного викладача Tur з ID 4 фрагмент вхідного файлу:

4 tur PM21 1 0
4 tur I21 1 0
4 tur PM21 2 0
4 tur I21 2 0

задає дві поточних звичайних пари для груп PM21 та I21, початкові номера яких 1 та 2 відповідно.

Потім задається інформація про дві практичні для цих же груп з номерами 3 та 4:

4 tur PM21 3 0
4 tur I21 4 0

Далі знову потоки:

4 tur PM21 5 0
4 tur I21 5 0

Вже 5 пара для цих груп. Далі задаються практичні для групи PM41, при цьому початкова нумерація пар може йти спочатку, бо група вже інша:

4 tur PM41 1 0
4 tur PM41 2 0

Якщо є поділ на дві підгрупи, то задається блок для двох підгруп:

43 soroka MI11 6 0
43 soroka MI12 7 0

Або при поділі на три підгрупи:

5 mor MI11 1 0
5 mor MI12 1 0
5 mor MI13 1 0

Таким чином, інформація про поточні пари задається специфічно – ідентифікатором потоку є одиничка для відповідних груп у полі group для відповідного екземпляра класу *vykl*. При цьому в силу можливості поділу на 3 підгрупи кожної групи, а також специфічної нумерації груп, наприклад, МІ11, МІ12, МІ13 ідентифікатором потоку виступатиме 1 в бітових представленнях елементів масиву *group*[60] для груп, номер яких кратний 3.

По такому принципу працюють відповідні функції перевірки на потоки. Розглянемо принцип генерації на прикладі однієї з функцій вставки function *perevriad3()*.

Вхідні дані: номер рядка матриці розкладу y1, номер позиції j, ідентифікатор викладача i, yt-номер біта відповідної групи в класі викладача.

В функції перевірки по рядку для знаходження місця вставки спочатку перевіряється, чи до позиції (y1,j) елемента i ще немає.

Перевірка сусідніх елементів, врахування заборон та пар–«мигалок» в циклі *for* nv ← 1 *to* 80 *do* виглядає наступним чином. Спочатку перевіряються різноманітні заборони (можливість вставки, чи в даній позиції вже не стоїть ідентифікатор викладача тощо). Далі здійснюється перегляд масиву груп.

З огляду на описану вище концепцію структур даних, реалізовані всі функції програми з врахуванням побітових операцій (див. додатки).

Загальна концепція побудови розкладу є наступною. Спочатку виставляються пари, які мають абсолютний пріоритет. Він визначається за допомогою системи функцій побажань. В таких функціях окремо розглядаються випадки наявності потоків та їх відсутності, наявності мигалок. Перевірки здійснюються із використанням функцій аналізу побітового представлення інформації про необхідність проведення пари у екземплярах класів, що відповідають викладачам.

Також визначаються функції заборон, що мають обов'язковий характер, зокрема, заборона окремої пари для окремого викладача, заборона дня, заборона групи, заборона четвертих пар тощо.

Реалізовано, зокрема, функції:

– void *zaborchetv*(int nv) – заборона четвертих пар;

– void *zaborday(char namevykl[10],int nomday)* – заборона певного дня за побажанням викладача по його Id;

– void *zaborpar*(char namevykl[10],int nomday,int para) – заборона певної пари за побажанням викладача;

– void *zaborgrup*(char namegr[4],int nomday,int para) – заборона певної пари за побажанням студентів групи.

Наступний аспект – це логіка роботи функцій вставки пар у розклад. Реалізовано функції двох видів: вставка звичайна void *vstavkazv1*(unsigned int i, int j, unsigned long yt), яка на вільне місце з врахуванням заборон виставляє пари фактично випадково і функція int *vstavkazag1*(unsigned int i, int j, unsigned long yt), що реалізовує вставку з врахуванням критерію мінімізації кількості робочих днів викладача. Аналогічні функції розроблені для вставки пар–мигалок.

Функції другого виду типу *vstavkazv2(unsigned int i,int j,int yt)* працюють за таким принципом: якщо десь в розкладі вже стоїть пара певного викладача і необхідно вставити його іншу пару, то вона вставлятиметься так, щоб у викладача не було вікна (доклеювання). Окрім того, реалізовані окремі функції вставки, що перевіряють можливість вставки двох чи трьох пар по підгрупах. Всі функції перевіряють відсутність можливих співпадінь по рядку у розкладі, а також враховують умову, що двох пар одного і того ж викладача у тій самій групі в один день не повинно бути.

Основна процедура формування розкладу за рахунок використання ієрархії класів та створення низки функцій для відповідних побітових операцій має просту структуру. В залежності від умов викликаються ті чи інші функції вставки елемента у матрицю розкладу. В прикладі розглядається 60 підгруп, 20 груп та 80 викладачів (звичайно, ці параметри можуть змінюватись).

Також реалізовано низку функцій, що здійснюють оптимізацію розкладу, наприклад, в останній робочий день тижня, якщо виявиться, що навантаження недостатнє та ціла система допоміжних для реалізації побітових операцій.

Результатом роботи функції генерації є числова матриця розкладу, що враховує пари–змінні. Приклад для реального розкладу занять у Рівненському державному гуманітарному університеті (РДГУ).

Цікаво, що незважаючи на дуже прості евристичні підходи та відсутність формального врахування низки критеріїв, програма спрацювала дуже гарно. Причиною є те, що евристична логіка формування розкладу в даній програмі фактично є системою, що повторює дії людей, експертів, які реально складають розклад занять. Перед початком програмної реалізації було опитано низку працівників, що вручну складали розклади в РДГУ. Інформація була зібрана як сукупність рекомендацій типу «Спочатку виставляємо поточні пари з Безпеки життєдіяльності, де задіяні усі групи з курсів та для яких аудиторії виділяються навчальною частиною», «Старшим викладачам, інвалідам, яким важко підійматись на 4 поверх, аудиторії виставляємо на перших поверхах» і т.д.

В реалізації враховано низку специфічних вимог саме цього ЗВО, наприклад, вимога, щоб іноземна мова виставлялась фактично потоком (кожна група у іншій аудиторії) із застосуванням низки викладачів іноземної мови, які вели пари на різних факультетах.

Відповідні рекомендації реалізувались функціями побажань та заборон. Далі був реалізований подібний алгоритм максимального заповнення матриці розкладів.

4.4. Деякі аспекти програмної реалізації перманентного алгоритму генерації СРПС

Розглянемо деякі особливості програмної реалізації методу перманентної декомпозиції. Розв'язок поставленої задачі можна розбити на два етапи:

1. Побудова множини всіх можливих СРПС.

2. Вибір підмножини, що відповідає відповідному набору умов (критеріїв).

Для вирішення першого етапу розв'язку задачі можемо запропонувати наступний підхід. Варіанти СРПС будемо зберігати у списку, утвореному екземплярами класу SRP:

```
class SRP
{
        public:
        static int nmax;
        int nlast;
        char *s;
        SRP * next;
        SRP()
        {
                s=new char;
        }
}
spysok;
```

Для збереження розширеної матриці інцидентності (що містить номери рядків та елементи, які відповідають стовпцям) будемо використовувати клас incydent, який і буде поступати на вхід рекурсивної функції генерації СРПС разом з елементом, який зумовив утворення відповідної матриці інцидентності в розкладі (адресою відповідного елемента списку, утвореного екземплярами класу SRP):

```
class incydent
{
        public:
        int nmaxrow;
        int nmaxcol;
        char **r;
        incydent(int nrow, int ncol)
        {
                nmaxrow=nrow;
                nmaxcol=ncol;
                r=new char[nmaxcol*nmaxrow];
        }
        ~incydent()
        {
                delete r;
        }
};
```

Рекурсивна функція generic, здійснює розклад за рядком із запам'ятовуванням. Тоді процедура генерації виглядає наступним чином:

Алгоритм generic

△ current – SRP instance

current ← a

for j ← 1 **to** I.nmaxcol **do**

△ IN – incident instance

generic(current, IN)

end for

У результаті роботи цієї програми буде сформований список екземплярів класів SRP, текстові поля яких міститимуть всі варіанти СРПС.

Для генерації матриці розкладу на основі СРПС можемо запропонувати різноманітні підходи.

Одним із можливих варіантів формування матриці розкладу може бути послідовний аналіз із виключеннями усіх можливих варіантів СРПС, що матиме значну обчислювальну складність. Адже, якщо кількість СРПС у множині є n, кількість рядків є k, то кількість необхідних варіантів перевірок є C_n^k. А тому більш оптимальний та елегантний метод розв'язання цієї задачі розглянемо нижче.

Розглянемо основні особливості програмної реалізації процесу генерації матриці розкладів на основі АДФ. Опишемо структури даних та реалізацію описаних вище алгоритмів засобами C++. Для збереження інформації про окремі АДФ введемо відповідний клас:

```
class ADF
{
        public:
        SRP* context;
        char * after;
        char * before;
        ADF *next;
        int delflag;
        ADF()
        {
                after=new char[2];
                before=new char[2];
                delflag=0;
        }
};
```

У цьому класі окрім поля для зберігання СРП є ще поля для збереження типу «з'єднувальних конструкцій» – двохелементні масиви after, before. Перший елемент містить знак операції, другий – можливу дужку.

Функція ADF* *startgener* (SRP *head) генерує по списку СРПС початкову АДФ, аналізуючи перші елементи рядків СРПС. Тіло функції описано у Додатку А.

Функція int *ident*(int m,int j, SRP* si, incyd* pi) здійснює перевірку можливості включення наступного елемента в СРП.

Функція void *generlexc*(int n,int k, int *mas) є допоміжною і генерує наступну найближчу в сенсі лексикографічного порядку комбінацію елементів масиву, починаючи з позиції k для бар'єру n.

Функція int *goodblock*(ADF* head, int *mas, int k, int m,int j,incyd* pi) перевіряє, чи не потрапляє 2 СРПС в один диз'юнктивний підблок, а також чи не більший індекс за кількість коректних входжень.

Функція ADF* *copyblock*(ADF* startblock) копіює блок обов'язкового включення одразу ж за startblock.

Функція void *genernewblock* (int m,int j, int* mas,int size,ADF* startblock) генерує новий блок з включеннями елементів у відповідності до вибраної комбінації mas номерів елементів СРПС, що включатимуться. Працює коректно після goodblock().

Функція ADF* *nextblockadr*(ADF* head) повертає адресу наступного блоку для аналізу чи включення.

Функція int *mjgeneric*(ADF* head ,incydent* pi,int m,int j) генерує всі блоки обов'язкового включення по m–j елементу матриці інцидентності та записує після head.

Функція ADF* *ostgeneric*(SRP* head, incydent* pi) представляє основний інтерфейс для генерації АДФ та здійснює послідовний запуск функцій startgener(head), mjgeneric(rozklad,pi,m,j); а також видалення помічених СРПС.

4.5.Деякі результати комп'ютерних експериментів

4.5.1.Програмна реалізація евристичного підходу

Розглянемо програмну реалізацію евристичного підходу, який описаний вище та максимально враховує інтереси стейкхолдерів. Додаток розроблений

з використанням C++ в середовищі Rad Studio (ілюстрація запуску програми наведено на рисунку.4.6).

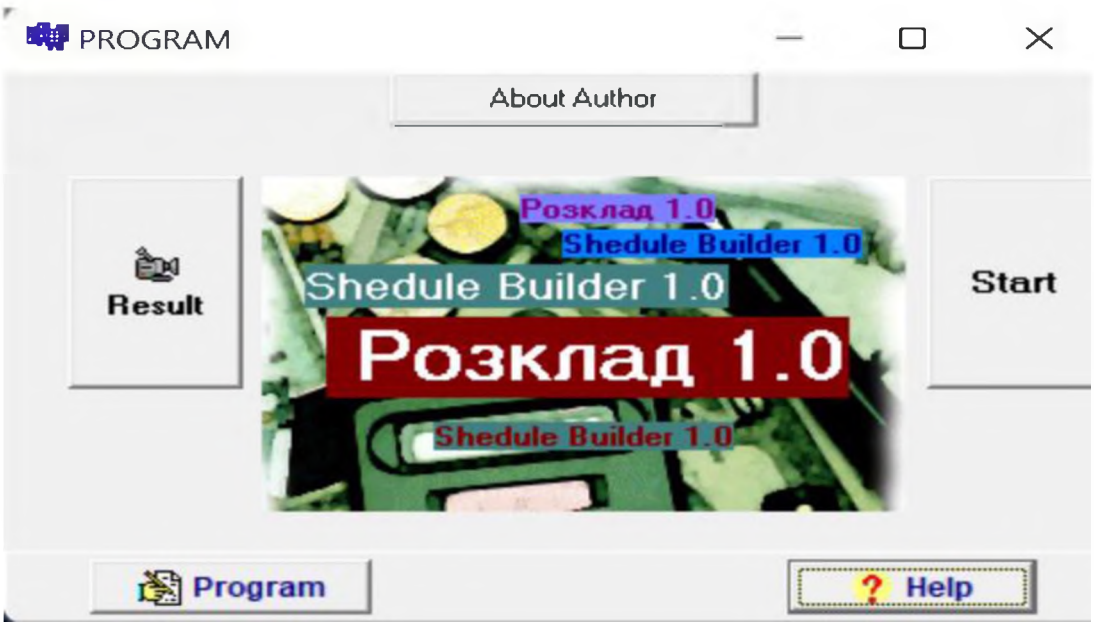

Рис.4.6. Ілюстрація програми для генерації матриці розкладу

Сама програма має мінімалістичний дизайн та набір вхідних параметрів, оскільки автор ставив собі за мету максимально спростити її використання на практиці та, у відповідності з методикою, що пропонується, головна увага приділяється саме формуванню файлу вхідних даних, який створюється в будь-якому текстовому редакторі та розміщується у папці, де міститься виконуваний файл.

Програма формує вихідний файл – матрицю розкладів, який записується в також папку, де знаходиться виконуваний файл програми.

Після цього диспетчер формує, власне, розклад занять у форматі чи програмному середовищі, які прийняті в конкретному ЗВО.

Розглянемо приклад файлу вхідних даних, який був використаний для тестування програми у Рівненському державному гуманітарному університеті на факультеті математики та інформатики (рисунок.4.7.).

```
1 babych MF21 6 0
1 babych PM11 1 0
1 babych I11 1 0
1 babych PM11 2 0
1 babych I11 2 0
1 babych PM11 3 0
1 babych I11 4 0
1 babych PM11 5 0
1 babych I11 6 0
2 turbal I11 1 1
2 turbal PM11 1 1
2 turbal I11 2 0
2 turbal PM11 2 0
2 turbal PM11 3 0
2 turbal PM11 4 1
53 tikal I11 1 0
53 tikal I11 2 1
4 bomba PM21 1 0
4 bomba I21 1 0
4 bomba PM21 2 0
4 bomba I21 2 0
4 bomba PM21 3 0
4 bomba I21 4 0
4 bomba PM21 5 0
4 bomba I21 5 0
4 bomba PM41 1 0
4 bomba PM41 2 0
10 jed PM11 11 0
68 work PM12 11 0
27 thing I11 12 0
26 doctor I12 12 0
10 jed PM22 2 0
10 jed I22 3 0
26 doctor PM23 2 0
8 kasht I21 1 0
8 kasht PM21 2 0
8 kasht I21 2 0
27 thing PM21 3 0
27 thing PM21 4 0
27 thing I21 5 0
27 thing I21 6 0
9 vitua I22 5 0
9 vitua I22 6 0
26 doctor PM22 3 0
26 doctor PM22 4 0
9 vitua PM23 3 0
9 vitua PM23 4 0
9 vitua MF11 5 0
9 vitua ME11 5 0
9 vitua MI41 7 0
9 vitua MF41 7 0
9 vitua ME41 7 0
9 vitua PM51 10 0
9 vitua PM51 12 0
9 vitua PM51 11 0
29 zarai PM52 11 0
9 vitua PM52 13 0
29 zarai PM51 13 0
10 jed MI21 7 0
10 jed ME21 7 0
10 jed MF21 7 0
10 jed MI21 8 0
10 jed ME21 9 0
10 jed MF21 10 0
5 dydk MI22 8 0
5 dydk ME22 9 0
5 dydk MF22 10 0
10 jed MI31 9 0
6 gus MI42 8 0
12 shych MI41 9 0
6 gus MI42 9 0
12 shych MI51 7 0
12 shych MI51 9 0
12 shych MI51 10 0
8 kasht MI52 10 0
12 shych MI51 11 0
8 kasht MI52 11 0
13 djun ME21 4 0
13 djun ME21 5 1
13 djun PM51 1 0
13 djun PM51 2 0
13 djun PM52 15 0
9 vitua PM51 15 0
13 djun PM51 16 0
9 vitua PM52 16 0
9 vitua MI11 1 0
9 vitua MI41 7 0
15 madsi MI51 4 0
15 madsi PM41 5 0
15 madsi PM41 6 1
16 didyk ME41 3 0
16 didyk ME41 4 0
16 didyk ME42 5 0
17 sasao MI31 2 0
17 sasao ME31 2 0
17 sasao MI31 3 0
17 sasao ME31 3 0
17 sasao MI31 4 0
17 sasao ME31 5 0
9 vitua MI41 2 0
9 vitua ME41 2 0
9 vitua MF41 2 0
```

Рис.4.7. Приклад вхідного файлу

Для коректної роботи програми необхідно у файлі вказати ідентифікатор викладача, прізвище чи довільний символьний рядок для ідентифікації особи (цей параметр не використовується у програмі і введений для диспетчера, щоб перевірити коректність вхідних даних; в прикладі реальна інформація змінена з етичних міркувань), ідентифікатор групи (в прикладі наведені реальні групи даного ЗВО), інформацію про кількість пар викладача протягом тижня у форматі, описаному вище, останній ідентифікатор, чи пара є «змінною» (0 – звичайна, 1 – «змінна»).

Результат роботи наведено на рисунку. 4.8.

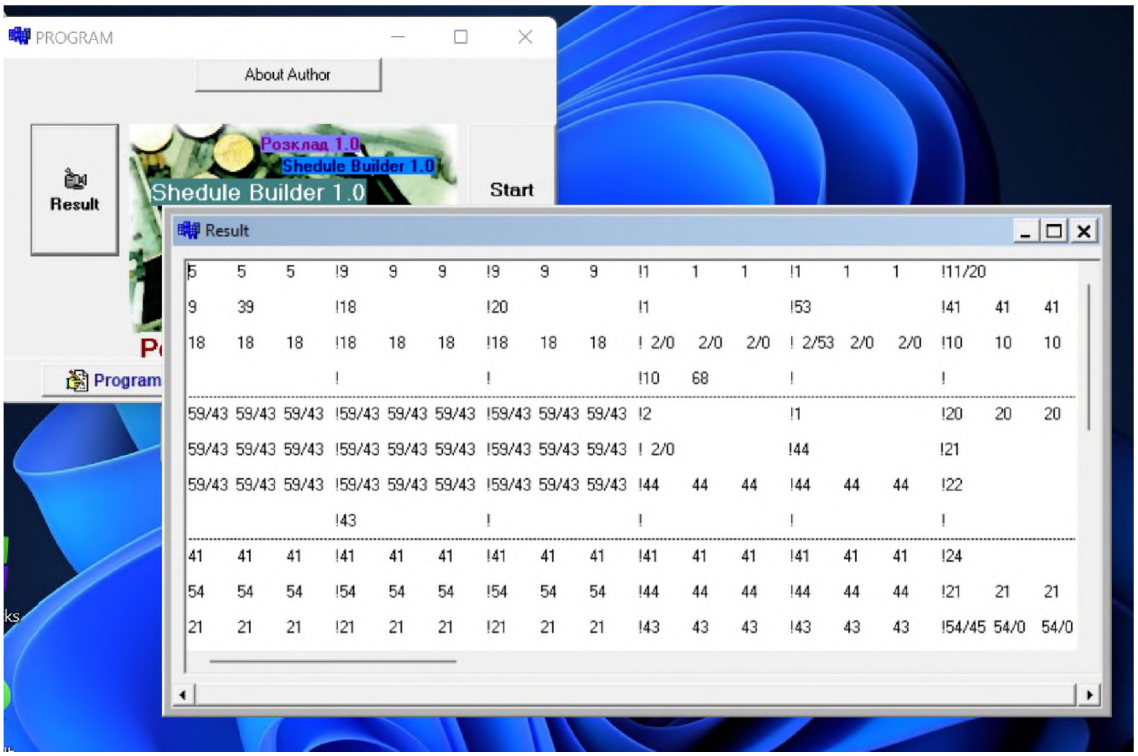

Рис.4.8. Приклад сформованої матриці розкладу

Відповідний вихідний текстовий файл сформувався у папці, де знаходиться виконуваний файл програми та може бути переглянутий в будь-якому редакторі. В якості розділювача між стовпцями розкладу по групах використано знак «!». Як бачимо, матриця розкладів містить лише номери викладачів. Для остаточного формування розкладу необхідно створити конвертор, який замість номера викладача вказує конкретну його пару з назвою предмета. Однак, це легко зробити, якщо позиції для даного викладача коректно заповнені в матриці розкладів. Окрім того, як вже вказувалось, кожен ЗВО має власні формати представлення електронної чи паперової версії розкладу занять. Незважаючи на те, що при постановці задачі було враховано можливість поділу кожної групи на три підгрупи для лабораторних занять та на дві підгрупи для практичних, допускаються пари «змінні» як для потоків, так і для звичайних пар, використовується ціла низка побажань як окремих викладачів, так і студентів, дана програма автоматично згенерувала пристойний варіант розкладу, у якому практично відсутні вікна та враховані

усі побажання. Окрім того, за рахунок використання оптимізації структур даних, відсутності роботи з базами даних та використання побітових операцій C++ дана програма працює дуже швидко. Наприклад, для ПК на базі процесора Intel Core i5–1135G7 з 16GB RAM час формування розкладу становить < 1 с.

Для порівняння, аналогічна задача на основі генетичних алгоритмів може бути розв'язана за декілька хвилин за однакових обчислювальних потужностей та за умови вирішення проблеми зупинки генетичного алгоритму [169].

Звичайно, що такий результат залежить від коректності та вичерпності інформації у вхідному файлі. Ситуація, коли якісь пари не помістились у розклад, в даному випадку вирішується можливістю виставлення четвертих пар (в матриці, як бачимо окремі пари є четвертими; рисунок.4.9.).

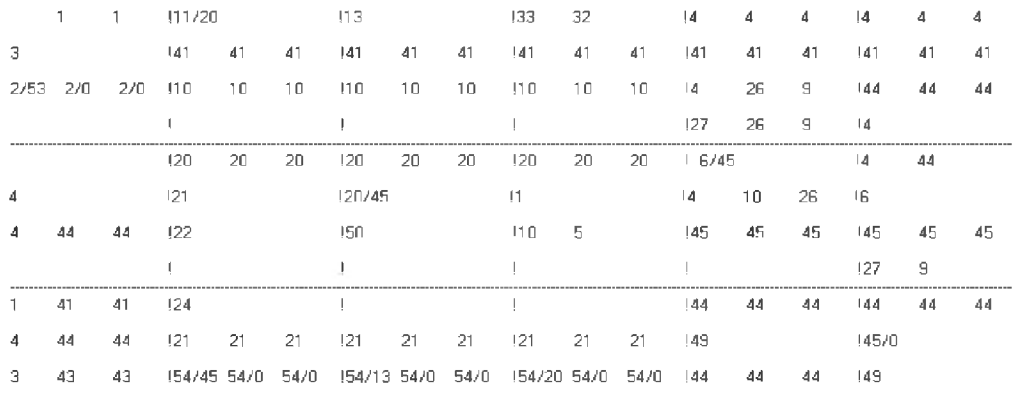

Рис.4.9. Фрагмент матриці розкладу

Для аналізу отриманої матриці розкладу скористаємось критерієм (2.2). Будемо враховувати вимоги мінімізації «вікон» викладачів, мінімізації «вікон» студентів, а також задоволення побажань окремих викладачів та особливостей аудиторного фонду. Для реалізації освітнього процесу було задіяно 64 викладачі, 20 студентських груп, кожна з яких може ділитись на три або дві підгрупи. Аналіз матриці розкладу показує, що щоден з викладачів не має «вікна» у розкладі, оскільки критерій мінімізації вікон викладачів мав більший пріоритет. Студенти ж мають «вікна», однак їх лише 7 з усього розкладу. Якщо

враховувати виконання відповідної вимоги для окремої групи, то матимемо 120 днів з врахуванням трьох пар на день, отже ступінь виконання вимоги $\left|\frac{\overline{Z_j}}{Z_j}\right| = \frac{113}{120} = 0.94$. Введемо вагові коефіцієнти окремих вимог, виходячи з експертних оцінок: 0.2, 0.1, 0.35, 0.35 (дві останні вимоги – це виставлення потокових пар для іноземної мови та фізичної культури у визначений день). Отже. отримуємо: $F = \sum_{j=1}^{J} K_j \cdot S_j. = 0.2 + 0.094 + 0.35 + 0.35 = 0.994$. Бачимо, що показник якості розкладу є досить високим. Такий результат отриманий з врахуванням особливостей навчального плану, а також достатнього аудиторного фонду для реалізації навчального процесу.

4.5.2. Дослідження ефективності методу перманентної декомпозиції у порівнянні з класичними підходами

Розглянемо як приклад тестову задачу, яка полягає у тому, щоб згенерувати можливі перестановки з n елементів.

Будемо розглядати метод перманентної декомпозиції, який записує перестановки у відповідні поля однозв'язного списку. Для методу перманентної декомпозиції всі перестановки будуть записуватись у пам'ять, оскільки результат генерації перестановок нам буде потрібен для задачі складання розкладу. Це важлива обставина, яка повинна враховуватись при порівнянні з відомими методами генерації перестановок. В даному випадку генерація перестановок еквівалентна формуванню СРПС. Усі СРПС нам необхідні для подальшого застосування алгебри адитивно–диз'юнктивних форм.

Для вибору відомих методів для порівняння з методом перманентної декомпозиції використаємо результати робіт [155-156], де теоретично та експериментально порівнювались різні алгоритми генерації перестановок, зокрема, висхідний алгоритм, лексикографічний алгоритм, Heap-алгоритм та

алгоритм Штайнмайєра–Джонсона–Троттера. Аналіз кількості арифметичних операцій показує, що всі ці алгоритми належать до одного і того ж класу складності ($O(n!)$).

Результати тестування наведені в Таблицях 4.1-4.2 для використання методів «brute–forse» та «divide and conquer» [156]. Однак, експериментальні дослідження показують, що на практиці суттєві переваги має алгоритм Штайнмайєра–Джонсона–Троттера.

Експерименти проводились на комп'ютері з процесором Intel Core 2 dual core, 2.66 GHz, 256KB кеш, 2GB RAM.

Таблиця 4.1

Результати тестування методів у варіанті «brute–forse»

Test Case	**Input Length**	**Bottom–Up**	**Метод Джонсона – Троттера**	**Лексикографічний алгоритм**
1	4	< 1 ms	< 1 ms	< 1 ms
2	6	< 1 ms	< 1 ms	< 1 ms
3	8	187.5 ms	46 ms	328 ms
4	9	1.45 s	437 ms	2.1 s
5	10	14.9 s	3.52 s	20.4 s

Таблиця 4.2

Результати тестування методів у варіанті «divide and conquer».

Test Case	**Input Length**	**Bottom–Up**	**Метод Джонсона – Троттера**	**Лексикографічний алгоритм**
1	4	< 1 ms	< 1 ms	< 1 ms
2	6	< 1 ms	< 1 ms	< 1 ms
3	8	171.2 ms	47.11 ms	301 ms
4	9	1.79 s	469 ms	2.02 s
5	10	16.7 s	3.2 s	22.7 s

Як бачимо з наведених результатів, на практиці найкращі результати у методу Джонсона–Троттера (Штайнмайєра-Джонслна-Троттера). Отже, виберемо цей метод для порівняння з методом перманентної декомпозиції. Також зауважимо, що згідно із публікаціями Стівена Скієни, на період 1990 р. швидким вважався Heap-алгоритм. Для тестування будемо використовувати OnlineGDB C++ компілятор.

Спочатку розглянемо метод перманентної декомпозиції. При величині вхідного масиву 4–6 метод працює < 1 ms (рисунок 4.10.). Для масиву 7-9 результати наведені на рисунках.4.11.-4.19. Загальне порівняння наведено у Таблиці 4.3.

Таблиця 4.3

Результати тестування оглянутих методів

Test Case	Input Length	Метод перманентної декомпозиції	Метод Джонсона – Троттера	Heap – алгоритм
1	7	433 ms	776 ms	1236 ms
2	8	4.591 s	5.683 s	13.713 s
3	9	54.440 s	52.015 s	101.562 s

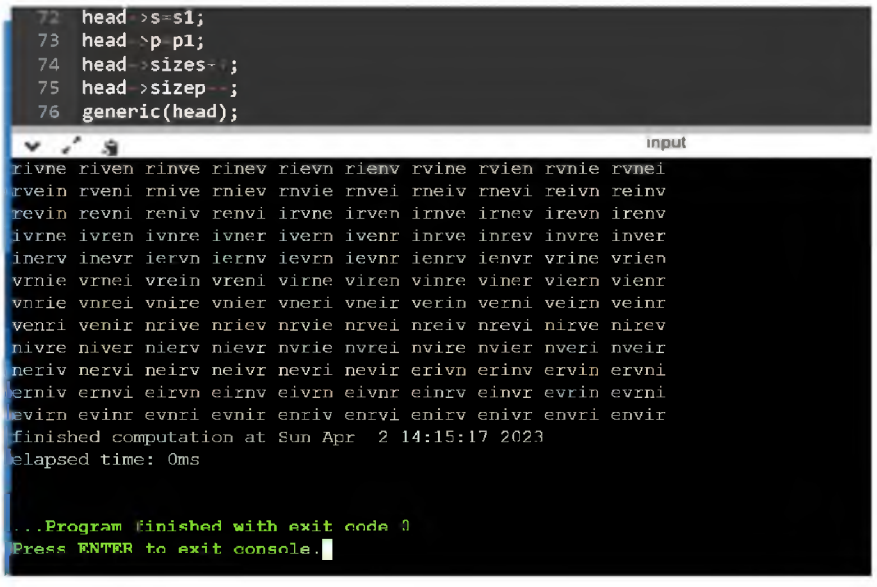

Рис.4.10. Результати тестування швидкодії методу перманентної декомпозиції генерації усіх перестановок, n=5

Як бачимо, метод перманентної декомпозиції за швидкодією майже однаковий з найшвидшим методом ШтайнмайєраДжонсонаТроттера. При цьому Heap-алгоритм значно поступається згаданим методам.

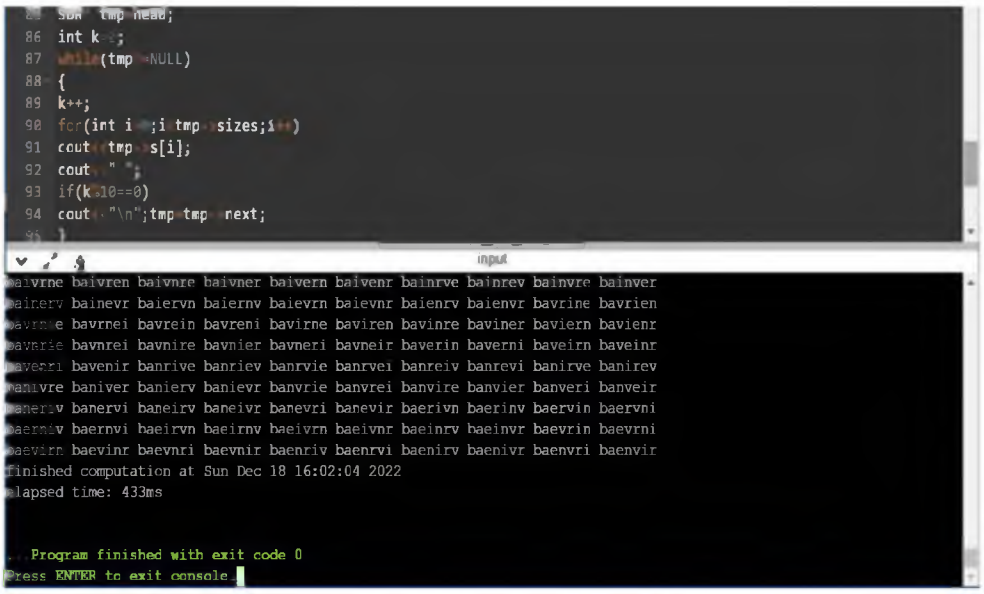

Рис.4.11. Результати тестування швидкодії методу перманентної декомпозиції генерації усіх перестановок, n=7

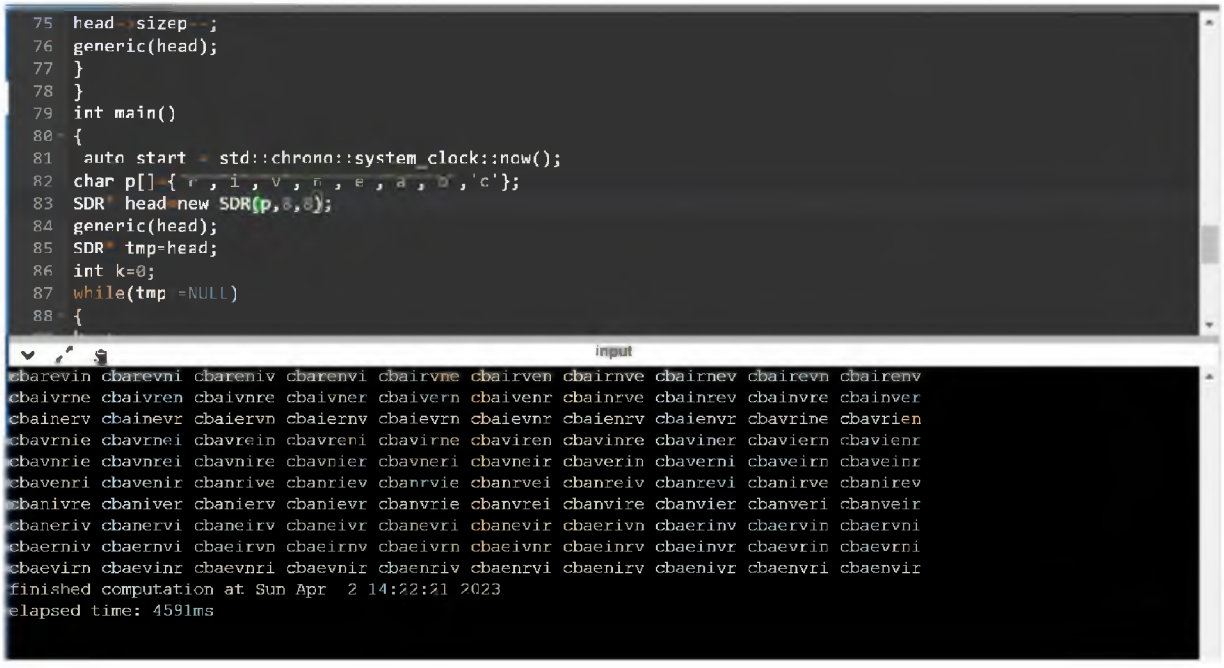

Рис.4.12. Результати тестування швидкодії методу перманентної декомпозиції генерації усіх перестановок, n=8

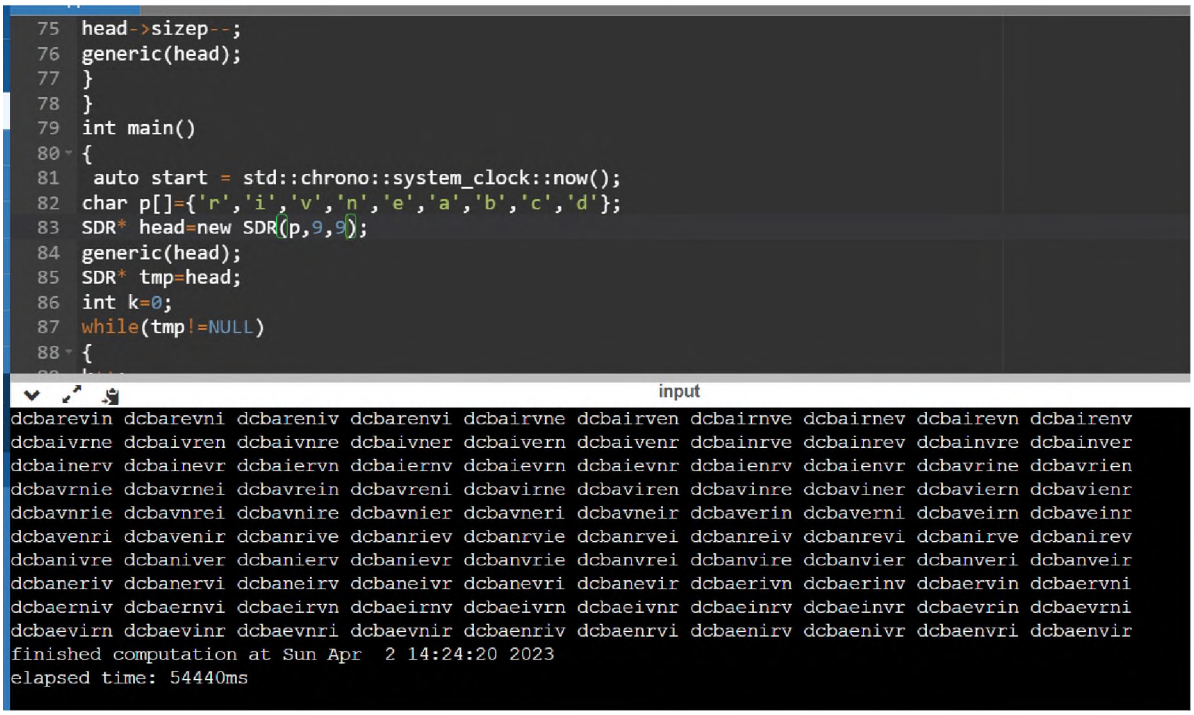

Рис.4.13. Результати тестування швидкодії методу перманентної декомпозиції генерації усіх перестановок, n=9

Розглянемо алгоритм Штайнмайєра-Джонсона-Троттера. На вхідних даних при величині масиву 4–6 результати аналогічні, менше 1ms.

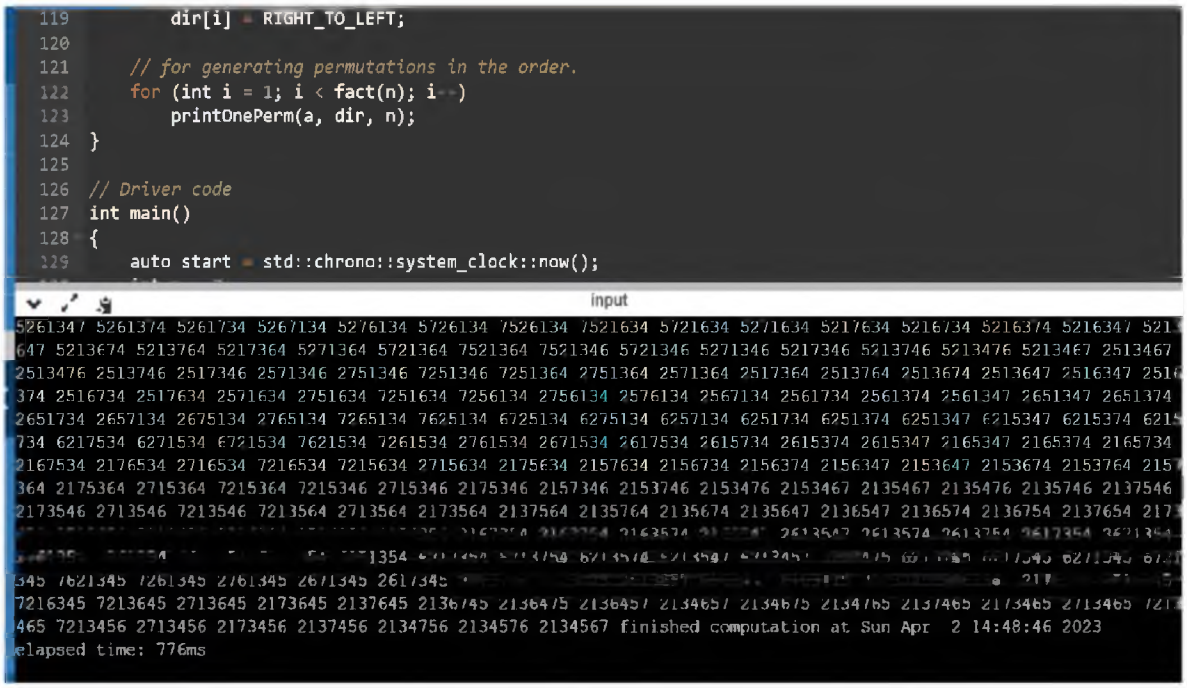

Рис.4.14. Результати тестування швидкодії методу Штайнмайєра–Джонсона–Троттера генерації усіх перестановок, n=7

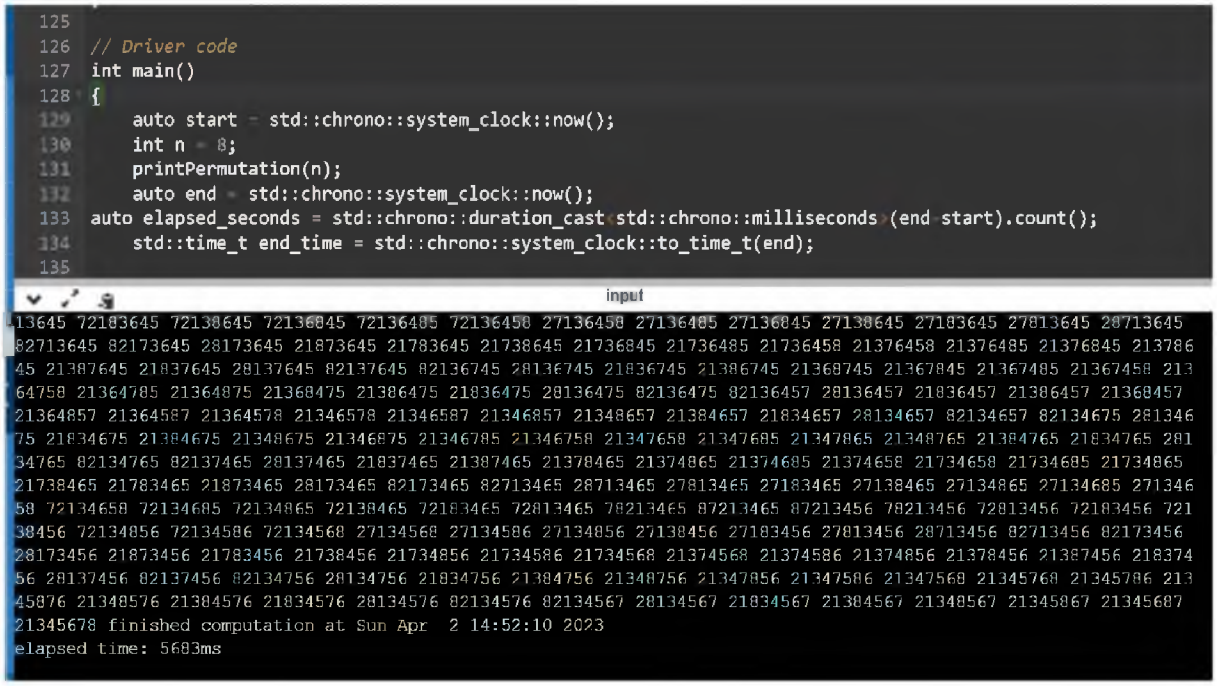

Рис.4.15. Результати тестування швидкодії методу Штайнмайєра-Джонсона-Троттера генерації усіх перестановок, n=8

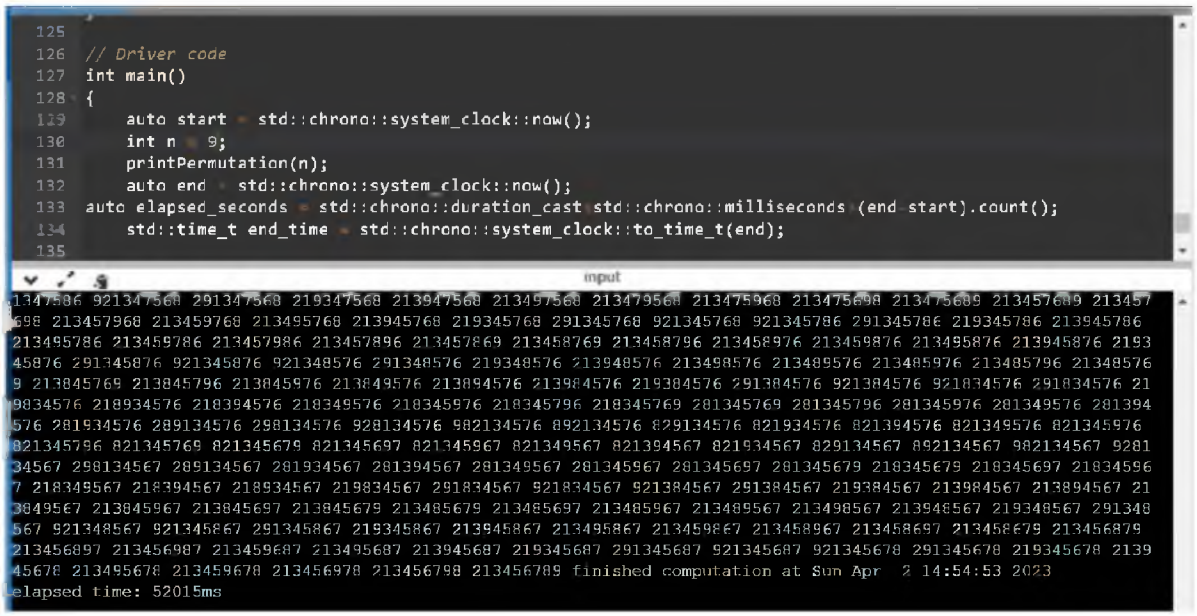

Рис.4.16. Результати тестування швидкодії методу Штайнмайєра-Джонсона-Троттера генерації усіх перестановок, n=9

Розглянемо Heap-алгоритм [155].

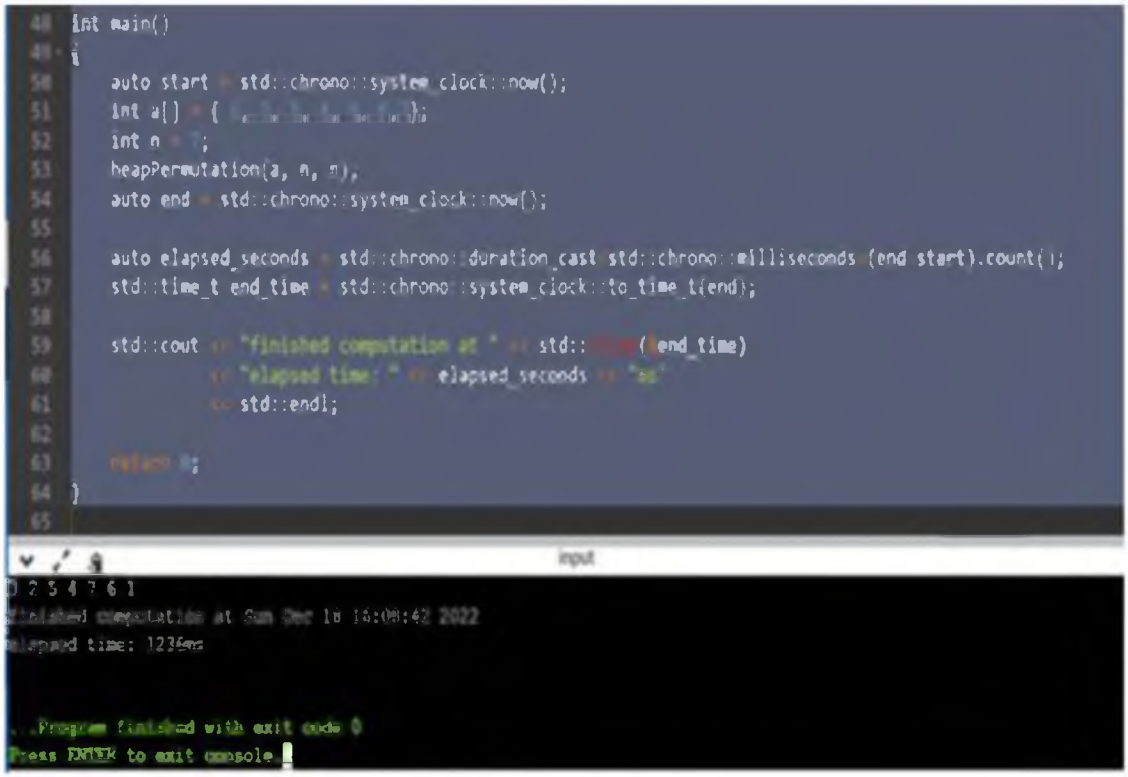

Рис.4.17. Результати тестування швидкодії Heap-алгоритму генерації усіх перестановок, n=7

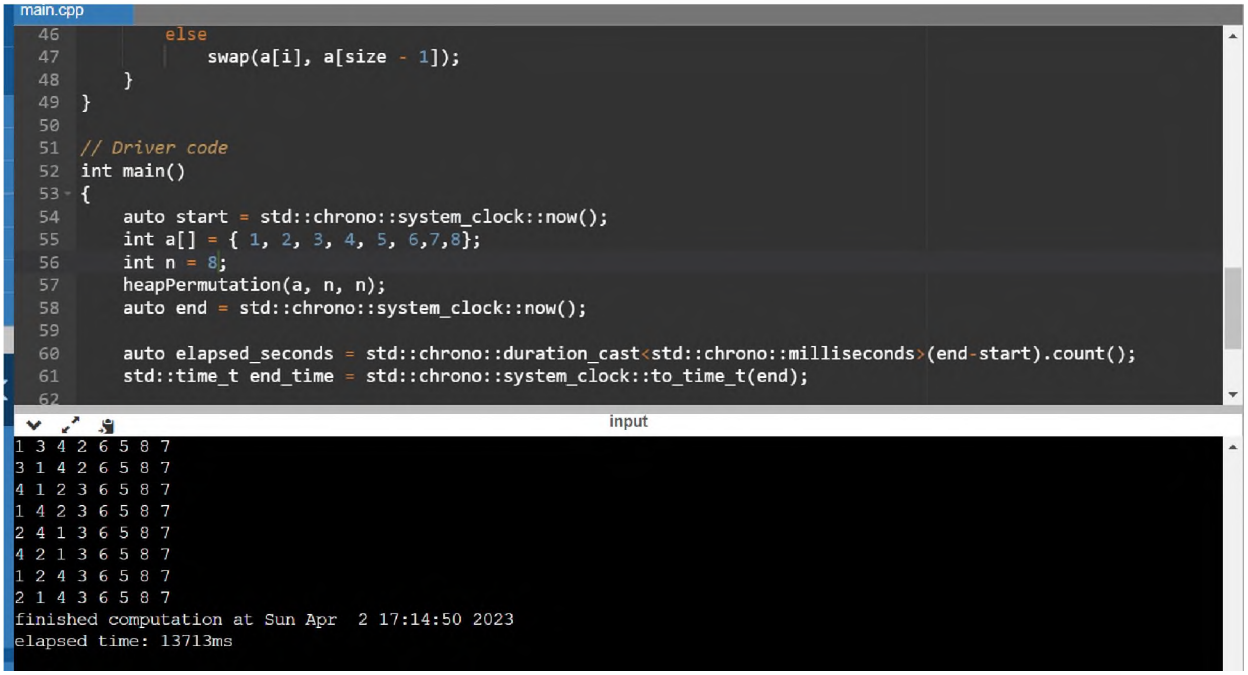

Рис.4.18. Результати тестування швидкодії Heap-алгоритму генерації усіх перестановок, n=8

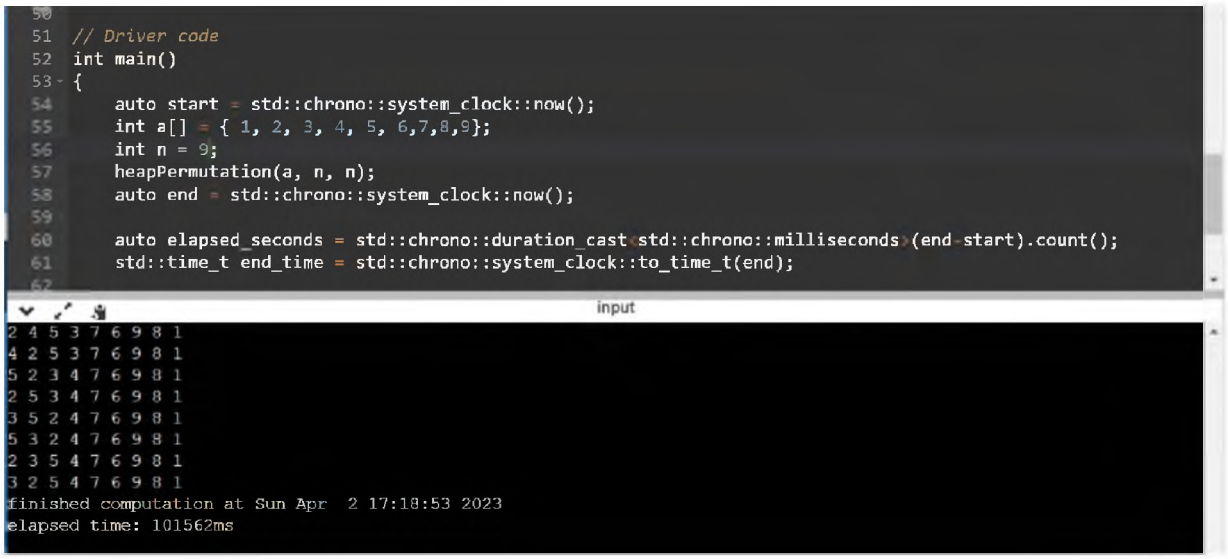

Рис.4.19. Результати тестування швидкодії Heap-алгоритму генерації усіх перестановок, n=9

Як бачимо, незважаючи на те, що класи складності алгоритмів однакові, як і передбачалось вище, метод перманентної декомпозиції втричі швидший за Heap–алгоритм, незважаючи на те, що Heap-алгоритм використовує лише одновимірний масив і всі комбінації не записуються в пам’ять.

Наступний крок – це генерація матриці розкладу.

Як бачимо, розклад згенеровано коректно повністю у відповідності до матриці інцидентності.

Розглянемо ілюстрацію роботи методу перманентної декомпозиції (рисунок 4.20.) для матриці інцидентності виду:

```
int z[4][5]={   1,65,2,3,4,
                0,1,1,1,0,
                1,1,0,1,0,
                1,1,0,0,1   };
```

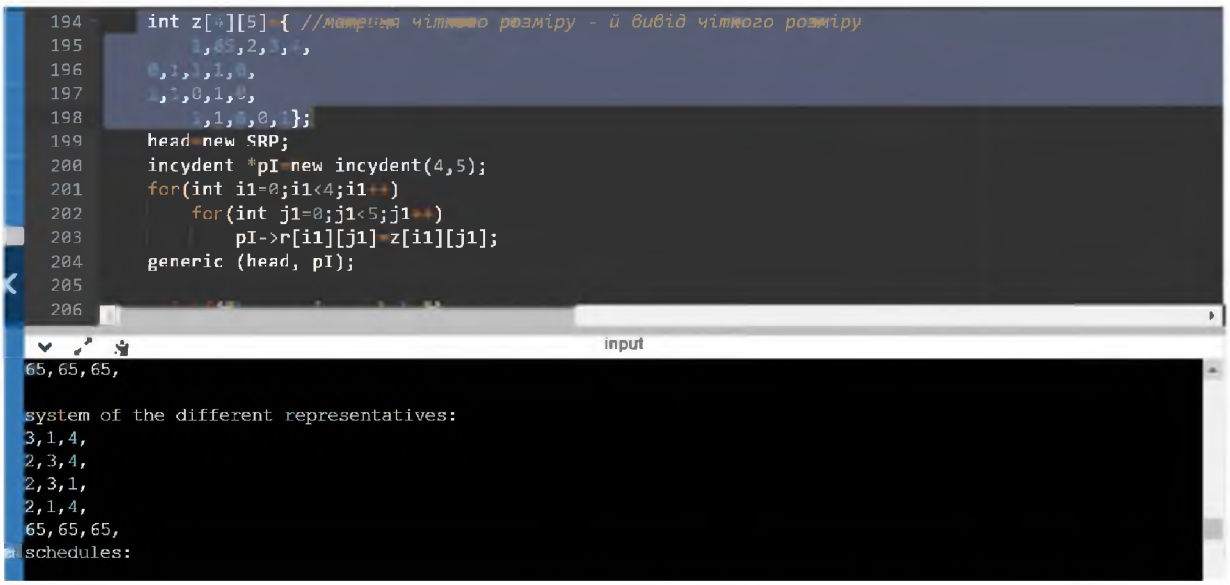

Рис.4.18. Ілюстрація роботи функції генерації СРПС методом перманентної декомпозиції.

Як бачимо, є п’ять викладачів з ідентифікаторами 1, 65, 2, 3, 4 та три групи, де вони повинні провести пари. Викладач 65 проводить поточну пару в усіх групах (в даному випадку ознакою поточності є номер викладача).

Результати роботи методу перманентної декомпозиції дає усі можливі комбінації систем різних представників стовпців матриці розкладів (рисунок 4.21).

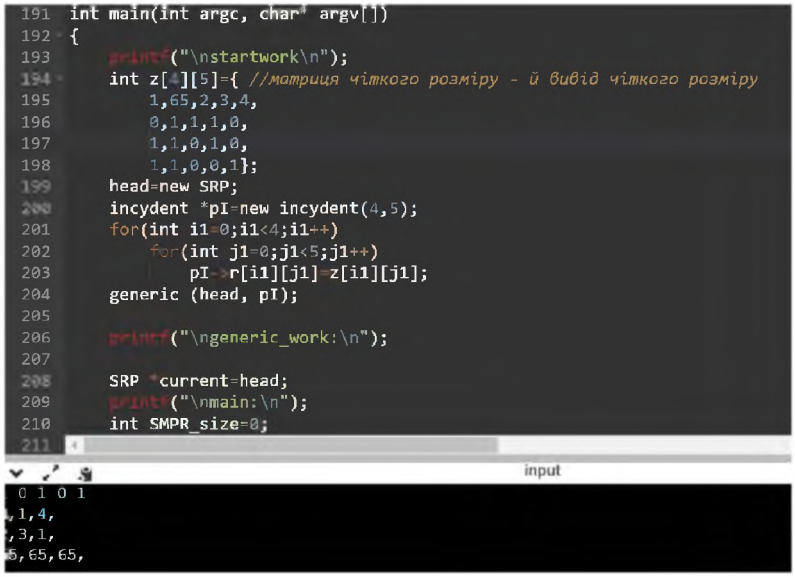

Рис.4.21. Приклад генерації матриці розкладу.

4.6. Висновки до четвертого розділу

Шляхом узагальнень результатів усіх попередніх розділів вперше сформульовано інформаційну технологію для розв'язання задачі генерації розкладів, що являє собою цілісну систему, яка поєднує низку підходів, зокрема конфігураційного підходу до аналізу вхідних даних та алгоритмів формування початкових допустимих матриць розкладів, перманентного підходу генерації СРПС та застосування алгоритмів перманентної декомпозиції, лексикографічного підходу на основі відповідних відношень порядку, алгебри адитивно–диз'юнктивних форм. Підхід, що пропонується, дуже зручний з точки зору програмної реалізації та побудови відповідної ієрархії класів. Він дозволяє власне створити інформаційну технологію як цілісну систему, що поєднує відповідні алгоритми та методи.

Вперше запропоновано інформаційну технологію складання розкладу занять, яка максимально враховує інтереси стейкхолдерів. Запропоновано власну систему кодування для представлення вхідних даних, яка оптимізована з точки зору ресурсів пам'яті – усі дані кодуються на бітовому рівні в межах цілих чисел. Така система представлення даних зумовлює використання побітових операцій у всіх реалізованих функціях.

Проведено експерименти, що підтверджують ефективність інформаційної технології генерації матриць розкладів. Зокрема, матриця розкладу, що була згенерована у тестових прикладах (факультет математики та інформатики РДГУ) була проаналізована з використанням критерію якості (2.2) та отримано числову оцінку ефективності рівну 0,994. Отримана оцінка свідчить про високу ефективність алгоритму генерації, зокрема, з використанням критеріїв відсутності «вікон» у викладачів та студентів.

Експериментально досліджувався метод перманентної декомпозиції для задач генерації комбінаторних об'єктів. Зокрема, для тестових задач генерації перестановок метод перманентної декомпозиції показав практично однакові результати з методом Джонсона-Троттера, який на сьогодні вважається одним

з найефективніших та є майже втричі швидшим у порівнянні з існуючими методами, такими як метод генерації на основі лексикографічного порядку чи Heap-алгоритм.

Основні результати розділу описані в роботах [11–13].

ВИСНОВКИ

У дисертаційній роботі розв'язано актуальне наукове завдання, яке полягає у покращенні ефективності технологій складання розкладів шляхом розробки інформаційної технології, в основі якої покладено методи перманентної декомпозиції. У результаті виконання роботи:

1. Введено поняття конфігурацій у матрицях розкладів та розроблено нові алгоритми, що дозволяють здійснити оптимізацію матриць розкладів, які мають спеціальну структуру, зокрема, складаються лише з тернарних, бінарних конфігурацій та їх комбінацій, можуть містити «нульові» елементи. Побудовано формальні критерії можливості оптимізації матриць розкладів за певними класами вимог, що має важливе значення для процедур аналізу вхідних даних при використанні широкого класу алгоритмів генерації розкладів та дозволяє покращити обчислювальну складність на 50%.

2. Запропоновано алгоритм формування модифікованої матриці інцидентності та визначення модифікованого перманента, що дає можливість побудувати ефективний алгоритм декомпозиції та алгоритм формування систем різних представників стовпців.

Відповідний алгоритм декомпозиції із «запам'ятовуванням» може бути використаний для розв'язання широкого класу задач генерації комбінаторних об'єктів. Застосування такого підходу дозволяє побудувати узагальнений алгоритм для генерації різнотипних комбінаторні об'єктів – від найпростіших перестановок до складних систем різних представників матриць розкладів. У роботі здійснено оцінку складності відповідних алгоритмів, зокрема, генерації перестановок на основі перманент і показано, що клас складності алгоритму $O(n!)$, що аналогічна відомим алгоритмами, зокрема, методу лексикографічно впорядкованого перебору. Теоретично показано, що складність перманентного алгоритму генерації перестановок на 46% менша складності алгоритму, що базується на відношенні порядку. Практичні експерименти

підтвердили ефективність методу перманентної декомпозиції, зокрема, для задачі генерації перестановок метод показав час роботи, що практично однаковий з методом Штайнмайєра-Джонсона-Троттера та у кілька разів кращий за Heap-алгоритм.

3. Запропоновано спеціальну алгебру адитивно-диз'юнктивних форм (АДФ), що містить дві операції – диз'юнкції та обов'язкового включення (додавання) та відрізняється алгоритмічним характером її операцій. Операція диз'юнкції означає вибір і, відповідно, дублювання відповідних списків, що містять диз'юнктивні форми, у процесі роботи рекурсивних процедур генерації. Операція обов'язкового включення означає просте включення відповідної СРПС як наступного рядка в біжучу матрицю розкладу.

На основі АДФ запропоновано алгоритми формування матриць розкладів, що дозволяють формувати матриці розкладу безпосередньо у процесі декомпозиції перманента модифікованої матриці інцидентності або утворювати АДФ на основі СМПР, що сформована як результат базового методу перманентної декомпозиції.

4. Розроблено інформаційну технологію для розв'язання задач генерації розкладів, що являє собою цілісну систему, яка поєднує низку підходів, зокрема конфігураційного підходу до аналізу вхідних даних та алгоритмів формування початкових допустимих матриць розкладів, перманентного підходу генерації СРПС та застосування алгоритмів перманентної декомпозиції, лексикографічного підходу на основі відповідних відношень порядку, алгебри адитивно–диз'юнктивних форм та відрізняється максимальним врахуванням інтересів стейкхолдерів.

Список використаних джерел

1. Turbal Y. V., Babych S. V. Methods of the schedule matrix forming based on the modified permanent. *Telekomunikacja i Elektronika. Zeszyty Naukowe. Uniwersytet Technologiczno–Przyrodniczy Im. Jana i Jedrzeja Sniadeckich w Bydgoszczy,* 2018. Vol. 268, No. 21. Pp. 85–92.

2. Turbal Y. V., Babych S. V., Kunanets N. E. Permanent Decomposition Algorithm for the combinatorial object's generation. *Radio Electronics, Computer Science, Control*. 2022. Vol. 2. Pp. 74–79. (Фахове видання України, WoS, категорія А).

3. Бабич С. В. Активне навчання студентів у проектній формі. Управління проектами. Порівняння проектного та операційного підходів. *Вісник НУВГП. Сер. Технології навчання.* 2015. Вип. 14. С. 88–94.

4. Бабич С. В., Турбал Ю. В. Методи формування матриць розкладів на основі модифікованих перманент. *Інформаційні системи та мережі*. 2017. №872. С. 204–209.

5. Бабич С. В. Алгоритм побудови допустимої матриці розкладів. *Вісник Національного університету водного господарства та природокористування. Сер. Технічні науки*. 2014. Вип. 4, ч. 68. С. 274–281.

6. Бабич С. В., Турбал Ю. В. Програмне забезпечення задач календарного планування на основі конфігураційних підходів. *Вісник НУВГП. Сер. Технічні науки*. 2022. Вип. 2, ч. 98. С. 258–267.

7. Turbal Y. V., Babych S. V., Bachyshyna L., Kunanets N. and Kovalchuk N. Modification of the Permanent Decomposition Method for the Meeting Schedule Problem: *The 1st International Workshop on Information Technologies: Theoretical and Applied Problems (ITTAP–2021)*. Ternopil, Ukraine, 2021. Pp. 126–131. (Scopus).

8. Бабич С. В. Інформаційна технологія генерації матриць розкладів згідно перманентної декомпозиції. *Міжнародний науково–технічний журнал*

«Вимірювальна та обчислювальна техніка в технологічних процесах». 2022. Вип. 4. С. 120–127. URL: https://doi.org/10.31891/2219-9365-2022-72-4-17.

9. Turbal Y. V., Babych S. V. Information technology for the schedule generation based on the algebra of additive–disjunctive forms and the modified method of permanent decomposition. *Computer sytems and information technologies*. 2022. Vol. 4, No. 9. Pp. 120–127. URL: https://doi.org/10.31891/CSIT.

10. Turbal Y. , Babych S., Kunanets N., Melnyk L. and Pasichnyk V. "Permanent" algorithm for the meeting shedule problem: *IEEE 16th International Conference on Computer Sciences and Information Technologies (CSIT)*. 2021. Pp. 355–359. URL: https://doi.org/10.1109/CSIT52700.2021.9648831. (Scopus).

11. Бабич С.В., Турбал Ю.В. Алгоритм формування матриці розкладів в задачах календарного планування: *Сучасні проблеми математичного моделювання та обчислювальних методів:* матеріали Всеукраїнської наукової конференції. Рівне, 2015. С. 6.

12. Бабич С.В., Турбал Ю.В., Турбал Ю.В. Алгоритм побудови допустимої матриці розкладів: *XXV International conference PDMU*. Odesa, Ukraine, 2015. Pp. 60–61.

13. Бабич С.В., Турбал Ю.В. Алгоритми оптимізації матриць розкладу за базовими критеріями в межах конфігураційного підходу: *XXIV International conference PDMU*. 2014. Skhidnytsia, Ukraine, 2015. Pp. 97–98.

14. Глибовець М.М., Гулаєва Н.М., Пасічник М.М. Паралельний генетичний алгоритм побудови розкладу занять. Проблеми програмування. 2015. Вип. 2. С. 76–85.

15. Leite Nuno, Melicio Fernando, Rosa Agostinho. A fast simulated annealing algorithm for the examination timetabling problem. *Expert Systems with Applications*. 2018. Vol. 122.

16. Mayor Bagul1, Sunil Chaudhari, Suita Nagare, Pushcart Patil, Kumavt K.S. A Novel Approach For Automatic Timetable Generation. *International Journal of Computer Applications* (0975-8887). 2015. Vol. 127, No. 10.

17. Anuja Chowdhary TIME TABLE GENERATION SYSTEM. 2014. Vol.3 Issue. 2. Pp. 410–414.

18. Mosaic Space Blog. The Practice and Theory of Automated Timetabling. *PATAT 2010, Mosaic Space Blog. University and college planning andmanagement*. 2011. URL: http://mosaicd.com/blog.

19. D. G. Maere. How Working Group Automated Timetabling was founded. 2010. URL: http://www.asap.ac.nott.ac.uk/ (date of access: 09.12.2011).

20. Anirudha Nanda An Algorithm to Automatically Generate Schedule for School Lectures Using a Heuristic Approach. *International Journal of Machine Learning and Computing*. 2012. Vol. 2, No. 4.

21. Thepphakorn T., Pongcharoen P., Hicks C. An ant colony based timetabling tool. *International Journal of Production Economics*. 2014. Vol. 149, No. 3. Pp. 131–144.

22. Wang B., Song Y., Wang C., Huang W. and Qin X. A Study on Heuristic Task Scheduling Optimizing Task Deadline Violations in Heterogeneous Computational Environments. *IEEE Access*. 2020. Vol. 8. Pp. 205635–205645. URL: https://doi.org/10.1109/ACCESS.2020.3037965.

23. Kumar M., Kaur L. and Singh J. Dynamic and Static Energy Efficient Scheduling of Task Graphs on Multiprocessors: A Heuristic. *IEEE Access*. 2020. Vol. 8. Pp. 176351–176362. URL: https://doi.org/10.1109.

24. Ekanayake T. W., Subasinghe P., Ragel S., Gamage A. and Attanayaka S. Intelligent Timetable Scheduler: A Comparison of Genetic, Graph Coloring, Heuristic and Iterated Local Search Algorithms. *International Conference on Advancements in Computing (ICAC)*, Malabe, Sri Lanka, 2019. Pp. 85–90. URL: https://doi.org/10.1109/ICAC49085.2019.9103403.

25. James C. Chena, Cheng Chun, Wub Chia, Wen Chenc, Kou–Huang Chend Flexible job shop scheduling with parallel machines using Genetic Algorithm and Grouping Genetic Algorithm. *Expert Systems with Applications*. 2012. Vol. 39, Issue 11. Pp. 10016–10021.

26. Baptiste Philippe, Carlier Jacques, Queyranne Maurice Properties of optimal schedules in preemptive shop scheduling. *Discrete Applied Mathematics*. 2011. Vol. 159. Pp. 272–280. URL: https://doi.org/10.1016/j.dam.2010.11.015.

27. Tan J. S., Goh S. L., Kendall G., Sabar N. R. A survey of the state-of-the-art of optimisation methodologies in school timetabling problems. *In Expert Systems with Applications*. 2021. Vol. 165. Pp. 113943.

28. Goh S. L., Kendall G., Sabar N. R. Improved local search approaches to solve the post enrolment course timetabling problem. *In European Journal of Operational Research*. 2017. Vol. 261, No. 1. Pp. 17–29.

29. Soria-Alcaraza J. A., Özcan E., Swan J., Kendall G., Carpio M. Iterated Local Search Using an Add and Delete Hyper-heuristic for University Course Timetabling. *In Applied Soft Computing*. 2016. Vol. 40. Pp.581–593.

30. Abuhamdah A., Ayob M., Kendall G., Sabar N.R. Population based Local Search for university course timetabling problems. *In Applied Intelligence.* 2014. Vol. 40, No. 1. Pp. 44–53.

31. JIA Zhao–hong, Yan Jianhai, Leung Joseph, Li Kai, Chen Huaping Ant colony optimization algorithm for scheduling jobs with fuzzy processing time on parallel batch machines with different capacities. *Applied Soft Computing*. 2018. Vol. 75. URL: https://doi.org/10.1016/j.asoc.2018.11.027.

32. Tang Lixin, Zhao Xiaoli, Liu Jiyin, Leung Joseph Competitive two–agent scheduling with deteriorating jobs on a single parallel-batching machine. *European Journal of Operational Research*. 2017. Vol. 263. URL: https://doi.org/10.1016/j.ejor.2017.05.019.

33. Arroyo José, Leung Joseph An effective iterated greedy algorithm for scheduling unrelated parallel batch machines with non–identical capacities and

unequal ready times. *Computers & Industrial Engineering*. 2017. Vol. 105. URL: https://doi.org/10.1016/j.cie.2016.12.038.

34. Nodari Vakhania, Frank Werner, Alejandro Reynoso Scheduling a single machine with compressible jobs to minimize maximum lateness. *arXiv preprint arXiv:2203.09800*, 2022.

35. Petukhin D., Velykodniy S., Kozlovskaya V. Modeling the space of possible states of the lesson schedule in higher education institutions. *International Scientific and Practical Conference" Intellectual Systems and Information Technologies*. Odesa State Environmental University, Odesa, 2021. Pp. 13–19.

36. Моркун В.С., Бурнасов П.В., Бурнасова Т.П. Принципи автоматизації формування нежорстких вимог до розкладу занять в університеті з боку викладачів. *Вісник Криворізького національного університету. Збірник наукових праць*. 2017. Вип. 45. С. 60–66.

37. Morkun V.S. The management of the resources educational institution. V.S. Morkun, P.V. Burnasov. *Metallurgical and Mining Industry*. 2014. Vol. 4. Pp. 56–61. URL: http://www.metaljournal.com.ua/assets/Journal/12.2014.pdf.

38. Моркун В.С. Методи визначення якості розкладу занять ВНЗ. В.С. Моркун, П.В. Бурнасов. *Вісник східноукраїнського національного університету імені Володимира Даля*. 2016. Вип. 1, ч. 225. С. 129–138.

39. Kysil′ V.V., Drach I.V., Kysil′ T.M. Model′ zadachi skladannya ta optymizatsiyi rozkladu zanyat′ za umovy zadovolennya ob″yektyvnykh ta sub″yektyvnykh vymoh navchal′noho zakladu [The model of the task of compiling and optimizing the class schedule, subject to the satisfaction of the objective and subjective requirements of the educational institution]. *Scientific notes of TNU named after V.I. Vernadsky. Series: technical sciences*. 2019. Vol. 30, No. 69. Ch. 1 No. 6. Pp. 65–70 [in Ukranian].

40. Snytyuk V.YE., Sipko YE.N. Aspekty formuvannya tsil′ovoyi funktsiyi v zadachi skladannya rozkladu zanyat′ u vyshchykh navchal′nykh zakladakh na osnovi sub″yektyvnykh perevah [Aspects of formation of the objective function in

the problem of scheduling classes in universities based on subjective advantages]. *Automation. Automation. Electrotechnical complexes and systems*. 2013. Vol. 2. Pp. 98–104 [in Ukranian].

41. Мулява І. Я. Вирішення задачі автоматизованого формування розкладу навчального закладу за допомогою генетичних алгоритмів. *Міжнародний науковий журнал "Інтернаука"*. 2018.

42. Снитюк В.Є., Сіпко Є.Н. Про особливості формування цільової функції та обмежень в задачі складання розкладу занять. *Математичні машини і системи.* 2014. Вип. 3. С. 67–76.

43. Томашевський В.М., Новіков Ю.Л., Камінська П.А. Складання розкладів занять у дистанційних системах навчання. *Вісник НТУУ КПІ. Серія «Інформатика, управління та обчислювальна техніка».* 2011. Вип. 52. С. 118–130.

44. Payus Carolyn, Mohd Ali, Siti Aishah, Vun, Leong Wan Environmental Science: Scientific Project Guidelines. 2015. ISBN: 978-967-0582-37-5.

45. Omotehinwa T.O., Oluwatosin Temidayo Examining the developments in scheduling algorithms research: A bibliometric approach. Omotehinwa, Temidayo Oluwatosin. Heliyon. 2022. Vol. 8, Issue 5, e09510. Open Access. Published: May 21, 2022. URL: https://doi.org/10.1016/j.heliyon.2022.e09510 (дата звернення 01.01.2023).

46. Dao S.D., Abhary K., Marian R. A bibliometric analysis of Genetic Algorithms throughout the history. *Comput. Ind.* Eng. 2017. Vol. 110. Pp. 395–403.

47. Rahimi I., Gandomi A.H., Deb K., Chen F., Nikoo M.R. Scheduling by NSGA-II: review and bibliometric analysis. *Processes.* 2022. Vol. 10, No. 1. Pp. 1–31.

48. Shishido H.Y., Estrella J.C. Bibliometric analysis of workflow scheduling in grids and clouds. *Proceedings – International Conference of the Chilean Computer Science Society, SCCC.* 2017. Pp. 1–9.

49. Edutimer | School Timetable Software | School Scheduler [2023 Rankings]. URL: https://www.edutimer.com (дата звернення 01.01.2023).

50. Yu J., Yang Z., Zhu S., Xu B., Li S., Zhang M. A bibliometric analysis of cloud computing technology research. *Proceedings of 2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference, IAEAC.* 2018. Pp. 2353–2358.

51. Almansour N., Allah N.M. A survey of scheduling algorithms in cloud computing. *International Conference on Computer and Information Sciences, ICCIS.* 2019.

52. Arora N., Banyal R.K. Hybrid scheduling algorithms in cloud computing: a review. *Int. J. Electr. Comput. Eng.* 2022. Vol. 12, No. 1. Pp. 880-895.

53. Arunarani A.R., Manjula D., Sugumaran V. Task scheduling techniques in cloud computing: a literature survey. *Future Generat. Comput. Syst.* 2019. Vol. 91. Pp. 407–415.

54. Ghafari R., Kabutarkhani F.H., Mansouri N. Task scheduling algorithms for energy optimization in cloud environment: a comprehensive review. *Cluster Comput.* 2022. Vol. 25, No. 2. Pp. 1035–1093.

55. Prajapati H.B., Shah V.A. Scheduling in grid computing environment. *International Conference on Advanced Computing and Communication Technologies, ACCT.* 2014. Pp. 315-324.

56. Maipan–uku J., Rabiu I., Mishra A. Immediate/batch mode scheduling algorithms for grid computing: a review. *Int. J. Regul. Govern.* 2017. Vol. 5, No. 7. Pp. 1–13.

57. Khalfay A., Crispin A., Crockett K. A review of technician and task scheduling problems, datasets and solution approaches. *Intelligent Systems Conference (IntelliSys).* 2017. Pp. 288–296. URL: https://doi.org/10.1109/IntelliSys.2017.8324306.

58. Zsolt Kosztyán T. An exact algorithm for the flexible multilevel project scheduling problem. *Expert Systems with Applications.* 2020. Vol. 158. Pp. 113485.

59. Zsolt Kosztyán T. Exact algorithm for matrix-based project planning problems. *Expert Systems with Applications*. 2015. Vol. 42, Issue 9. Pp. 4460–4473.

60. Guillermo 'Fred' Rivera Week Assignment-Scheduling, MRP and ERP. *University of Oxford, Module: Physics*, 2017.

61. Masuchun R., Masuchun W., Thepmanee T. Integrating m-Machine Scheduling into MRP. *Fourth International Conference on Innovative Computing, Information and Control (ICICIC*). Kaohsiung, Taiwan, 2009. Pp. 1365–1368. URL: https://doi.org/10.1109/ICICIC.2009.248.

62. Abdallah A Imetieg Project scheduling method with time using MRP system: A case study: Construction project in Libya. *The European Journal of Applied Economics, Centre for Evaluation in Education and Science (CEON/CEES)*. 2015. Pp. 58–66. URL: https://doi.org/10.5937/EJAE12–7815.

63. Sahu M., Singh A. V., Khatri S. K. A Classical Constraint Satisfaction Problem and its Solution using Artificial Intelligence. *Amity International Conference on Artificial Intelligence (AICAI)*. Dubai, United Arab Emirates, 2019. Pp. 429–433. URL: https://doi.org/10.1109/AICAI.2019.8701325.

64. Shen J., Mei D. The Freuder Width in a General Model of Constraint Satisfaction Problem. *9th International Symposium on Computational Intelligence and Design (ISCID)*. Hangzhou, China, 2016. Pp. 303–306. URL: https://doi.org/10.1109/ISCID.2016.1076.

65. Medema M., Lazovik A., The Community Structure of Constraint Satisfaction Problems and Its Correlation with Search Time. *IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*. Baltimore, MD, USA, 2020. Pp. 153-160. URL: https://doi.org/10.1109/ICTAI50040.2020.00034.

66. Wang Y. M., Yin H. L. Algorithm based on K–neighborhood search for no–wait job shop scheduling problems. *The 27th Chinese Control and Decision Conference (2015 CCDC)*. Qingdao, China, 2015. Pp. 4221–4225. URL: https://doi.org/10.1109/CCDC.2015.7162672.

67. Xu L., Yanpeng L., Xuan J. Based on Tabu Search and Particle Swarm Optimization Algorithms Solving Job Shop Scheduling Optimization Problems. *Fourth International Conference on Digital Manufacturing & Automation*. Shinan, China, 2013. Pp. 322–324. URL: https://doi.org/10.1109/ICDMA.2013.78.

68. Bhatia M., Bhatia M. S. Resource requirement prioritized grid scheduling model. *International Conference on Emerging Trends in Networks and Computer Communications (ETNCC)*. Udaipur, India, 2011. Pp. 388–391. URL: https://doi.org/10.1109/ETNCC.2011.6255927.

69. Chauhan A., Singh S., Negi S. Verma S. K. Algorithm for deadline based task scheduling in heterogeneous grid environment. *2nd International Conference on Next Generation Computing Technologies (NGCT)*. Dehradun, India, 2016. Pp. 219–222. URL: https://doi.org/10.1109/NGCT.2016.7877418.

70. Zhang J., Chen C., Zheng H. –k., Luo Q. –y. A High Priority Random Task Fuzzy Scheduling Algorithm for CPS. *Chinese Control And Decision Conference (CCDC)*. Nanchang, China, 2019. Pp. 482–487. URL: https://doi.org/10.1109/CCDC.2019.8832879.

71. Bahga A., Madisetti V. K. A Dynamic Resource Management and Scheduling Environment for Embedded Multimedia and Communications Platforms. *IEEE Embedded Systems Letters*. 2011. Vol. 3, No. 1. Pp. 24–27. URL: https://doi.org/10.1109/LES.2010.2092414.

72. Yu Y., Wang C., Zhu J. An Agent–Based Dynamic Scheduling Solution for Resource–Constrained Project Scheduling. *Fourth International Conference on Computational Intelligence, Modelling and Simulation*. Kuantan, Malaysia, 2012. Pp. 120–123. URL: https://doi.org/10.1109/CIMSim.2012.44.

73. Гайтан О.М Автоматизація генерації розкладу навчального процесу університету. *ІНФОРМАТИКА, ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА АВТОМАТИЗАЦІЯ. Вчені записки ТНУ імені В.І. Вернадського. Серія: технічні науки.* 2020. Том 31 (70), Вип.1, ч. 2. УДК 004.91:378.145, URL: https://doi.org/10.32838/2663–5941/2020.2–1/09.

74. Юрчак І. Ю., Москович Т. Р. Застосування генетичних алгоритмів в автоматизованій системі розподілу навчального навантаження. 2018. *Lviv Polytechnic National University Institutional Repository,* URL: http://eom.lp.edu.ua/sntk/doc/ksm2018/moskovytch.pdf (дата звернення 01.01.2023).

75. Демчук М.В., Мазурець О.В. *Хмельницький національний університет, Україна.* АВТОМАТИЗАЦІЯ ВЕДЕННЯ РОЗКЛАДУ ЗАНЯТЬ У ВУЗІ. 2014.

76. Sabri M. F. M., Husin M. H., Chai S. K. Development of a timetabling software using soft-computing techniques with a case study. *The 2nd International Conference on Computer and Automation Engineering (ICCAE).* Singapore, 2010. Pp. 394–397. URL: https://doi.org/10.1109/ICCAE.2010.5451223.

77. Widayu U. R. K., Mukhlason A. Nurkasanah I. Automation and Optimization of Course Timetabling Using the Iterated Local Search Hyper-Heuristic Algorithm with the Problem Domain. *International Timetabling Competition," 2021 3rd East Indonesia Conference on Computer and Information Technology (EIConCIT).* Surabaya, Indonesia, 2021. Pp. 134–138. URL: https://doi.org/10.1109/EIConCIT50028.2021.9431892.

78. Bong Chia Lih, Sze San Nah and Bolhassan N. A. A study on heuristic timetabling method for faculty course timetable problem. *9th International Conference on IT in Asia (CITA).* Sarawak, Malaysia, 2015. Pp. 1–3. URL: https://doi.org/10.1109/CITA.2015.7349832.

79. Egwuche O. S. Examination Timetabling with Graph Coloring for Emerging Institutions. *International Conference in Mathematics, Computer Engineering and Computer Science (ICMCECS).* Ayobo, Nigeria, 2020. Pp. 1–6. URL: https://doi.org/10.1109/ICMCECS47690.2020.246988.

80. Ekanayake T. W., Subasinghe P., Ragel S., Gamage A., Attanayaka S. Intelligent Timetable Scheduler: A Comparison of Genetic, Graph Coloring, Heuristic and Iterated Local Search Algorithms. *International Conference on*

Advancements in Computing (ICAC). Malabe, Sri Lanka, 2019. Pp. 85–90. URL: https://doi.org/10.1109/ICAC49085.2019.9103403.

81. Hemant Kumar Singh, Tapabrata Ray, Warren Smith Constrained Pareto simulated annealing for constrained multi-objective optimization. *Information Sciences*. 2010. Vol. 180, Issue 13. Pp. 2499–2513. ISSN 0020–0255. URL: https://doi.org/10.1016/j.ins.2010.03.021.

82. Koshino M., Otani T. Constraint propagation + Ant Colony Optimization for automated school timetabling. *IEEE 6th International Workshop on Computational Intelligence and Applications (IWCIA)*. Hiroshima, Japan, 2013. Pp. 107–111. URL: https://doi.org/10.1109/IWCIA.2013.6624795.

83. Ribić S., Konjicija S. Evolution strategy to make an objective function in two–phase ILP timetabling. *19thTelecommunications Forum (TELFOR) Proceedings of Papers*. Belgrade, Serbia, 2011. Pp. 1486–1489. URL: https://doi.org/10.1109/TELFOR.2011.6143838.

84. Бичков О.С. Когнітивні методи кібернетики : *Навчальний посібник. К.: Видавничо–поліграфічний центр "Київський університет"*. 2006. С. 127.

85. Fatma A. Omara, Mona M. Arafa Genetic algorithms for task scheduling problem. *Journal of Parallel and Distributed Computing*. 2010. Vol. 70, Issue 1. Pp. 13–22. URL: https://doi.org/10.1016/j.jpdc.2009.09.009.

86. Kamaljit Kaur, Amit Chhabra, Gurvinder Singh Heuristics Based Genetic Algorithm for Scheduling Static Tasks in Homogeneous Parallel System. *International Journal of Computer Science and Security (IJCSS)*. 2010. Vol. 4, Issue 2. Pp. 183–198.

87. Lawrence Davis D., Kenneth De Jong, Michael Vose D., Darrell Whitley L. Evolutionary Algorithms. *Springer Science & Business Media*. 2012. ISBN: 978–1–4612–1542–4.

88. Umut Beşikcia, Ümit Bilgea, Gündüz Ulusoyb Multi–mode resource constrained multi–project scheduling and resource portfolio problem. *European Journal of Operational Research*. 2015. Vol. 240, Issue 1. Pp. 22–31.

89. Wen Songa, Hui Xib, Donghun Kanga, Jie Zhangc An agent–based simulation system for multi–project scheduling under uncertainty. *Simulation Modelling Practice and Theory.* 2018. Vol. 86. Pp. 187–203.

90. Jha S.K. Exam Timetabling Problem using Genetic algorithm. *International Journal of Research in Engineering and Technology.* 2014. Vol.3, No5. Pp. 649–655.

91. Verma O.P., Garg. R., Bisht V.S., Optimal Time–Table Generation by Hybridized Bacterial Foraging and Genetic Algorithm. *Proceedings of International Conference on Communication Systems and Network Technologies (CSNT''12).* 2012. Pp. 919–923.

92. Годлевський М. Д. Дослідження ефективності паралельних генетичних алгоритмів для вирішення задачі створення розкладу занять вузу на базі Grid–системи. М. Д. Годлевський, О. О. Абабілов. *Радіоелектронні і комп'ютерні системи.* 2011. Вип. 3. С. 68–72.

93. Feizi Derakhshi M. R., Zandi M. A solution to the problem of university examinations timetabling through information related to the units taken previous terms. *The 2nd International Conference on Computer and Automation Engineering (ICCAE).* Singapore, 2010. Pp. 559-562. URL: https://doi.org/10.1109/ICCAE.2010.5451866.

94. Hiryanto L. Incorporating dynamic constraint matching into vertex–based graph coloring approach for university course timetabling problem. Yogyakarta, Indonesia, 2013. Pp. 68–72. URL: https://doi.org/10.1109/QiR.2013.6632539.

95. Najdpour N., Feizi–Derakhshi M. –R. A two–phase evolutionary algorithm for the university course timetabling problem. *2nd International Conference on Software Technology and Engineering.* San Juan, PR, USA, 2010. Pp. V2–266–V2–271. URL: https://doi.org/10.1109/ICSTE.2010.5608807.

96. University Timetabling. URL: https://www.unitime.org (дата звернення: 01.01.2023).

97. Топ-8 найкращих безкоштовних програм для створення онлайн-графіків [2021 рейтинг]. URL: https://uk.myservername.com/top-8-best-free-online-schedule-maker-tools (дата звернення: 01.01.2023).

98. Нова Школа Розклад. URL: https://start.rozklad.org (дата звернення: 01.01.2023).

99. Tools for teachers | APptavi. URL: https://www.apptavi.co.uk (дата звернення: 01.01.2023).

100. Skolaris: Online school timetable software. URL: https://skolaris.net/ (дата звернення: 01.01.2023).

101. Omniscol | Timetable and school management. URL: https://www.omniscol.com (дата звернення: 01.01.2023).

102. School scheduling software for Mac, PC, tablet and phone – Prime Timetable. URL: https://primetimetable.com (дата звернення: 01.01.2023).

103. Booking System for Education – ShoolBooking. URL: https://www.schoolbooking.com (дата звернення: 01.01.2023).

104. TimeTabler | Easy and Efficient School Timetabling Software. URL: https://www.timetabler.com (дата звернення: 01.01.2023).

105. Teachmint : School LMS & ERP | Integrated School Platform (ISP). URL: https://www.teachmint.com (дата звернення: 01.01.2023).

106. HR andPayroll Management Software | HR Software – Timelabs. URL: https://www.timelabs.in (дата звернення: 01.01.2023).

107. Free printable class schedule templates to customize | Canva. URL: https://www.canva.com/class–schedules/templates (дата звернення: 01.01.2023).

108. Free College Schedule Maker. URL: https://gizmoa.com/college–schedule–maker (дата звернення: 01.01.2023).

109. Free online schedule maker | Plan weekly activities. URL: https://schedulebuilder.org (дата звернення: 01.01.2023).

110. Adobe Spark. URL: https://www.adobe.com/express/learn/blog/welcome–to–adobe–spark (дата звернення: 01.01.2023).

111. Visme. URL: https://www.visme.co (дата звернення: 01.01.2023).

112. Free online meeting scheduling tool | Doodle. URL: https://doodle.com (дата звернення: 01.01.2023).

113. Appointment scheduling software and reservation booking system. URL: https://www.supersaas.com (дата звернення: 01.01.2023).

114. Coursicle | Plan your schedule and get into classes. URL: https://www.coursicle.com (дата звернення: 01.01.2023).

115. aSc TimeTables – School Scheduling. Best timetable software to create school timetable. URL: https://www.asctimetables.com (дата звернення: 01.01.2023).

116. The Ultimate Guide to the Best 10 Software for School Time Table. URL: https://scienceandliteracy.org/software–for–school–time–table (дата звернення: 01.01.2023).

117. Best School Time Table Software. URL: https://www.softwaresuggest.com/us/school–time–table–software (дата звернення: 01.01.2023).

118. Top 8 Best Free Online Schedule Maker Software [2023 Rankings]. URL: https://www.softwaretestinghelp.com/online–schedule–maker (дата звернення: 01.01.2023).

119. Burdett R.L., Kozan E. An integrated approach for scheduling health care activities in a hospital. *Eur. J. Oper. Res*. 2018. Vol. 264, No.2. Pp. 756–773.

120. Kumar M., Sharma S.C., Goel A., Singh S.P. A comprehensive survey for scheduling techniques in cloud computing. *Journal of Network and Computer Applications*. Academic Press, 2019. Vol. 143. Pp. 1–33.

121. Omotehinwa T.O., Igbaoreto A., Oyekanmi E. An improved round robin CPU scheduling algorithm for asymmetrically distributed burst times. *Afr. J. MIS*. 2019. Vol. 1, No. 4. Pp. 50–68.

122. Chandiramani, R. Verma, M. Sivagami A modified priority preemptive algorithm for CPU scheduling. Procedia Comput. *Sci*. 2019. Vol. 165. Pp. 363–369.

123. Nazar T., Javaid N., Waheed M., Fatima A., Bano H., Ahmed N. Modified shortest Job first for load balancing in cloud–fog computing. *Lecture Notes on Data Engineering and Communications Technologies, 25, Springer. Cham*, 2019. Pp. 63–76.

124. Yan P., Cai X., Ni D., Chu F., He H. Two–stage matching-and-scheduling algorithm for real-time private parking–sharing programs. *Comput. Oper. Res.* 2021. Vol. 125. Pp. 105083.

125. Omotehinwa T.O., Azeez S.I., Olofintuyi S.S. A simplified improved dynamic round robin (SIDRR) CPU scheduling algorithm. *Int. J. Informat. Proc. Commun.* 2019. Vol. 7, No. 2. Pp. 122–140.

126. Sharma D.K., Kwatra K., Manwani M., Arora N., Goel A. Optimized resource allocation technique using self-balancing fast MinMin algorithm. *Lecture Notes Data Eng. Commun. Technol.* 2021. Vol. 54. Pp. 473–487.

127. Sharma R., Nitin N., AlShehri M.A.R., Dahiya D. Priority-based joint EDF–RM scheduling algorithm for individual real–time task on distributed systems. *J. Supercomput.* 2021. Vol. 77, No. 1. Pp. 890–908.

128. Das N.K.C., George M.S., Jaya P. Incorporating weighted round–robin in honeybee algorithm for enhanced load balancing in cloud environment. *Proceedings of the 2017 IEEE International Conference on Communication and Signal Processing, ICCSP*. 2017. Pp. 384–389.

129. Ghosh S., Banerjee C. Dynamic time quantum priority based round robin for load balancing in cloud environment. *4th IEEE International Conference on Research in Computational Intelligence and Communication Networks, ICRCICN*. 2018. Pp. 33–37.

130. Choudhari T., Moh M., Moh T.S. Prioritized task scheduling in fog computing. *Proceedings of the ACMSE 2018 Conference*. 2018. Pp. 1–8.

131. Tychalas D., Karatza H. A scheduling algorithm for a fog computing system with bag-of-tasks jobs: simulation and performance evaluation. *Simulat. Model. Pract. Theor.* 2020. Vol. 98. Pp. 101982.

132. Almansour N., Allah N.M. A survey of scheduling algorithms in cloud computing. *International Conference on Computer and Information Sciences, ICCIS*. 2019.

133. Arora N., Banyal R.K. Hybrid scheduling algorithms in cloud computing: a review. *Int. J. Electr. Comput. Eng.* 2022. Vol. 12, No. 1. Pp. 880–895.

134. Ghafari R., Kabutarkhani F.H., Mansouri N. Task scheduling algorithms for energy optimization in cloud environment: a comprehensive review. *Cluster Comput*. 2022. Vol. 25, No. 2. Pp. 1035–1093.

135. Arunarani A.R., Manjula D., Sugumaran V. Task scheduling techniques in cloud computing: a literature survey. *Future Generat. Comput. Syst*. 2019. Vol. 91. Pp. 407–415.

136. Sana M.U., Li Z. Efficiency aware scheduling techniques in cloud computing: a descriptive literature review. *PeerJ Comp. Sci.* 2021. Vol. 7. Pp. 1–37.

137. Agrawal P., Gupta A.K., Mathur P. CPU scheduling in operating system: a review. *Lect. Notes Netw. Syst*. 2021. Vol. 166. Pp. 279–289.

138. Davis R.I., Burns A. A survey of hard real–time scheduling for multiprocessor systems. *ACM Comput. Surv*. 2011. Vol. 43, No. 4.

139. Olofintuyi S.S., Omotehinwa T.O., Owotogbe J.S. A survey of variants of round robin CPU scheduling algorithms. *FUDMA J. Sci*. 2020. Vol. 4, No. 4. Pp. 526–546.

140. Maipan–uku J., Rabiu I., Mishra A. Immediate/batch mode scheduling algorithms for grid computing: a review. *Int. J. Regul. Govern*. 2017. Vol. 5, No. 7. Pp. 1–13.

141. Prajapati H.B., Shah V.A. Scheduling in grid computing environment. *International Conference on Advanced Computing and Communication Technologies, ACCT*. 2014. Pp. 315–324.

142. Li J., Zheng G., Zhang H., Shi G. Task Scheduling Algorithm for Heterogeneous Real-time Systems Based on Deadline Constraints. *IEEE 9th International Conference on Electronics Information and Emergency*

Communication (ICEIEC). 2019. Pp. 113–116. URL: https://doi.org/10.1109/ICEIEC.2019.8784641.

143. Wang B., Song Y., Wang C., Huang W., Qin X. A Study on Heuristic Task Scheduling Optimizing Task Deadline Violations in Heterogeneous Computational Environments. *IEEE Access*. 2020. Vol. 8. Pp. 205635–205645. URL: https://doi.org/10.1109/ACCESS.2020.3037965.

144. Mbarka Soualhia, Foutse Khomh, Sofiene Tahar Failure Analysis of Hadoop Schedulers using an Integration of Model Checking and Simulation. *Software Engineering (cs.SE); Performance (cs.PF)*. EPTCS 342, 2021. Pp. 114–128. URL: https://doi.org/10.4204/EPTCS.342.10

145. Xu X., Tian Q., Xing Y., Yin B. Hu A. Large–Scale Data Intensive Heterogeneous Task Scheduling Method Based on Parallel GATS–TS Algorithm. *4th International Conference on Communications, Information System and Computer Engineering (CISCE)*. 2022. Pp. 482–485. URL: https://doi.org/10.1109/CISCE55963.2022.9851157.

146. Malewicz G., Rosenberg A. L., Yurkewych M. Toward a theory for scheduling dags in Internet–based computing. *IEEE Transactions on Computers*. 2006. Vol. 55, No. 6. Pp. 757–768. URL: https://doi.org/10.1109/TC.2006.91.

147. Sardaraz M., Tahir M. A Hybrid Algorithm for Scheduling Scientific Workflows in Cloud Computing. *IEEE Access*. 2019. Vol. 7. Pp. 186137–186146. URL: https://doi.org/10.1109/ACCESS.2019.2961106.

148. Wang Y., Tang L., Zhou Y., Zhu C., Zhang W., Yu L. Bottleneck Identification of Extended Flexible Job Shop Scheduling Problem. *6th International Symposium on Computational and Business Intelligence (ISCBI)*. 2018. Pp. 23–27. URL: https://doi.org/10.1109/ISCBI.2018.00015.

149. Yang M., Ma H., Wei S., Zeng Y., Chen Y., Hu Y. A Multi–Objective Task Scheduling Method for Fog Computing in Cyber–Physical–Social Services. *IEEE Access*. 2020. Vol. 8. Pp. 65085–65095. URL: https://doi.org/10.1109/ACCESS.2020.2983742.

150. Ali A., Iqbal M. M. A Cost and Energy Efficient Task Scheduling Technique to Offload Microservices Based Applications in Mobile Cloud Computing. *IEEE Access*. 2022. Vol. 10. Pp. 46633–46651. URL: https://doi.org/10.1109/ACCESS.2022.3170918.

151. Shao X., Kim C. S. An Adaptive Job Shop Scheduler Using Multilevel Convolutional Neural Network and Iterative Local Search. *IEEE Access*. 2022. Vol. 10. Pp. 88079–88092. URL: https://doi.org/10.1109/ACCESS.2022.3188765.

152. Huaxiu Yao, Yu Wang, Ying Wei, Peilin Zhao, Mehrdad Mahdavi, Defu Lian, Chelsea Finn Meta–learning with an Adaptive Task Scheduler. *35th Conference on Neural Information Processing Systems (NeurIPS 2021)*. 2021.

153. Liu L., Qi, D. An Independent Task Scheduling Algorithm in Heterogeneous Multi–core Processor Environment. *IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. 2018. Pp. 142–146. URL: https://doi.org/10.1109/IAEAC.2018.8577208.

154. Wang Bo, Wang Changhai, Huang Wanwei, Song Ying, Qin Xiaoyun Security–aware task scheduling with deadline constraints on heterogeneous hybrid clouds. *Journal of Parallel and Distributed Computing*. 2021. Vol. 153. Pp. 15–28. URL: https://doi: 10.1016/j.jpdc.2021.03.003.

155. Edutimer | School Timetable Software | School Scheduler [2023 Rankings]. URL: https://www.edutimer.com (дата звернення: 01.01.2023).

156. Knuth, D. E. The Art of Computer Programming. *Addison-Wesley*. 2005. Vol. 4. ISBN 0–201–85393–0.

157. Youssef Bassil A. Comparative Study on the Performance of Permutation Algorithms. *Journal of Computer Science & Research (JCSCR)*. 2012. Vol. 1, No. 1, Pp. 7–19.

158. Krishnamurthy Kirthi, Subhash Kak The Narayana Universal Code. *arXiv preprint arXiv:1601.07110*, 2016. URL: https://arxiv.org/ftp/arxiv/papers/1601/1601.07110.pdf.

ДОДАТОК А

Псевдокод окремих функцій в ілюстрації евристичного підходу складання розкладу занять

1. Приклад функції перевірки місця вставки

Вхідні дані: y1– позиція вставки, номер рядка, j–номер групи, i – ідентифікатор викладача, yt–позиція в бітовому представлення пари даного викладача і в групі j

```
function perevriad3
    for per ← 0 to j do
        if stbyte(rozk[y1][per]) = i or molbyte(rozk[y1][per]) = i then
            return 1
        end if
    end for
    if j mod 3 = 0 and j <= 57 then
        for nv ← 1 to 80 do
            if vydilibit(v[nv].group[j+1], yt+1) > 0 then
                if zabor[nv][y1][j+1] = 1 or rozk[y1][j+1] > 0 then
                    return 1
                end if
                for per ← 0 to j + 1 do
                    if (stbyte(rozk[y1][per]) = nv
                      or molbyte(rozk[y1][per]) = nv)
                      and nv! = i then
                        return 1
                    end if
                end for
            end if
        end for
```

```
for nv ← 1 to 80 do
                    if vydilibit(v[nv].group[j+2], yt+1) > 0 then
                        if zabor[nv][y1][j+2] = 1 or rozk[y1][j+2] > 0 then
                            return 1
                        end if
                        for per ← 0 to j + 2 do
                            if (stbyte(rozk[y1][per]) = nv
                              or molbyte(rozk[y1][per]) = nv)
                              and nv != i then
                                return 1
                            end if
                        end for
                    end if
                end for
                if j < 57 then
                    pi ← 1
                    while vydilibit(v[i].group[j+pi], yt+1) > 0 do
                        pi ← pi + 3
                    end while
                    if pi > 3 then
                        for asd ← j to j + pi – 1 do
                            if rozk[y1][asd] > 0 then
                                return 1
                            end if
                            if zabor[i][y1][asd] = 1 then
                                return 1
                            end if
                        end for
                    end if
```

```
                end if
                if vydilibit(v[i].group[j–3],yt+1) > 0
                        return 1
                end if
        end if
        return 0
end function
```

1. Приклад функції врахування побажань стейкхолдерів

Вхідні дані: namevykl – ПІБ викладача, namegr – назва групи, nurok – номер пари, myg – ідентифікатор змінні, pot – ідентифікатор потоку

```
fuinction pobajzv
        △ ngr – integer value
        △ nomvykl – integer value
        if strcmp(namegr, “MI11”) = 0 then
                ngr ← 0
        end if
        △ …
        for I ← 0 to 81 do
                if strcmp(v[i].name, namevykl) = 0 then
                        nomvykl ← i
                end if
        end for
        if pot = 0 then
                for y ← 0 to 24 do
                        if vydilibit(v[nomvykl].group[ngr], y + 1) > 0
                          and vydilibit(v[nomvykl].group[ngr + 3], y + 1) = 0 then
                                if  myg = 0 then
                                        rozk[nurok][ngr]  ←  rozk[nurok][ngr]  or
nomvykl
```

```
                        obnulbit(v[nomvykl].group[ngr], y + 1)
                        break
                    else
                        rozk[nurok][ngr] ← nomvykl << 8 (???)
                        obnulbit(v[nomvykl].group[ngr], y + 1)
                        break
                    end if
                else
                    △ potik
                end if
            end for
        end if
end function
```

2. Основна процедура формування розкладу :

Вхідні дані: v–масив класів викладачів

```
function formrozklad
    △ qwerty – integer value
    △ l – usignent integer value
    for j ← 0 to 60 do
        for I ← 0 to 80 do
            h1 ← 0
            l ← v[i].group[j]
            for y ← 9 to 24 do
                if vydilibit(l, y + 1) > 0 then
                    if j = 0 then
                        if vydilibit(l, 25 + h1) = 0 then
                            vstavkazv2(i, j, y)
                        else
                            vstavkazvmyg2(i, j, y)
```

```
                    end if
                end if
                if j = 1 then
                    if vydilibit(l, 25 + h1) = 0 then
                        vstavkazv1(i, j, y)
                    else
                        vstavkazvmyg1(i, j, y)
                    end if
                end if
                if j mod 3 = 0 and j > 1 then
                    if vydilibit(l, 25 + h1) = 0 then
                        qwerty ← vstavkazag2(i, j, y)
                    else
                        qwerty ← vstavkazagmyg2(i, j, y)
                    end if
                end if
                if j mod 3 > 0 and j > 2 then
                    if vydilibit(l, 25 + h1) = 0 then
                        qwerty ← vstavkazag1(i, j, y)
                    else
                        qwerty ← vstavkazagmyg1(i, j, y)
                    end if
                end if
                h1 ← h1 + 1
            end if
        end for
    end for
end for
end function
```

ДОДАТОК Б

Структури даних та функція генерації для методу перамнентної декомпозиції

Клас для представлення СРПС:

```
class SDR
{
public:
char *s;
char* p;
SDR* next;
int sizes;
int sizep;
int n;
SDR(char* _p, int _n, int psize)
{
n=_n; sizep=psize;
sizes=_n–psize;
p=new char[psize];
for(int i=0;i<psize;i++)
p[i]=_p[i];
next=NULL; }
SDR(SDR* head, int k)
{
sizes=1+head–>sizes;
n=head–>n;
sizep=n–sizes;
next=NULL;
s=new char[sizes];
```

```
p= new char[sizep];
for(int j=0;j<sizes–1;j++)
s[j]=head–>s[j];
s[sizes–1]=head–>p[k];
int l=0;
for(int i=0;i<sizep+1;i++)
if(i!=k)
p[l++]=head–>p[i];
}
                };
```

Функція генерації:

```
void generic(SDR* head)
{
if (head–>sizes<head–>n)
{
for(int i=head–>sizep–1;i>0;i—)
{
SDR *tmp=new SDR(head,i);
tmp–>next=head–>next;
head–>next=tmp;
generic(tmp);
}
char* s1=new char[head–>sizes+1];
char* p1= new char[head–>sizep–1];
for(int j=0;j<head–>sizes;j++)
s1[j]=head–>s[j];
s1[head–>sizes]=head–>p[0];
```

```
for(int i=0;i<head–>sizep–1;i++)
p1[i]=head–>p[i+1];
delete head–>s;
delete head–>p;
head–>s=0;
head–>p=0;
head–>s=s1;
head–>p=p1;
head–>sizes++;
head–>sizep—;
generic(head);
}
}
```

ДОДАТОК В

Структури даних та генерація АДФ

//генерує по списку СРПС початкову АДФ, аналізуючи перші елементи рядків СРПС:

```
ADF* startgener (SRP *head)
{
ADF *rez;
ADF* last=new ADF;
SRP* lasthead=head';
last before[1]='(';
if(head!=NULL) last–>context=head;
last–>next=NULL;
rez=last;
head=head–>next;
while(head!=NULL)
{
ADF * current=new ADF;
current–>context=head;
current–>next=NULL;
int i=0,j=0;
while(lasthead–>s[i]==lasthead–>s[i+1]) i++;
while(head–>s[j]==head–>s[j+1]) j++;
if((lasthead–>s[0]==head–>s[0]) && (i==j))
{
last–>after[1]='v';
current–>before[0]='v';
}
else{ last–>after[0]=')';
```

```
last–>after[1]='+';
current–>before[0]='+';
current–>before[1]='(';
}
last–>next=current;
last=current;
lasthead=head;
head=head–>next;}
if(head–>next==NULL)
current–>after[0]=')';
}
return rez;

int ident(int m,int j,SRP* si, incyd* pi)
{
if(si–>s[j]==pi–>r[0][j])
{
//непоточний елемент
if((j>0)&&(pi–>r[0][j]!=pi–>r[0][j–1] || (j==0))
return 1;
else// поточний елемент
{
for(int k=0;k<pi–>nmaxrow;k++)
if((pi–>r[k][j]>0)&&(si–>s[k]!=r[0][j]))
return 0;
return 1;
}
}
}
```

```
void generlexc(int n,int k, int *mas)
//генерує наступну найближчу комбінацію. Всі елементи масиву
//впорядковані
{
int barjer=n–1;
for(int l=k–1;l>0;l++)
{
if(mas[l]<barjer)
{
mas[l]++;
return 1;
}
barjer—;
}
return 0;
}

ADF* copyblock(ADF* startblock)
//копіює блок обовязкового включення одразу ж за startblock
{
ADF* current=startblock;
while((current–>next!=NULL)&&!((current–>after[0]==')')&& (current–
>after[1]=='v'))
current=current–>next;
//current став наприкінці блоку, вказує на останній елемент блоку
ADF* current1=startblock;
while((current1–>next!=NULL)&&!((current1–>after[0]==')')&&
(current1–>after[1]=='v'))
```

```
        {
        if(current1==startblock)
        ADF* startblock1=current–>next;
        ADF* tmp=new ADF;
        tmp–>context=current1–>context;
        tmp–>before[0]=   current1–>before[0];
        tmp–>before[1]=current1–>before[1];
        tmp–>after[0]=current1–>after[0];
        tmp–>after[1]=current1–>after[1];}
        tmp–>next=current–>next;
        current–>next=tmp;
        current=current–>next;
        current1=current1–>next;}
        return startblock1;
        }
        void genernewblock (int m,int j, int* mas,int size,ADF* startblock)
        //генерує новий блок з включеннями елементів у відповідності до
//вибраної комбінації mas номерів елементів српс, що включатимуться.
//Працює після goodblock
        {
        ADF* startblock1=copyblock(startblock);
        ADF* current=startblock1;
        ADF* start;
        int kilk=0,m=0;
        while((current–>next!=NULL)&&!((current–>next–>after[0]==')')&&
(current–>next–>after[1]=='v'))
        {
        if(current–>next–>before[1]=='(')
        start=current;
```

```
if(ident(m,j,current–>context)==1)
kilk++;
if(kilk==mas[m])
//видаляємо все крім визначеної СРПС current в дужках
{
m++;
ADF* nai=start;
nai1=nai–>next;
while(nai1!=current)
{
nai1=nai–>next;
delete nai;
ADF* nai=nai1;
}
nai=nai–>next;
while(nai1–>after[0]!=')')
{
nai1=nai–>next;
delete nai;
ADF* nai=nai1;
}
current–>before[0]='+';
current–>after[1]='+';
start–>next=current;
current–>next=nai1–>next;
}
//endif
else//kilk!=mas[m]
{
```

```
//видаляємо current
ADF* nai=start;
while(nai–>next!=current)
{
nai=nai–>next;
nai–>next=current–>next;
delete current;
}
}
}

int mjgeneric(ADF* head ,incydent* pi,int m,int j)
//генерує всі блоки обовязкового включення по m–j елементу матриці
//інцидентності та записує після head
{
//for(int j=0;j<pi–>nmaxcol;j++)
if(pi–>r[m][j]>0)
{
//рух по всіх блоках обовязкового включення, між якими стоять
дизюнкції
while(head!=NULL)
{
ADF* tmp=head, nexthead =nextblockadr(head);
int kv=0;
while(tmp!=nexthead)
{
if(ident(m,j,tmp–>context)==1)
kv++;
tmp=tmp–>next;
```

```
}
int index=pi–>r[m][j];
int *nu=new int[index];
for(int fi=0;fi<index;fi++)
nu[fi]=fi;
//головна частина
for(int i=0;i<factorial(kv)/( factorial(index)* factorial(kv–index);i++ )
{
tmp=head;
if(goodblock(tmp, nu, index, int m, int j, pi)
genernewblock( m,j,nu,index,tmp);
generlex(kv,index,nu};
}
//end for
// з блоком все , за ним за head нагенеровано всі варіанти можливі
блоків //обовязкового включення. Тепер потрібно помітити head для
подальшого //видалення і перейти за адресою nexthead
head=nexthead;
head–>delflag=1;//прапорець для подальшого видалення
}
//end while
}
//end if
}
```

ДОДАТОК Г

Акти впровадження наукових результатів

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ «РІВНЕНСЬКИЙ ФАХОВИЙ КОЛЕДЖ
НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ»
33001, м. Рівне, вул. Коперніка, 44 Тел./факс: (0362) 26-59-23
e-mail: rcnubip@nubip.edu.ua сайт: www.rcnubip.org.ua

від 18.12.2021 № 159/1

АКТ
про впровадження наукових результатів дисертаційної роботи БАБИЧА Сергія Васильовича на тему «Інформаційна технологія календарного планування згідно перманентної декомпозиції»

Ми, комісія у складі:

КОРСУН Ярослав Петрович, директора коледжу, Заслуженого працівника освіти України,

ЦАРУК Василь Юрійович, заступник директора з навчальної роботи, доктора економічних наук,

БАЛДИЧ Людмила Володимирівна, методист коледжу,

ЯКИМЧУК Ірина Олександрівна, завідувач відділення програмування,

склали цей акт про те, що у Відокремленому структурному підрозділі «Рівненський фаховий коледж Національного університету біоресурсів і природокористування України» впроваджена інформаційна система, що грунтується на принципах декомпозиції перманенту – календарного планування, що впроваджена у ВСП «РФК НУБіП України» як елемент програмного комплексу для складання розкладу занять навчальних груп відділення програмування.

Відповідний програмний комплекс використовується у локальній версії навчальною частиною коледжу.

Рекомендуємо включити інформаційну систему в освітній процес, як об'єкт дослідження під час викладання дисципліни «Управління проектами» та на відповідних практичних заняттях «Об'єктно-орієнтованого програмування» для

здобувачів освіти освітньо-професійного ступеня «Фаховий молодший бакалавр» та освітнього ступеня «Бакалавр».

Голова комісії

Директор коледж, Заслужений працівник освіти України — Ярослав КОРСУН

Члени комісії

Заступник директора з навчальної роботи, доктора економічних наук — Василь ЦАРУК

Методист коледжу — Людмила БАЛДИЧ

Завідувач відділення програмування — Ірина ЯКИМЧУК

Рівненська обласна рада

**КОМУНАЛЬНЕ ПІДПРИЄМСТВО
"РІВНЕНСЬКА ОБЛАСНА КЛІНІЧНА ЛІКАРНЯ
ІМЕНІ ЮРІЯ СЕМЕНЮКА"
РІВНЕНСЬКОЇ ОБЛАСНОЇ РАДИ**

вул. Київська, 78-г, м. Рівне, 33007, тел. (0362) 64-38-23, факс 64-33-02,
р/р UA333052990000026005030705566 в АТ КБ Приватбанк код 02000010 МФО 305299,
e-mail: kzrokl@gmail.com сайт: http://www.rokl.rv.ua

Акт

Про впровадження наукових результатів дисертаційної роботи Бабича Сергія Васильовича «Інформаційна технологія календарного планування згідно перманентної декомпозиції»

м. Рівне 12 грудня 2022 року

Члени комісії у складі:

ПІОНТКОВСЬКОГО Валентина Костянтиновича – заступника директора з питань реформування надання медичної допомоги, Заслуженого лікаря України, доктора медичних наук, доцента.

ВЕРЕЩУК Лариси Леонідівни – медичного директора з терапії

РОМАНЧУК Людмили Русланівні – завідувача відділу юридичного забезпечення та закупівель

МОНАСТИРСЬКОЇ Ірини Миколаївни – менеджера з персоналу, провідного склали цей акт про те, що в комунальному підприємстві «Рівненська обласна клінічна лікарня імені Юрія Семенюка» Рівненської обласної ради впроваджено інформаційну систему, що ґрунтується на принципах декомпозиції перманент та враховує побажання усіх стейкхолдерів для складання розкладу запису клієнтів до лікарів – спеціалістів амбулаторно – поліклінічного відділення та стаціонарних підрозділів через сервісний центр закладу.

В основі даної інформаційної системи використано наступні наукові результати Бабича С.В.:

- метод перманентної декомпозиції;
- алгебру аддиктивно-диз'юнктивних форм для складання відповідного розкладу.

ГОЛОВА КОМІСІЇ :

Заступник директора з питань реформування надання медичної допомоги, Заслужений лікар України, доктор медичних наук, доцент _____ Валентин ПІОНТКОВСЬКИЙ

ЧЛЕНИ КОМІСІЇ:

Медичний директор з терапії _____ Лариса ВЕРЕЩУК

Завідувач відділу юридичного забезпечення та закупівель _____ Людмила РОМАНЧУК

Менеджер з персоналу, провідний _____ Ірина МОНАСТИРСЬКА

АКТ

про впровадження наукових результатів дисертаційної роботи БАБИЧА Сергія Васильовича «Інформаційна технологія складання розкладу занять згідно перманентної декомпозиції»

від 12.04.2023 № 20

Комісія, у складі:

КЛАП Анатолій Віталійович, директора компанії ТОВ «СМАРТ КІНГ ЛТД.

КОВАЛЬЧУК Сергій Леонідович, головного бухгалтера.

РЕДЬКО Олександр Ігорович, системного адміністратора.

склали акт про те, що у ТОВ «СМАРТ КІНГ ЛТД» впроваджено інформаційну систему, що ґрунтується на принципах декомпозиції перманент для складання розкладів scrum – зустрічей.

В основі даної інформаційні системи було використано наступні наукові результати Бабича С.В.:

- метод перманентної декомпозиції;
- конфігураційний підхід до генерації матриць розкладів.

Голова комісії:

Директор компанії А.В. Клап

Члени комісії:

Головного бухгалтер С.Л. Ковальчук

Системного адміністрації О.І. Редько