

Запорізький національний університет  
Міністерство освіти і науки України

Кваліфікаційна наукова праця  
на правах рукопису

**СЄЛЮТІН ЄВГЕН КИРИЛОВИЧ**

**УДК**

**ДИСЕРТАЦІЯ**

**ФРАГМЕНТАРНІ МОДЕЛІ В ЗАДАЧАХ ОПТИМАЛЬНОЇ КЛАСИФІКАЦІЇ**

113 прикладна математика

11 фізико-математичні науки

Подається на здобуття наукового ступеня доктора філософії. Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Є.К. Селютін

Науковий керівник

**Козін Ігор Вікторович**

доктор фізико-математичних наук, професор

Запоріжжя 2021

## АНОТАЦІЯ

*Селютін Є.К.* Фрагментарні моделі в задачах оптимальної класифікації. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 113 – Прикладна математика. – Запорізький національний університет, Запоріжжя, 2021.

Дисертаційна робота присвячена узагальненню і розробці теоретичних основ математичного апарату для побудови та дослідження фрагментарних моделей та метаевристичних методів для розв’язку задачі класифікації. Розглянуті в дисертації моделі та методи можуть використовуватися під час пошуку розв’язків у багатьох наукових та практичних задачах, в тому числі, задачі розміщення виробництва.

У **вступі** обґрунтовано актуальність теми дисертаційної роботи, сформульовано мету та основні задачі дослідження, показано їх зв’язок з науковими програмами. Визначено методи дослідження, наукову новизну та практичне значення отриманих результатів.

У **розділі 1** проведено огляд результатів за тематикою дисертаційної роботи. Обґрунтовано вибір напрямків подальших досліджень, пов’язаних з розв’язком задачі оптимальної класифікації. Розглянуто та проаналізовано існуючі задачі оптимальної класифікації та методів пошуку їх розв’язків. Досліджено обчислювальну складність існуючих задач класифікації.

Виявлено, що постановка задачі пошуку оптимальної класифікації дозволяє віднести цю задачу до багатокритеріальних. Майже всі математичні методи оптимізації призначено для відшукування оптимального рішення однієї функції –

одного критерію. Тому одним з варіантів вирішення багатокритеріальних завдань є її приведення до однокритеріальною з одним узагальненим критерієм.

Проаналізовано, що для вирішення задачі класифікації досить тривалий час використовувалися методи найближчого сусіда (Nearest Neighbor), K-найближчого сусіда (k-Nearest Neighbor); байєсовські мережі (Bayesian Networks); індукція дерев рішень; нейронні мережі (neural networks), метод опорних векторів; статистичні методи, зокрема, лінійна регресія; класифікація cbr – методом або за допомогою генетичних алгоритмів. Але більш ефективними себе показали метаевристичні методи.

Виявлено, що більшість задач оптимальної класифікації є важкорозв'язуваними (складними в обчислювальному сенсі), оскільки до них поліноміально зводиться хоча б одна NP-повна задача. Для таких задач на сьогодні невідомі алгоритми пошуку точного розв'язку простіших, ніж повний перебір всіх допустимих розв'язків задачі. Тому є сенс шукати прості наближені алгоритми, які хоч і не дають точного розв'язку, але мають високу швидкодію. Серед таких алгоритмів виділяється клас «жадібних» алгоритмів.

Виділено невіршені задачі класифікації, зокрема, розпізнавання спаму електронної пошти, зображень та мовлення в якості цифрового паролю, опис мікроматриці ДНК тощо.

У **розділі 2** наведено ряд основних понять та результатів, які використовуються в роботі та стосуються комбінаторного об'єкту «фрагментарна структура». Вивчено особливості та властивості фрагментарних структур, встановлено зв'язок з категорією «опуклість» та задачею покриття графів. Досліджено поняття метаевристики та проаналізовано метаевристичні методи пошуку оптимальних рішень у задачі класифікації. Встановлено зв'язок між задачею оптимальної перестановки та класифікації.

Визначено, що задачі оптимальної класифікації на дискретних множинах мають велику кількість переваг. Зокрема, вони адекватно відображають нелінійні залежності, неподільність об'єктів, враховують логічні та технологічні обмеження, а також «якісні» вимоги. Але у прикладних задачах, як правило, є багато обмежень, які ускладнюють застосовність відомих алгоритмів. Тому для таких задач є виправданим застосування метаевристик.

Виявлено, що експериментальні дослідження розподілу локальних оптимумів свідчать про високу концентрацію їх в безпосередній близькості від глобального оптимуму (гіпотеза про існування «великої долини» для задач на мінімум або «центрального гірського масиву» для задач на максимум).

Показано, що проблема пошуку початкових популяцій для реалізації еволюційно-фрагментарної моделі зводиться до *задачі покриття (або розбиття) простору перестановок опуклими множинами*.

У **розділі 3** розглянуто фрагментарний алгоритм покриття графу зірками. Проаналізовано найпростіші варіанти задачі розміщення виробництва з точки зору задачі класифікації, а також наведено метаевристичні алгоритми пошуку оптимальних рішень складних задач розміщення виробництва.

Сформульована задача покриття: заданий граф  $G = (V, E)$ , ребра якого  $E$  зважені функцією  $p: E \rightarrow R_+^1$ . Знайти підмножину мінімальної ваги з неперетинних по вершинах зірок у цьому графі, об'єднання яких містить всі вершини графу  $G$ .

Побудовано фрагментарну модель задачі. Множиною елементарних фрагментів є множина  $E$  всіх ребер графу, елементи якого нумеруються числами  $1, 2, \dots, m$ .

Визначено умову приєднання елементарного фрагменту: або ребро  $e_i \in E$  має рівно одну спільну вершину з однією з зірок фрагмента і ця вершина – центр зірки, або це ребро не має загальних вершин з уже обраними ребрами. Розглядаються довільні впорядкування ребер графу, кожне з яких описується перестановкою з групи перестановок  $S_m$ .

Розглянуто задачі розміщення виробництва – одні з найбільш поширених та актуальних в наш час, що мають масовий характер.

Визначено, що найкращі результати при розв'язанні задачі про розподіл економічного навантаження з урахуванням впливу на навколишнє середовище демонструє метод рою часток із підбором коефіцієнтів соціалізації та персоналізації на основі генетичного алгоритму. В тому числі, ці результати є кращими за результати генетичного алгоритму, який є досить відомим при розв'язанні даної задачі. Більше того, результати генетичного алгоритму є найгіршими у порівнянні із усіма розглянутими алгоритмами. Визначена доцільність використання методу рою часток та фрагментарної моделі, а також їх модифікацій.

У **розділі 4** було запропоновано програмні реалізації генерації випадкових графів та розв'язку задачі класифікації за допомогою метаевристичних алгоритмів. Було проведено порівняння ефективності роботи алгоритмів.

Наведено алгоритм генерації графів з 50 вершинами у розрідженому (50 ребер) та насиченому (500 ребер) варіантах. Розріджений граф не є зв'язковим, оскільки кожна його вершина з'єднана тільки з невеликою кількістю інших вершин; насичений граф, безсумнівно, є зв'язковим, тому що кожна його вершина пов'язана в середньому з 20 іншими вершинами.

Розглянуто реалізацію створення випадкових ребер. Для заданої кількості вершин  $V$  генеруються довільні ребра, тобто пари випадкових чисел від 0 до  $V-1$ . Результатом, швидше за все, буде довільний мультиграф з петлями. Будь-яка пара може містити два однакових числа (тобто можливі петлі); і будь-яка пара може повторитися кілька разів (тобто можливі паралельні ребра). Програма генерує ребра до тих пір, поки не набереться  $E$  ребер; рішення про видалення паралельних ребер залишається за реалізацією. Якщо видаляти паралельні ребра, то в насичених графах кількість генеруються ребер буде значно більше, ніж кількість

використаних ребер ( $E$ ); тому даний метод зазвичай використовується для розріджених графів.

Проведено оцінку якості метаевристичних алгоритмів розв'язку задачі класифікації на основі генерації випадкових графів за допомоги порівняння результатів роботи цього алгоритму з іншими алгоритмами на досить великій серії задач.

Для оцінки ефективності метаевристик на фрагментарних структурах було розроблено програму оцінки ефективності для різних модельних задач, в якій для багатьох дискретних задач, що допускають фрагментарну модель, були реалізовані універсальні алгоритми ряду метаевристик. Зокрема, реалізовані метод випадкового пошуку, метод ітеративного локального пошуку, метод імітації відпалу, еволюційно-фрагментарний алгоритм, метод перемішаних стрибаючих жаб. У програмі реалізовані генератор випадкових завдань з різними обмеженнями, база даних завдань і програма порівняння ефективності алгоритмів.

Для задачі покриття графа зірками проведено чисельний експеримент на базі 53 випадково згенерованих завдань. Розглядалися зв'язкові графи з числом вершин від 20 до 50 і з щільністю ребер 0,5-0,8. Для кожної з задач було побудовано фрагментарну модель і застосовувалася група алгоритмів (випадковий пошук, еволюційно-фрагментарний алгоритм, метод імітації відпалу тощо). Параметри алгоритмів підбиралися таким чином, щоб трудомісткість обчислень була приблизно однаковою.

Результати порівняння різних алгоритмів показують, що жоден з них не володіє явною перевагою перед іншими. Це побічно підтверджує відому теорему «про відсутність безкоштовних обідів» для метаевристик. Таким чином, в умовах реальної експлуатації розумно застосовувати не одну, а кілька метаевристик і вибирати найкращий результат. Розглянутий в дисертаційній роботі метод використання фрагментарних моделей для пошуку субоптимальних рішень задач

класифікації, дозволяє порівняно просто побудувати універсальну комп'ютерну систему для таких завдань. Причому універсальними будуть програми реалізації метаевристик, а індивідуальними методи побудови фрагментарних моделей і алгоритми розрахунку значень критеріїв.

**Ключові слова:** фрагментарна модель, метаевристичні методи, генетичний алгоритм, метод стрибаючих жаб, задача оптимальної класифікації.

## ABSTRACT

Selyutin E.K. Fragmentary models in optimal classification problems. – Qualifying scientific work on the rights of the manuscript.

The dissertation on competition of a scientific degree of the doctor of philosophy on a specialty 113 Applied Mathematics. – Zaporizhzhya National University, Zaporizhzhya, 2021.

The dissertation is devoted to the generalization and development of the theoretical basis for the mathematical apparatus for constructing and researching fragmentary models and metaheuristics methods for solving a classification problem. The models and methods considered in the thesis can be used for resolving many scientific and practical problems, including the problem of manufactory location.

The introduction substantiates the relevance of the study, formulates major goals and objectives of the research, shows their connection with scientific programs. Defined research methods, scientific novelty and practical significance of the results.

**The Chapter 1** shows the overview of the results on the subject of the dissertation work. The choice of directions of further research related to the solution of the optimal classification problem is substantiated. The existing problems of optimal classification and methods of searching for their solutions are considered and analyzed. The computational complexity of existing problems of classification is investigated.

It was found that setting the task of finding the optimal classification allows you to attribute this task to multicritical. Almost all mathematical optimization methods are designed to find the optimal solution of one function – one copy. Therefore, one of the solutions to multi-copy problems is to bring it to a single-key with one generalized criterion.

It was analyzed that the methods of the nearest neighbor (Nearest Neighbor), K-Nearest Neighbor (k-Nearest Neighbor) were used to solve the classification problem for quite a long time; Bayesian Networks; induction of decision trees; neural networks, method of reference vectors; statistical methods, in particular, linear regression; classification CBR – by method or by means of genetic algorithms. But metaheuristic methods have shown themselves to be more effective.

It was found that most optimal classification tasks are difficult to solve (complex in the computational sense), since at least one NP-complete problem is polynomially reduced to them. For such problems, today unknown algorithms for finding an exact solution are easier than a complete selection of all permissible solutions to the problem. Therefore, it makes sense to look for simple approximate algorithms that, although they do not give an accurate solution, but have high performance. Among such algorithms stands out class of "greedy" algorithms.

Unsolved classification tasks such as e-mail spam recognition, images and speech as a digital password, description of DNA microarray are highlighted.

**The Chapter 2** provides a number of basic concepts and results that are used in the work and relate to the "fragmentary structure" combinatoric object. The peculiarities and properties of fragmentary structures were studied, the connection with the category "bulge" and the task of covering graphs was established. The concept of meta-heuristics is studied and metaheuristic methods of searching for optimal solutions in the problem of classification are analyzed. The connection between the task of optimal permutation and classification has been established.

It is determined that the problems of optimal classification on discrete sets have a large number of advantages. In particular, they adequately reflect nonlinear dependencies, the indivisibility of objects, take into account logical and technological restrictions, as well as "high-quality" requirements. But in applied tasks, as a rule, many

limitations complicate the applicability of known algorithms. Therefore, for such tasks, the use of meta-heuristics is justified.

It was found that experimental studies of the distribution of local optimums indicate a high concentration of them near of the global optimum (the hypothesis about the existence of a "great valley" for minimum tasks or "central mountain range" for maximum tasks).

It is shown that the problem of finding initial populations for the implementation of an evolutionary-fragmentary model is reduced to the task of covering (or breaking) the permutation space with convex sets.

**The Chapter 3** discusses the fragmentary algorithm for covering graph stars. The simplest options for the task of placing production in terms of the classification task are analyzed, as well as metaheuristic algorithms finding optimal solutions to complex production placement tasks are given.

The coverage problem is formulated: a given graph  $G = (V, E)$  whose edges  $E$  are weighted by a function  $p: E \rightarrow R_+^1$ . Find a subset of the minimum weight of non-intersecting vertices of stars in this graph, the union of which contains all the vertices of the  $G$ 's graph.

A fragmentary model of the problem is constructed. The set of elementary fragments is the  $E$  set of all edges of the graph, the elements of which are numbered  $1, 2, \dots, m$ .

The condition of joining an elementary fragment is determined: either the edge  $e_i \in E$  has exactly one common vertex with one of the stars of the fragment and this vertex is the center of the star, or this edge has no common vertices with the already selected edges. Arbitrary orderings of the edges of the graph are considered, each of which is described by a permutation from the group of permutations  $S_m$ .

The problems of production location, the most common and relevant in current time and which have a mass character, are considered.

It is determined that the best results in solving the problem of distribution of economic burden considering the impact on the environment demonstrate the method of swarming of particles with the selection of coefficients of socialization and personalization based on a genetic algorithm. In particular, these results are better than the results of the genetic algorithm, which is well known for solving this problem. Moreover, the results of the genetic algorithm are the worst in comparison with all the considered algorithms. The expediency of using the method of particle swarm and fragmentary model, as well as their modifications is determined.

**The Chapter 4** proposed software implementations for generating random graphs and solving the classification problem using metaheuristic algorithms. A comparison of the efficiency of the algorithms was performed.

An algorithm for generating graphs with 50 vertices in sparse (50 edges) and saturated (500 edges) variants is given. A sparse graph is not connected, since each of its vertices is connected to only a small number of other vertices; a saturated graph is undoubtedly connected, because each of its vertices is associated with an average of 20 other vertices.

The implementation of creating random edges is considered. For a given number of vertices  $V$ , arbitrary edges are generated, which means pairs of random numbers from 0 to  $V-1$ . The result is likely to be an arbitrary multigraph with loops. Any pair can contain two identical numbers (possible loops), and any pair can be repeated several times (parallel edges are possible). The program generates edges until  $E$  edges are typed; the decision to remove the parallel edges remains to be implemented. If you delete the parallel edges, then in saturated graphs the number of generated edges will be much greater than the number of used edges ( $E$ ); therefore, this method is usually used for sparse graphs.

The quality of metaheuristic algorithms for solving the classification problem based on the generation of random graphs is evaluated by comparing the results of this algorithm with other algorithms on a fairly large series of problems.

To estimate the effectiveness of metaheuristics on fragmentary structures, a program for evaluating the effectiveness of various model problems was developed, where for many discrete problems that allow a fragmentary model, universal algorithms of a number of metaheuristics have been implemented. In particular, the method of random search, the method of iterative local search, the method of simulation of annealing, the evolutionary-fragmentary algorithm, the method of mixed jumping frogs are implemented.

For the problem of covering the graph with stars, a numerical experiment was performed based on 53 randomly generated problems. Connected graphs with the number of vertices from 20 to 50 and with edges` density of 0.5-0.8 were considered. A fragmentary model was built for each of the problems and a group of algorithms was used (random search, evolutionary-fragmentary algorithm, annealing simulation method, etc.). The parameters of the algorithms were selected so that the complexity of the calculations was approximately the same.

The results of comparing different algorithms show that none of them has a clear advantage over the others. This indirectly confirms the well-known "No free lunch" theorem for metaheuristics. Thus, in the conditions of real operation, it is reasonable to apply not one, but several metaheuristics and choose the best result. The method of using fragmentary models for finding suboptimal solutions to classification problems considered in the dissertation allows building a relatively universal computer system for such problems. Moreover, the programs of realization of metaheuristics will be universal, and the methods of construction of fragmentary models and algorithms of calculation of values of criteria will be individual.

**Keywords:** fragmentary model, metaheuristic methods, genetic algorithm, jumping frog method, optimal classification problem.

## Список опублікованих праць за темою дисертації

### *Статті у наукових фахових виданнях України:*

1. Козін, І. В., **Селютін, Є. К.** Особливості пошуку оптимальних класифікацій: еволюційні алгоритми / Вісник Запорізького національного університету. Фізико-математичні науки, (2), 2020. – С. 62 – 68.

2. **Selyutin Y., Kozin I.** Comparative effectiveness of metaheuristic methods / Науковий вісник Ужгородського університету : серія Математика і Інформатика / редкол. : М. М. Маляр, Г. І. Сливка-Тилищак та ін. – Ужгород : Говерла, 2020. – Вип. 1 (36). – С. 105–111.

3. Козин И.В., **Селютин Е.К.** Метаэвристики для поиска оптимальных классификаций / Питання прикладної математики і математичного моделювання [Текст]: зб. наук. пр. / редкол.: О.М. Кісельова (відп. ред.) [та ін.]. – Дніпро, 2020. – Вип. 20. – С. 93 – 101.

4. Козін І. В., Максишко Н.К., **Селютін Є.К.** Використання еволюційних алгоритмів пошуку оптимальний класифікацій / Запорізький національний університет [Електронний ресурс]. – Режим доступу: <http://visnykznu.org/issues/2019/2019-econ-2/15.pdf>

### *Статті у наукових періодичних виданнях Європейського Союзу з наукового напрямку, з якого підготовлено дисертацію:*

5. Kozin I.V., **Selyutin E.K., Polyuga S.I.** Jumping frog method for optimal classifications / International Academy Journal Web of Scholar. 2(52). doi: 10.31435/rsglobal\_wos/30042021/7519

*Наукові праці, які засвідчують апробацію матеріалів дисертації:*

6. Selyutin Ye., Kozin I. Features of metaheuristic methods / Журнал «Молодий вчений». – Київ, 2021. – № 2. – С. 109 – 113.

7. **Селютин Е.К.** Применение метода прыгающих лягушек для задачи размещения производства / Комбінаторні конфігурації та їхні застосування: Матеріали ХХІІ Міжнародного науково-практичного семінару імені А.Я. Петренюка (Запоріжжя - Кропивницький, 15-16 травня 2020 року) / за ред. Г.П. Донця – Кропивницький: ПП «Ексклюзив-Систем», 2020. – С. 133 – 137.

8. **Selyutin Y., Kozin I.** The Metaheuristic Application in Classification Problems / Інформаційні технології: теорія і практика: Тези доповідей ІІІ-ї Всеукраїнської науково-практичної інтернет-конференції здобувачів вищої освіти і молодих учених, 2020 р., м. Харків) [Електронний ресурс] / Редкол. : М. В. Новожилова, І.О.Яковлева, Г. Л. Козіна, Г.В. Бакурова, Т.А. Желдак. Електрон. дані. – Харків : ХНУМГ імені О.М.Бекетова, 2020. – С. 22 – 24.

9. **Селютин Е.К.** Применение метода прыгающих лягушек для поиска оптимальных классификаций / Математичне та програмне забезпечення інтелектуальних систем (МПЗІС-2020): Тези доповідей ХVІІІ Міжнародної науково-практичної конференції, Дніпро, 18-20 листопада 2020 р. / Під загальною редакцією О.М. Кісельової. – Дніпро: ДНУ, 2020. – С. 227 – 229.

## ЗМІСТ

Перелік умовних позначень .....	6
Вступ.....	7
<b>РОЗДІЛ 1. ЗАДАЧІ КЛАСИФІКАЦІЇ. МОДЕЛІ І МЕТОДИ .</b> .....	<b>24</b>
1.1 Задачі класифікації. Оптимальні класифікації.....	24
1.2 Поняття відстані в задачах оптимальної класифікації.....	33
1.3 Обчислювальна складність. Складність задачі класифікації .....	35
1.4 Методи пошуку розв’язків задач оптимальної класифікації.....	37
1.5 Невирішені проблеми задачі класифікації. Типи задач класифікації.....	46
Висновки до розділу 1 .....	48
<b>РОЗДІЛ 2. ДИСКРЕТНІ ЗАДАЧІ КЛАСИФІКАЦІЇ. АЛГОРИТМИ НА ОСНОВІ МЕТАЕВРИСТИК</b> .....	<b>49</b>
2.1 Задача класифікації на дискретній множині .....	49
2.2 Зв'язок класифікації з задачею покриття графів.....	50
2.3. Підходи до пошуку оптимальних рішень на основі метаевристик .....	54
2.4 Опуклі множини в метричних просторах.....	56
2.5. Принципи побудови метаевристик на основі поняття опуклості.....	60
2.6. Фрагментарні структури і їх властивості .....	61
2.7. Зведення задач оптимізації до задачі пошуку оптимальної перестановки ..	63
2.8 Еволюційні метаевристики .....	65
2.9 Ройові метаевристики.....	68
Висновки до 2 розділу .....	71
<b>РОЗДІЛ 3. ЗАДАЧА РОЗМІЩЕННЯ ВИРОБНИЦТВА</b> .....	<b>74</b>
3.1 Задача покриття графу зірками.....	74
3.2 Найпростіші варіанти задачі розміщення виробництва.....	78

3.3 Метаевристичні алгоритми пошуку оптимального розв'язку складних задач розміщення виробництва .....	69
3.4 Фрагментарний алгоритм розв'язку задачі розміщення виробництва.....	82
3.5 Ройовий алгоритм для задачі розміщення виробництва.....	83
3.6 Алгоритм мурашиної колонії для задачі розміщення виробництва .....	87
Висновки до 3 розділу .....	88
<b>РОЗДІЛ 4. ПРОГРАМНІ РЕАЛІЗАЦІЇ АЛГОРИТМІВ ПОШУКУ ЗАДАЧ ОПТИМАЛЬНОЇ КЛАСИФІКАЦІЇ.....</b>	<b>90</b>
4.1 Генерація випадкових графів.....	90
4.2 Еволюційні алгоритми.....	94
4.3 Бібліотека тестових задач, що була використана для дослідження ефективності еволюційних методів.....	95
4.4 Алгоритм мурашиної колонії.....	90
4.5 Порівняння ефективності використаних алгоритмів .....	92
ВИСНОВКИ.....	110
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	113
ДОДАТОК А. ПРОГРАМНА РЕАЛІЗАЦІЯ НАЙПРОСТІШОЇ ГЕНЕРАЦІЇ ВИПАДКОВИХ ГРАФІВ .....	124
ДОДАТОК Б. ПРОГРАМНА РЕАЛІЗАЦІЯ ВИПАДКОВИХ ГРАФІВ .....	127
ДОДАТОК В. ПРОГРАМНА РЕАЛІЗАЦІЯ СИМЕТРИЧНОЇ МАТРИЦІ ВИПАДКОВОГО ГРАФУ .....	141
ДОДАТОК Г. БІБЛІОТЕКА ТЕСТОВИХ ЗАДАЧ РОЗМІЩЕННЯ ВИРОБНИЦТВА .....	145
Довідки про впровадження результатів дисертаційної роботи.....	152

## **ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ**

АІВ – алгоритм імітаційного відпалу

ГА – генетичний алгоритм

КО – комбінаторна оптимізація

МВС – метод вектору спаду

МГА – модифікований генетичний алгоритм

## ВСТУП

**Актуальність теми.** На сьогоднішній день інтенсивно розвивається інтелектуалізація методів обробки та аналізу даних. Цей напрямок покликаний мінімізувати зусилля особи, яка приймає рішення (ОПР), в процесі аналізу даних. Однією з найбільш поширених задач аналізу даних є класифікація. Ця оптимізаційна задача знайшла досить широке поширення в фінансовій сфері (прийняття рішення про видачу кредиту клієнту банку), медицині (діагностування захворювань), техніці (розпізнавання зображень і звуку) тощо.

В Україні основним науковим центром дослідження дискретних оптимізаційних задач є Інститут кібернетики НАН України ім. В. М. Глушкова (академік НАНУ І. В. Сергієнко, д.ф.-м.н., проф. В. В. Шило, д.ф.-м.н., проф. Г. П. Донець, д.ф.-м.н., проф. Н. В. Семенова та ін.). В Кропивницькому розв'язуванням задач на графах займається школа д.ф.-м.н., проф. А. Я. Петренюка. В Запоріжжі в області багатокритеріальної дискретної оптимізації працює школа В. О. Перепелиці (д.ф.-м.н., проф. В. О. Перепелиця, д.ф.-м.н., проф. І. В. Козін, д.е.н., проф. Н. К. Максишко) та Л. Н. Сергєєвої. Задачами геометричного проектування займаються в харківській науковій школі академіка НАН України В. Л. Рвачова та члена-кореспондента НАН України Ю. Г. Стояна, а також д.ф.-м.н. С. В. Яковлева і д.ф.-м.н. М. В. Новожилової. У Дніпрі працює відома наукова школа члена-кореспондента НАН України О. М. Кісельової, а також к.ф.-м.н., доц. В. А. Турчини, що досліджує оптимізаційні задачі розбиття.

В Європі у 1975 р. була заснована організація з дослідження операцій, The Association of European Operational Research Societies (EURO), яка є складовою частиною Міжнародної Федерації з дослідження операцій (The International

Federation of Operational Research Societies, IFORS). EURO включає в себе робочі групи, які займаються окремими питаннями дослідження операцій, зокрема, робоча група ESICUP (EURO Special Interest Group on Cutting and Packing) працює над задачами розкрою та упаковки, робоча група VeRoLog (EURO Working Group on Vehicle Routing and Logistics) займається задачами маршрутизації та логістики.

Задачами дискретної оптимізації займаються вчені Інституту математики ім. С. Л. Соболева (д.ф.-м.н. Е. Х. Гімаді, д.ф.-м.н. С. В. Севастьянов, д.ф.-м.н. Ю. А. Кочетов, д.ф.-м.н. О. О. Колоколов, д.ф.-м.н. В. В. Сервах, д.ф.-м.н. В. П. Іль'єв). В Білорусі в цій сфері працюють д.ф.-м.н. В. О. Ємелічев, д.ф.-м.н. В. М. Котов.

Гібридні моделі на основі комбінації фрагментарних моделей та евристичних алгоритмів різного роду дають можливість значно обмежити область допустимих рішень та знайти субоптимальні рішення. Отже, представлена робота, що спрямована на застосування цього універсального підходу до вирішення складних задач дискретної оптимізації, є своєчасною та актуальною.

**Зв'язок роботи з науковими програмами, планами, темами.** Дисертаційна робота виконувалася відповідно до планів наукових досліджень на кафедрі економічної кібернетики Запорізького національного університету МОНУ в межах науково-дослідної тем: «Моделювання складних соціально-економічних систем методами нелінійної динаміки та інтелектуального аналізу даних» (№ держ. реєстрації 0115U006765), «Інтелектуальні методи пошуку субоптимальних розв'язків прикладних дискретних оптимізаційних задач» (№ держ. реєстрації 0120U000375).

**Мета і задачі дослідження.** Метою роботи є застосування фрагментарних структур та метаевристичних методів для розв'язання задачі оптимальної класифікації.

Для досягнення цієї мети в роботі потрібно було розв'язати такі задачі:

- провести аналітичний огляд існуючих задач класифікації та методів пошуку їх розв'язків, у тому числі, на основі метаевристичних алгоритмів та методів;
- дослідити особливості задачі класифікації на дискретній множині;
- встановити зв'язок задачі класифікації із задачею покриття графу зірками та задачею розміщення виробництва;
- дослідити обчислювальну складність існуючих задач класифікації;
- дослідити особливості метаевристичних алгоритмів, у тому числі, еволюційних метаевристик;
- розробити фрагментарні моделі для пошуку субоптимальних рішень задачі оптимальної класифікації;
- довести властивість досяжності для розроблених моделей;
- запропонувати методи до оцінки якості метаевристик;
- розробити програмну реалізацію генерації випадкових графів та розв'язку задачі класифікації за допомогою метаевристичних алгоритмів;
- провести математичні експерименти для оцінки якості метаевристичних алгоритмів.

*Об'єкт дослідження:* дискретні задачі оптимальної класифікації.

*Предмет дослідження:* фрагментарні моделі та методи пошуку субоптимальних розв'язків дискретних задач оптимальної класифікації.

*Методи дослідження:* при розв'язанні поставлених задач у дисертації використовувалися методи теорії дискретної оптимізації, теорії фрагментарних структур, теорії графів, комбінаторного аналізу, теорії обчислювальної складності, теорії множин, імітаційного моделювання.

**Наукова новизна одержаних результатів.** У рамках розв'язання задач дисертаційного дослідження отримано такі основні наукові результати:

*Вперше:*

- запропоновано низку фрагментарних моделей та методів для пошуку субоптимальних рішень задачі оптимальної класифікації;
- доведено властивість досяжності для пошуку субоптимальних рішень у межах запропонованих моделей;

*Удосконалено:*

- низку метаевристичних методів (локальний пошук, еволюційний алгоритм, метод стрибаючих жаб та ін.) для розв'язку, зокрема, розміщення виробництва;

*Дістали подальшого розвитку:*

- методи оцінки якості метаевристик;
- методи побудови фрагментарних моделей на базі теорії графів.

**Практичне значення одержаних результатів.** Робота має як теоретичну, так і прикладну спрямованість. Розроблені фрагментарно-еволюційні моделі реалізовано в вигляді комп'ютерних програм, що використано в автоматизованих системах проектування та управління. Запропонований підхід та результати дослідження було впроваджено у навчальний процес Запорізького національного університету Міністерства освіти і науки України.

**Особистий внесок здобувача.** Всі результати, які складають суть дисертаційної роботи, одержані здобувачем самостійно. У публікаціях, які видані у співавторстві, автору належить: формалізація методу стрибаючих жаб; принцип дії еволюційно-фрагментарного алгоритму; опис задачі розміщення виробництва як задачі оптимальної класифікації; дослідження метаевристичних алгоритмів та порівняння їх ефективності стосовно розв'язку задачі оптимальної класифікації.

**Апробація результатів дисертації.** Основні положення дисертаційної роботи були представлені та доповідалися на наступних конференціях та семінарах: міжнародних науково-практичних конференціях «Математичне та програмне забезпечення інтелектуальних систем» (м. Дніпро, 2019, 2020 р.),

«Інформаційні технології: теорія і практика» (м. Кривий Ріг, 2019 р.).

**Публікації.** Основні результати за темою дисертації викладено у 8 опублікованих роботах, серед яких: 4 статті у наукових журналах і збірниках, що входять до переліків фахових видань, затверджених МОН України, 1 стаття в журналі, що індексується у наукометричній базі SCOPUS, 4 тез доповідей в збірниках праць наукових конференцій.

**Структура та обсяг дисертації.** Дисертаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел із 101 найменування (11 сторінок) та 4 додатків. Загальний обсяг роботи складає 153 сторінки, у тому числі 123 сторінки основного тексту, який містить 10 рисунків і 8 таблиць.

## РОЗДІЛ 1. ЗАДАЧІ КЛАСИФІКАЦІЇ. МОДЕЛІ І МЕТОДИ

### 1.1 Задачі класифікації. Оптимальні класифікації

Сьогодні в різних галузях людської діяльності накопичено великі обсяги інформації про різні матеріальні та нематеріальні сутності, їх властивості, поведінку та зв'язки. Обробка інформаційних масивів полегшується за рахунок використання ефективних методів класифікації.

Класифікація як найбільш продуктивний науковий метод пізнання дозволяє знаходити підходи до вирішення загального питання для багатьох областей досліджень — формування нових знань шляхом організації спостережуваних даних в наочні структури. Задача класифікації виникає практично у всіх областях і на всіх етапах збору та обробки інформації: при аналізі результатів досліджень, при проектуванні і прогнозуванні, при оцінці і прийнятті рішень. Часто маючи просте формулювання, задача класифікації виявляється досить складною і неоднозначною. Більш того, іноді при спробах класифікації виникають цікаві парадокси, пов'язані з об'єднанням в один клас принципово різних об'єктів.

Рішення задачі класифікації, як правило, включає значну частку суб'єктивізму, індивідуальних оцінок, нечітких, неформальних висновків. Часто на рішення цієї задачі впливають пріоритети особи, що приймає рішення (ОПР). Це призводить до побудови принципово різних класифікацій на основі однієї і тієї ж первинної інформації.

Особливо часто ця ситуація виникає в тих областях знань, в яких неможливо використовувати числові оцінки при класифікації об'єктів і явищ, в силу чого виникає необхідність нечітких оцінок, використання понять «схожі», «подібні». Аналіз багатовимірних об'єктів враховує велику кількість параметрів, що

потребує застосування спеціальних методів їх обробки. Зокрема при класифікації багатовимірних даних використовуються методи математичної статистики, теорії прийняття рішень тощо.

*Задача класифікації* — це задача розбиття множини об'єктів або спостережень на апріорно задані групи, звані класи, всередині кожної з яких вони передбачаються схожими один на одного, мають приблизно однакові властивості і ознаки. При цьому рішення отримується на основі аналізу значень атрибутів [2].

Існують також наступні визначення класифікації.

Класифікація – системний розподіл досліджуваних предметів, явищ, процесів за родами, видами, типами, з якими-небудь істотними ознаками для зручності їх дослідження; угруповання вихідних понять і розташування їх у певному порядку, що відбиває ступінь цієї схожості [2].

Класифікація – впорядкована за деяким принципом множина об'єктів, які мають подібні класифікаційні ознаки (одна або декілька властивостей), обраних для визначення схожості або відмінності між цими об'єктами [24].

Введемо поняття «класифікації» та опишемо терміни, пов'язані з нею [24].

*Класифікувати об'єкт* — означає, вказати номер (чи назву) класу, до якого відноситься даний об'єкт.

*Класифікація об'єкту* — номер або найменування класу, що видається алгоритмом класифікації в результаті його застосування до даного конкретного об'єкту.

*Класифікація вимагає дотримання наступних правил:*

- на кожному етапі необхідно застосовувати тільки одну основу;
- ділення повинне бути пропорційним, тобто загальний обсяг видових понять повинен дорівнювати об'єму діленого родового поняття;
- члени ділення повинні взаємно виключати один одного, їх об'єми не повинні перехрещуватися;

– ділення повинне бути послідовним.

*Мета процесу класифікації* полягає в тому, щоб побудувати модель, яка використовує прогнозуючі атрибути в якості вхідних параметрів і отримує значення залежного атрибута. Процес класифікації полягає в розбитті множини об'єктів на класи за певним критерієм.

*Класифікатором* називається якась сутність, що визначає, якому з визначених класів належить об'єкт за вектором ознак.

Задачу класифікації в дисертації будемо розглядати як задачу дослідження множини довільної природи [28].

Основні визначення і результати цього розділу складають основу подальших досліджень. У чисто математичній постановці задача класифікації формулюється аксіоматично з використанням понять бінарних відносин і розбиття множини на класи. Весь необхідний математичний апарат можна знайти, наприклад, в [57, 59, 69].

Задача класифікації на множині  $X$  – це задача розбиття множини  $X$  на непересічні класи, тобто, це задача пошуку такого сімейства підмножин  $\{X_\alpha\}$  множини  $X$ , для якої виконані наступні умови відповідно до наступного визначення.

*Визначення 1.1.* Система  $C$  множин  $X_\alpha$  називається розбиттям множини  $X$ , якщо вона задовольняє таким умовам:

$$\begin{aligned}
 & 1. (\forall X_\alpha \in C)[X_\alpha \subseteq X], \\
 & 2. (\forall X_\alpha \in C)[X_\alpha \neq \emptyset], \\
 & 3. (\forall X_\alpha \in C)(\forall X_\beta \in C)[X_\alpha \neq X_\beta \rightarrow X_\alpha \cap X_\beta = \emptyset], \\
 & 4. [X \subseteq \bigcup_\alpha X_\alpha].
 \end{aligned}
 \tag{1.1}$$

при цьому елементи системи  $C$  називають класами розбиття.

З заданим розбиттям  $S$  пов'язано відношення еквівалентності  $\sim$ , при цьому відношення  $\sim$  це єдине відношення еквівалентності для якого елементи  $S$  є класами еквівалентності [69]. Нагадаємо, що відношення еквівалентності – це бінарне відношення з властивостями рефлексивності, симетричності і транзитивності: 1)  $\forall x \in X \quad x \sim x$  2)  $\forall x, y \in X \quad x \sim y \rightarrow y \sim x$  3)  $\forall x, y, z \in X \quad x \sim y, y \sim z \rightarrow x \sim z$ .

Наведемо найпростіші приклади класифікацій.

1. Напрями в багатовимірному просторі  $R^n \setminus \{0\}$  – класи однаково спрямованих векторів. Два ненульових вектори еквівалентні, якщо вони однаково спрямовані;

2. Залишки за натуральним модулем  $n$ . Два цілих числа належать до одного класу в тому і тільки в тому випадку, коли різниця між ними кратна  $n$ ;

3. Компоненти зв'язності в графі. Дві вершини еквівалентні, якщо вони лежать в одній компоненті зв'язності.

Будь-яке відношення еквівалентності на множині  $X$  породжує класифікацію на цій множині. Класи еквівалентності, тобто максимальні по включенню непересічні класи попарно еквівалентних елементів є елементами цієї класифікації. З властивостей відносини еквівалентності випливає, що класи еквівалентності попарно не перетинаються.

Наведемо два алгоритми утворення класів еквівалентності з заданим відношенням еквівалентності на кінцевій множині  $X$  [57].

Перший з них лінійний алгоритм працює наступним чином:

*Крок 0.* Обирається довільне упорядкування (нумерація) елементів множини  $X$ :  $x_1, x_2, \dots, x_n \in X$ . Визначається список представників класів еквівалентності, який на початковому етапі роботи алгоритму порожній.

Сімейство класів еквівалентності також порожньо.

*Крок  $i$ .* Обирається черговий елемент  $x_i$  впорядкованої послідовності

елементів множини  $X$  і послідовно порівнюється зі списком представників вже визначених класів еквівалентності. Якщо елемент  $x_i$  еквівалентний представнику  $x_k$  класу  $X_k$ , то цей елемент  $x_i$  додається в клас  $X_k$ . Якщо елемент  $x_i$  не еквівалентний жодному з елементів списку представників класів, то елемент заноситься в список представників і визначає новий клас еквівалентності.

Алгоритм закінчує роботу, коли всі елементи множини  $X$  будуть рознесені по класах. Результатом роботи алгоритму є список представників сформованих класів і сімейство  $X_k, k = 1, m, m \leq n$  класів еквівалентних елементів.

Наведемо опис ще одного алгоритму формування класів еквівалентності. Це алгоритм вибору представників, що складається з наступних кроків.

*Крок 0.* Як і в лінійному алгоритмі обирається довільне упорядкування (нумерація) елементів множини  $X: x_1, x_2, \dots, x_n \in X$ . Визначається список представників класів еквівалентності, який на початковому етапі роботи алгоритму порожній. Множина класів еквівалентності також порожня.

*Крок 1.* Побудова списку представників. Перевіряються елементи множини  $X$  в обраному порядку. Кожен з елементів порівнюється з елементами списку представників. Якщо елемент  $x_i$  не еквівалентний жодному елементу списку, то цей елемент  $x_i$  заноситься в список представників і видаляється з множини  $X$ . Крок закінчується, коли всі елементи множини перевірені і побудовано список представників.

*Крок 2.* Побудова класів еквівалентності. Послідовно перевіряються елементи списку представників в порядку їх додавання в цей список. Для кожного елемента списку в множині  $X$  виділяються всі елементи, які йому еквівалентні, і з них формується відповідний клас еквівалентності, після чого елементи сформованого класу видаляються з множини  $X$ . Крок закінчується, коли всі елементи множини  $X$  рознесені за класами еквівалентності (множина стала порожньою).

Результат роботи алгоритму вибору представників є таким же, як і лінійного алгоритму. Трудомісткість алгоритму вибору представників більше, ніж у лінійного алгоритму. Тому алгоритм вибору представників використовують зазвичай лише для вибору представників класів еквівалентності (Крок 1), або для побудови класів, якщо представники класів вже задані (Крок 2).

*Вхідні дані для класифікації можуть бути наступних типів.*

1) Характеристичний опис — найпоширеніший випадок. Кожен об'єкт описується набором своїх характеристик, які називаються ознаками. Ознаки можуть бути числовими або нечисловими.

2) Матриця відстаней між об'єктами. Кожен об'єкт описується відстанями до всіх інших об'єктів навчальної вибірки. З цим типом вхідних даних працюють деякі методи, зокрема, метод найближчих сусідів, метод потенційних функцій.

3) Часовий ряд або сигнал є послідовність вимірів у часі. Кожен вимір може представлятися числом, вектором, а в загальному випадку — характеристичним описом досліджуваного об'єкта в цей час часу.

Зустрічаються і складніші випадки, коли вхідні дані представляються у вигляді графів, текстів, результатів запитів до бази даних тощо. Як правило, вони приводяться до першого або другого випадку шляхом попередньої обробки даних та вилучення характеристик.

Класифікацію сигналів та зображень називають також розпізнаванням образів.

*Розглянемо типи класів при проведенні класифікації.*

Двокласова класифікація є найпростішим в технічному відношенні випадком, який служить основою для вирішення складніших завдань.

Багатокласова класифікація має місце, коли число класів досягає багатьох тисяч (наприклад, при розпізнаванні ієрогліфів або злитого мовлення), задача класифікації стає істотно важчою.

Існує досить велика кількість практичних завдань, в яких число класів невідомо і має бути визначено в ході класифікації. В цьому випадку виникає проблема вибору найкращої в якомусь сенсі класифікації. У такій постановці задача класифікації може бути віднесена до групи оптимізаційних задач. Тому для її вирішення необхідно визначити критерії оптимізації і параметри, варіюючи які можна буде з множини можливих класифікацій вибрати одну, найкращу з точки зору критерію.

Розглянемо задачу *оптимальної класифікації* [24].

Грунтуючись на гіпотезі компактності образів, здається природним вважати, що за інших рівних умов класифікація тим краще, чим ближче один до одного точки всередині кожного класу. Разом з тим, класифікація тим краще, чим далі один від одного класи, отримані в результаті класифікації. З точки зору геометричного підходу, прийнятого в теорії розпізнавання образів, для визначення близькості точок можна використовувати відстані між ними або їх потенціал. Тому в якості міри близькості точок всередині класу можна прийняти «власний потенціал» класу

$$\Phi(A, A) = \frac{2}{N_A(N_A - 1)} \sum_{j, p, p > j} \Phi(x_j, x_p), \quad (1.2)$$

де  $N_A$  – число об'єктів в класі, при цьому підсумовування відбувається так, що  $j$  приймає всі значення від 1 до  $n$ , а  $p$  – для кожного  $j$  все значення, більші, ніж  $j$ ;  $N_A(N_A - 1) / 2$  – (число сполучень з  $N_A$  по 2) – кількість членів суми.

Іншими словами,  $\Phi(A, A)$  пропорційний сумі потенціалів, взаємно створюваних один на одному всіма точками класу. Очевидно, що  $\Phi(A, A)$  тим більше, чим тісніше між собою розташовані точки всередині класу, ніж компактніше клас.

Введемо величину «середнього власного потенціалу» даної класифікації:

$$F_1 = \frac{1}{k} \sum_{i=1}^k \Phi(A_i, A_i), \quad (1.3)$$

де  $k$  – число класів у класифікації.

Якщо порівнювати між собою кілька різних класифікацій, то величина  $F_1$  буде максимальною у тій з них, в якій точки кожного класу в середньому лежать найбільш тісно. Величину  $F_1$  можна прийняти за перший критерій оптимізації, а її максимальне значення буде відповідати класифікації, оптимальної з точки зору цього критерію.

Другим критерієм оптимізації повинна бути величина, що характеризує близькість класів між собою. Як цю величину приймемо

$$F_2 = \frac{2}{k(k-1)} \sum_{i,j,j>i}^k \Phi(A_i, A_j), \quad (1.4)$$

де  $k$  – число класів у класифікації,

$\Phi(A_i, A_j)$  – межа близькості між класами  $A_i$  і  $A_j$ .

Підсумовування проводиться так, що  $i$  приймає всі значення від 1 до  $k$ , а значення  $j$  вибираються для кожного  $i$  вибираються так, щоб вони були більше  $i$ . Якщо порівнювати між собою кілька різних класифікацій, то величина  $F_2$ , очевидно, буде мінімальна у тій з них, де класи в середньому розташовані далі один від одного. Величину  $F_2$  можна прийняти за другий критерій оптимізації, а її мінімальне значення буде відповідати класифікації, оптимальної з точки зору цього критерію.

При знаходженні найкращої класифікації на основі аналізу множини альтернативних варіантів необхідно прийняти рішення про вибір оптимального варіанту, тобто вирішити завдання вибору переваг між певною множиною альтернативних класифікацій. Варіант класифікації, який обрали за одним критерієм, може виявитися неприйнятним з точки зору іншого, не менш важливого критерію. Для досягнення потрібного ефекту прийняття остаточного рішення пропонується здійснити на основі порівняння класифікацій за значеннями їх обох критеріїв.

Така постановка задачі пошуку оптимальної класифікації дозволяє віднести цю задачу до багатокритеріальних.

Майже всі математичні методи оптимізації призначено для відшукування оптимального рішення однієї функції – одного критерію. Тому одним з варіантів вирішення багатокритеріальних завдань є її приведення до однокритеріальною з одним узагальненим критерієм.

При знаходженні найкращої класифікації шуканий узагальнений критерій повинен бути таким, щоб найкраща класифікація відповідала відносно великому  $F_1$  і, одночасно, щодо малому  $F_2$ . В якості такого критерію може бути прийнятий, наприклад,

$$F = \alpha F_1 - (1 - \alpha) F_2 . \quad (1.5)$$

Параметр  $\alpha \in [0, 1]$  задається апіорі, або вибирається з деяких зовнішніх міркувань.

Та класифікація з декількох альтернативних класифікацій однієї і тієї ж сукупності об'єктів, для якої критерій  $F$  приймає максимальне значення, очевидно, є в певному сенсі *об'єктивно оптимальною класифікацією*.

Розглянемо задачу оптимальної класифікації. Нехай задана кінцева дискретна множина  $A$ , яке має бути розбита на кластери. Потужність множини  $A$  дорівнює  $n$ . Нехай  $AN$  – множина всіх можливих розбиттів множини  $A$ . На множині  $AN$  введена метрика

$$\rho_{AN}(x, y) = \alpha \max \min \rho(\zeta, \eta) + (1 - \alpha) \max \min \rho(\zeta, \eta), \quad (1.6)$$

де  $x, y \in AN, x = (\zeta_1, \dots, \zeta_k), y = (\eta_1, \dots, \eta_s), \rho(\zeta, \eta)$  – відстань між вибірками  $\eta, \zeta \in A$ .

Нехай для елементів множини  $A$  існує деяка множина  $P = \{p_1, p_2, \dots, p_n\}$  спостережуваних параметрів. Результат вимірювання ознаки  $p_i$  елемента  $a_j$  множини  $A$  позначимо через  $b_{ij}$ . Тоді кожному елементу  $a_j$  множини  $A$  відповідає вектор значень ознак  $B_j = (b_{1j}, \dots, b_{ij}, \dots, b_{nj})$ .

Всій множині  $A$  відповідає множина векторів вимірювань  $B = \{B_1, \dots, B_n\}$ .

Нехай  $m \ll n$ . Задача кластерного аналізу полягає в тому, щоб на основі даних множин  $B$ , розбити множину об'єктів  $A$  на  $m$  кластерів  $A_1, \dots, A_m$  так, щоб кожен об'єкт належав одному і тільки одного кластеру, і щоб об'єкти, які належать одному кластеру були схожими, а ті, які належать різним кластерам були б несхожими при формалізованих поняттях схожості і несхожості.

Більш складною задачею кластерного аналізу є розбиття на кластери множини  $A$  без обмежень на кількість класів  $m$ . Кількість класів  $m$  визначається в процесі виконання завдання.

Рішенням наведених завдань є розбиття, яке задовольняє умов внутрішнього однорідності кластерів і доставляє оптимум цільової функції (критерію).

## 1.2 Поняття відстані в задачах оптимальної класифікації

Для пошуку розв'язку задачі класифікації використовуються різні методи вимірювання відстані, зокрема, відстань Геммінга, Манхеттенська метрика,

відстань Мінковського, Евклідова відстань.

Евклідова відстань є доцільною для класифікації елементів множини, яка включає ознаки одного типу. Для репрезентації ознак різних типів рекомендується використовувати Манхеттенську метрику [1].

Для задач класифікації вихід також може бути представлений як набір імовірностей елементу, що належить класу. Наприклад, ймовірність може бути розрахована як

$$P(O) = \frac{N_0}{N_0 + N_1}, \quad (1.7)$$

де  $P(O)$  – це ймовірність членства класу  $O$ ,

$N_0, N_1$  – числа сусідів, що відносяться до класів  $O$  і  $I$ , відповідно [27].

Значення  $k$  відіграє вирішальну роль у точності прогнозування алгоритму. Проте вибір значення  $k$  є нетривіальною задачею. Менші значення  $k$  швидше за все призведуть до зниження точності, особливо в наборах даних із великим шумом, оскільки кожен примірник набору тренувань тепер має більшу вагу в процесі прийняття рішення. Більші величини  $k$  знижують ефективність алгоритму. На додаток до цього, якщо значення занадто високе, модель може перенавчитись, що зробить межі класу менш чіткими і знову призводить до зниження точності. Як загальний підхід рекомендується обрати  $k$ , використовуючи формулу нижче:

$$k = \sqrt{n}. \quad (1.12)$$

Для класифікації завдань з рівною кількістю класів рекомендується вибрати непарну  $k$ , оскільки це призведе до виключення можливості встановлення нічиєї під час підрахунку більшості голосів. Недоліком цього алгоритму є погана

ефективність на нерівномірно розподілених наборах даних. Отже, якщо один клас значно домінує над іншими, то він, швидше за все, матиме більше сусідів цього класу через їх велику кількість.

### 1.3 Обчислювальна складність. Складність задачі класифікації

Різні алгоритми мають різну часову складність, і виявлення того, які алгоритми «досить ефективні», а які «абсолютно неефективні», завжди буде залежати від конкретної ситуації. Однак теоретики, які займаються розробкою і аналізом алгоритмів, пропонують для порівняння ефективності алгоритмів один простий підхід, що дозволяє істотно прояснити ситуацію. Йдеться про різницю між поліноміальними і експонентними алгоритмами [19, 26, 56].

*Поліноміальним алгоритмом або алгоритмом поліноміальної складності* називається алгоритм, у якого часова складність дорівнює  $O(p(n))$ , де  $p(n)$  – деяка поліноміальна функція, а  $n$  – вхідна довжина.

Алгоритми, часова складність яких не піддається подібній оцінці, називаються *експонентними*. Слід зазначити, що це визначення включає і такі функції, як  $n^{\log n}$ , хоча вони і не є поліноміальними, але зазвичай не вважаються експонентними. Більшість експоненційних алгоритмів – це просто варіанти повного перебору, в той час як поліноміальні алгоритми зазвичай можна побудувати лише тоді, коли вдається більш глибоко проникнути в суть розв'язуваної задачі.

Ця точка зору, що розрізняє, з одного боку, поліноміальні алгоритми, а з іншого боку, експоненціальні, є відправним пунктом у визначенні важкорозв'язуваних задач і теорії NP-повних задач.

*NP-повна задача* – в теорії алгоритмів та теорії складності це задача, що належить до класу  $NP$  та всі задачі з класу  $NP$  можна звести до неї за поліноміальний час [19].

*Клас задач NP-повноти*, тобто  $NP$ -повних задач – повних для недетермінованого поліноміального часу – це клас завдань, ні для однієї з яких не вдалося знайти алгоритму поліноміальної складності [19].

У цьому класі виділяється підклас  $P$  задач, які можуть бути вирішені за час, обмежений поліномом, що залежить від довжини умови задачі. Очевидно  $P \subseteq NP$ . Центральним питанням сучасної теорії складності є питання про те чи співпадають класи  $P$  і  $NP$  [19].

У класі  $NP$  виявлені універсальні  $NP$ -повні задачі, до яких зводиться за поліноміальний час будь-яка задача класу  $NP$ . У цьому сенсі  $NP$ -повні задачі дають еталон складності.

Задачу будемо називати *важкорозв'язуваною (складною в обчислювальному сенсі)*, якщо до неї поліноміально зводиться хоча б одна  $NP$ -повна задача. У такій постановці більшість з задач оптимальної класифікації є складними в обчислювальному сенсі. Огляд  $NP$ -повних задач можна знайти в [19, 22, 23, 26, 64].

Для таких задач на сьогодні невідомі алгоритми пошуку точного розв'язку простіших, ніж повний перебір всіх допустимих розв'язків задачі.

Тому є сенс шукати прості наближені алгоритми, які хоч і не дають точного розв'язку, але мають високу швидкодію. Серед таких алгоритмів виділяється клас «жадібних» алгоритмів.

## 1.4 Методи пошуку розв'язків задач оптимальної класифікації

Оскільки задача класифікації є у певному сенсі частковим випадком задачі кластеризації, розглянемо метод визначення числа кластерів (класів), який базується на використанні *нейронної мережі Кохонена*, критерію якості отриманих розбиттів та методі ідеальної точки.

Даний метод використовує два критерії. Перший припускає, що вірне розбиття на кластери – це розбиття з найбільшою частотою, отримане нейронною мережею Кохонена.

Згідно першому критерію проводиться серія запусків нейронної мережі та формується множина розбиттів, кількість яких дорівнює числу спроб. За розбиттями будується матриця, де рядкам відповідає множина об'єктів  $P$ , а стовпчикам – кластери розбиттів  $Q_l, l = \overline{1, k}$ . Елемент матриці обчислюється наступним чином:

$$a_{ij} = \begin{cases} 1, p_i \in C_t^l \\ -1, p_i \notin C_t^l \end{cases}, \quad (1.13)$$

де  $t = \overline{1, m}, l = \overline{1, k}, C_t^l \subset Q_l, Q_l \subset \prod_{l=1, k} Q_l$ .

Кожній матриці ставиться у відповідність граф. Для кожного ребра графу вказується оцінка  $c(e)$ , як сила зв'язку  $i$ -го вузла з  $j$ -м, яка обчислюється за формулою:

$$c(e) = \left( \sum_{q \in Q_l, l=1, k} w_q \right) / k, \quad (1.14)$$

де  $w_q = \begin{cases} 1, \exists C_t^l \subset Q_l : p_i \in C_t^l \ \& \ p_j \in C_t^l, t = \overline{1, m} \\ -1, \text{у протилежному випадку} \end{cases}$

Наступним кроком є вибір оптимального графу, в якому сумарна оцінка ребер – максимальна. Цей граф відповідає розбиттю з найбільшою частотою. Якщо сила зв'язку між  $i$ -м та  $j$ -м об'єктами більше 0 – їх відносять до одного кластеру, інакше – до різних.

Недоліком цього критерію є розбиття вибірки об'єктів на якомога менше число кластерів, тому для оцінки одержаних розбиттів використовується другий критерій, що задається як оптимізація цільової функції:

$$F = -\sum_{i \in M} f_i + g \rightarrow \max, \quad (1.15)$$

де  $f_i$  – сума відстаней точок кластеру від центру кластеру по всіх кластерах,  $i = \overline{1, M}$ ,

$M$  – кількість кластерів розбиття,

$g$  – сумарна відстань між центрами кластерів.

Даний критерій дає змогу знаходити розв'язки серед розбиттів з більшою потужністю.

Для зважування результатів обох критеріїв використовується *метод ідеальної точки*, яка визначається як точка з двома координатами – найкращими значеннями обох критеріїв. Правило вибору полягає у знаходженні альтернативи, що має оцінку найближчу до ідеальної точки.

Для побудови вектору ознак об'єкту, розподілених по різномірним групам, запропоновано метод, що базується на використанні нейронної мережі прямого розповсюдження. Метод побудови вектору ознак полягає у реалізації двох етапів: пошуку оптимальної вибірки, якій відповідає деяке число  $\Theta$ , та побудови вектору ознак об'єкту за числом  $\Theta$ . Дані етапи реалізовані відповідно в алгоритмах *A1* та *A2*.

Нова вибірка формується із векторів з середніми значеннями компонент векторів поточних вибірок; відповідна їй точка з відрізка – середнє значення точок, що відповідають поточним вибіркам. Результатом роботи алгоритму  $A1$  є знайдена точка  $\Theta$ , що відповідає оптимальній вибірці. Побудова вектору ознак за числом  $\Theta$  реалізована в алгоритмі  $A2$ .

Для простору ознак, розподілених більш ніж на дві групи, розв'язок задачі зводиться до реалізації алгоритмів  $A1$  та  $A2$ .

Виділивши кількісні ознаки, що характеризують заданий корпус об'єктів та, використовуючи даний метод, можна досліджувати широкий клас задач класифікації.

Наведено оцінки складності алгоритму пошуку оптимальної вибірки. Під обчислювальною складністю алгоритму пошуку оптимальної вибірки розуміється верхня границя для максимального числа основних операцій, які необхідно виконати алгоритму для знаходження оптимальної вибірки.

**Лема 1.1.** Нехай  $n$  – розмірність вектору ознак об'єкту одноелементної вибірки;  $\varepsilon$  – задана точність алгоритму  $A1$  (довжина відрізка, одна з крайніх точок якого відповідає оптимальній вибірці);  $W$  – кількість вагових зв'язків між нейронами, включаючи зсуви внутрішніх та вихідних нейронів. Тоді оцінка складності алгоритму  $A1$  з використанням одноелементної вибірки складає:

$$N \log_2 \frac{1}{\varepsilon} \cdot O(W).$$

Для випадку, коли навчальні вибірки представляють множину об'єктів, має місце **Теорема 1.1.** Нехай  $N$  – розмірність вектору ознак для об'єктів навчальної вибірки, вихідний опис яких задається групами ознак;  $P$  – кількість об'єктів (прикладів) у вибірці;  $\varepsilon$  – задана точність алгоритму  $A1$ ;  $W$  – кількість вагових зв'язків між нейронами, включаючи зсуви внутрішніх та вихідних нейронів. Тоді для алгоритму  $A1$  справедлива оцінка:  $N \log_2 \frac{1}{\varepsilon} \cdot O(WP)$ .

Позначимо через *приклад виду*  $T$  – приклад, для якого вектор по одній з груп ознак не містить значень, а по іншій групі ознак його розмірність дорівнює  $N$  (довжині вектору – прикладу). Тоді для випадку наявності у вибірці об'єктів з нерівномірним розподілом груп ознак мають місце:

Наслідок 1.1. Нехай  $N$  – розмірність вектору ознак для об'єктів навчальної вибірки, що включає ознаки всіх груп, які виділені при класифікації.

Якщо навчальна вибірка із  $P$  прикладів містить приклад виду  $T$ , то оцінка складності алгоритму  $AI$  дорівнює:  $N \log_2 \frac{1}{\varepsilon} \cdot O(WP)$ .

Наслідок 1.2. Якщо у вибірці з  $P$  прикладів є приклад виду  $T$  по групі  $\alpha$  та приклад виду  $T$  по групі  $\beta$ , то оцінка складності алгоритму  $AI$  складає:  $2N \log_2 \frac{1}{\varepsilon} \cdot O(WP)$ .

Наслідок 1.3. Якщо у вибірці з  $P$  прикладів є приклад виду  $T$  та приклад, для якого одна з груп ознак обмежена по довжині значенням  $C$ , то оцінка складності для алгоритму  $AI$  складає:  $(N + C) \log_2 \frac{1}{\varepsilon} \cdot O(WP)$ .

Для розв'язку задачі класифікації найчастіше застосовуються *методи*: найближчого сусіда (Nearest Neighbor),  $K$ -найближчого сусіда ( $k$ -Nearest Neighbor) [1]; байєсовські мережі (Bayesian Networks); індукція дерев рішень [19]; нейронні мережі (neural networks), метод опорних векторів [52, 58]; статистичні методи, зокрема, лінійна регресія; класифікація *sbr* – методом або за допомогою генетичних алгоритмів, що розглянуто в [58].

*Дерева рішень (decision trees)* є однією з найбільш популярних структур для розв'язку задачі класифікації. Дерева рішень дозволяють візуально і аналітично оцінити результати вибору різних рішень і використовується в галузі статистики та аналізу даних для прогнозних моделей. Вперше були запропоновані Ховілендом і Хантом (Hoveland, Hunt) наприкінці 50-х років минулого століття.

Дерева рішень використовують, коли потрібно прийняти рішення в умовах невизначеності, коли кожне рішення залежить від результату попередніх результатів або деяких заданих умов, що з'являються з певною ймовірністю.

Дерева рішень часто називаються деревами вирішальних правил, деревами класифікації і регресії. Якщо залежна (цільова) змінна приймає дискретні значення – це задача класифікації, якщо залежна змінна приймає безперервні значення, то вирішується задача чисельного прогнозування.

Дерево рішень, подібно його «прототипу» з живої природи, складається з гілок з атрибутами (від них залежить результат – цільова функція), листів зі значеннями цільової функції (вирішальні вершини – результат вибору певного значення атрибута), а також вузлів – випадкових вершин, в яких визначаються можливі варіанти розвитку подій з певного моменту. «Зростає» дерево до тих пір, поки альтернативні варіанти не почнуть повторюватися.

Метою процесу побудови дерева прийняття рішень є створення моделі, за якою можна було б класифікувати випадки і вирішувати, які значення може приймати цільова функція, маючи на вході кілька змінних.

*Розглянемо основні алгоритми для побудови дерева рішень.*

*ID3.* В основі цього алгоритму лежить поняття інформаційної ентропії – тобто, міри невизначеності інформації (зворотній до міри інформаційної корисності величини). Для того щоб визначити наступний атрибут, необхідно підрахувати ентропію всіх невикористаних ознак щодо тестових зразків і вибрати той, для якого ентропія мінімальна. Цей атрибут і вважатиметься найбільш доцільною ознакою класифікації.

*C5.* Цей алгоритм – удосконалення попереднього методу, що дозволяє, зокрема, «усікати» гілки дерева, якщо воно занадто сильно «розростається», а також працювати не тільки з атрибутами-категоріями, а й з числовими. Алгоритм виконується за тим же принципом, що і його попередник; відмінність полягає в

можливості розбиття області значень незалежної числової змінної на кілька інтервалів, кожен з яких буде атрибутом. Відповідно до цього вихідна множина ділиться на підмножини. Зрештою, якщо дерево виходить занадто великим, можливе зворотне угруповання – кількох вузлів в один лист. При цьому, оскільки перед побудовою дерева помилка класифікації вже врахована, вона не збільшується.

*CART*. Алгоритм розроблений з метою побудови так званих бінарних дерев рішень – тобто тих дерев, кожен вузол яких при розбитті «дає» тільки двох нащадків. Грубо кажучи, алгоритм діє шляхом поділу на кожному кроці множини прикладів рівно навпіл – по одній гілці йдуть ті приклади, в яких правило виконується (правий нащадок), за іншою – ті, в яких правило не виконується (лівий нащадок). Таким чином, в процесі «зростання» на кожному вузлі дерева алгоритм проводить перебір всіх атрибутів, і для наступного розбиття вибирає той, який максимізує значення показника, обчислюваного за математичною формулою і залежного від відносин числа прикладів у правому та лівому нащадків до загального числа прикладів.

*Алгоритми побудови дерев рішень розрізняються наступними характеристиками:*

1. Вид розгалуження – бінарне (binary), множинне (multi-way).
2. Критерії розгалуження – ентропія, Gini, інші.
3. Можливість обробки пропущених значень.
4. Процедура скорочення гілок або відсікання.
5. Можливості витягування правил з дерев.

Жоден алгоритм побудови дерева не можна апріорі вважати найкращим або досконалим, підтвердження доцільності використання конкретного алгоритму має бути перевірено і підтверджено експериментом.

Найбільш серйозна вимога, яке зараз пред'являється до алгоритмів конструювання дерев рішень – це масштабованість до різних розмірів вибірок даних.

Дерева прийнять рішень – один з основних і найбільш популярних методів допомоги у прийнятті рішень. Побудова дерев рішень дозволяє наочно продемонструвати іншим і розібратися в структурі даних, створити працюючу модель класифікації даних, якими б «великими» вони не були.

*Переваги дерев рішень:*

1. Інтуїтивність дерев рішень. Класифікаційна модель, представлена у вигляді дерева рішень, є інтуїтивною і спрощує розуміння розв'язуваної задачі. Результат роботи алгоритмів конструювання дерев рішень, на відміну, наприклад, від нейронних мереж, що представляють собою "чорні ящики", легко інтерпретується користувачем. Ця властивість дерев рішень не стільки важлива при віднесенні нового об'єкта до певного класу, але й корисна при інтерпретації моделі класифікації в цілому. Дерево рішень дозволяє зрозуміти і пояснити, чому конкретний об'єкт відноситься до того чи іншого класу.

2. Дерева рішень дають можливість формувати правила з бази даних природною мовою. Приклад правила: Якщо Вік $>$ 35 і Дохід $>$ 2000, то видати кредит.

3. Дерева рішень дозволяють створювати класифікаційні моделі в тих областях, де аналітику досить складно формалізувати знання.

4. Алгоритм конструювання дерева рішень не вимагає від користувача вибору вхідних атрибутів (незалежних змінних). На вхід алгоритму можна подавати всі існуючі атрибути, алгоритм сам вибере найбільш значущі серед них, і тільки вони будуть використані для побудови дерева. У порівнянні, наприклад, з нейронними мережами, це значно полегшує користувачеві роботу, оскільки в нейронних мережах вибір кількості вхідних атрибутів істотно впливає на час навчання.

5. Точність моделей, створених за допомогою дерев рішень, порівнянна з іншими методами побудови класифікаційних моделей (статистичні методи, нейронні мережі).

6. Розроблено ряд масштабованих алгоритмів, які можуть бути використані для побудови дерев рішення на надвеликих базах даних; масштабованість тут означає, що із зростанням числа прикладів або записів бази даних час, що витрачається на навчання, тобто побудова дерев рішень, зростає лінійно. Приклади таких алгоритмів: SLIQ, SPRINT.

7. Швидкий процес навчання. На побудову класифікаційних моделей за допомогою алгоритмів конструювання дерев рішень потрібно значно менше часу, ніж, наприклад, на навчання нейронних мереж.

8. Більшість алгоритмів конструювання дерев рішень мають можливість спеціальної обробки пропущених значень.

Багато класичних статистичних методів, за допомогою яких вирішується задача класифікації, можуть працювати тільки з числовими даними, в той час як дерева рішень працюють і з числовими, і з категоріальними типами даних.

Багато статистичних методів є параметричними, і користувач повинен заздалегідь мати певну інформацію, наприклад, знати вид моделі, мати гіпотезу про вид залежності між змінними, припускати, який вид розподілу мають дані. Деревя рішень, на відміну від таких методів, будують непараметричні моделі. Таким чином, дерева рішень здатні вирішувати завдання, в яких відсутня апріорна інформація про вид залежності між досліджуваними даними.

Потужним класом методів класифікації є *методи дискримінації*, які базуються на навчанні. Широке застосування серед методів даної групи здобули ядерні методи машинного навчання. Першою концепцією ядерного підходу став метод опорних векторів, запропонований В. Вапніком у 1992 р. Він одержав широке застосування при розв'язку цілого ряду задач аналізу зображень,

відновлення регресії, ідентифікації та ін. Подальшим удосконаленням методу опорних векторів, зокрема, для роботи з даними без лінійно-роздільної здатності, стала реалізація побудови вирішального правила у класі нелінійних поверхонь, запропонована В. Вапніком та С. Бургесом у 1995 р. Активний розвиток серед методів дискримінації отримав Байєсовський підхід, розроблений для оцінювання параметрів моделей при класифікації об'єктів, заданих набором атрибутів. Модифікацією Байєсовського підходу стали байєсовські мережі, які дозволили вирішити проблеми, пов'язані з неправомірністю припущення про умовну незалежність атрибутів об'єктів при використанні правила Байєса.

Альтернативою методам дискримінації є методи, що базуються на обчисленні відстаней: *метод k-найближчих сусідів*, *метод міркування за аналогією* (Case Based Reasoning, CBR), *класифікатор Роше*. Вони не потребують фази навчання та відносяться до класу методів, робота яких базується на збереженні даних у пам'яті для порівняння з новими елементами.

*Непараметричні алгоритми розпізнавання за мірою схожості з еталонними класами* діють за наступною схемою.

1. Обчислюємо спеціальну функцію – міру схожості  $d(X, \Pi_j)$  між об'єктом  $X$  і класом  $\Pi_j$ . Залежно від виду функції схожості можна виділити різні методи.

Розглянемо такі функції схожості:

*функція міри близькості*

$$d_{\text{ФМБ}}(X, P_j) = \left[ \prod_{X_i \in P_j} d(X, X_i) \right]^{1/n_j}; \quad j = 1, 2, \dots, K.$$

*відстань до центра*  $d_c(X, \Pi_i) = d_E(X, \mu_i)$ , де  $d_E(X, \mu_i)$  – евклідова відстань,  $\mu_i$  – вектор середніх класу  $\Pi_i$ ;

*потенціальна функція*

$$d_{\Pi}(X, \Pi_i) = \frac{1}{n_i} \sum_{X_j \in \Pi_i} d_{\Pi}(X, X_j), \quad \text{де } d_{\Pi}(X, X_j) = [1 + \alpha d_E^2(X, X_j)]^{-1}, \quad \alpha > 0.$$

*відстань Махаланобіса*  $d_M(X, \Pi_i) = (X - \mu_i)^T \Sigma_i^{-1} (X - \mu_i)$ , де  $\mu_i$  і  $\Sigma_i^{-1}$  відповідно вектор середніх і коваріаційна матриця класу  $\Pi_i$ .

2. Об'єкт  $X$  належить до класу  $\Pi_i$ , якщо  $d(X, \Pi_i) < d(X, \Pi_j), i \neq j, i, j = 1, 2, \dots, K$ , для функцій  $d_{ФМБ}, d_M, d_C$  і  $d(X, P_i) > d(X, P_j), i \neq j, i, j = 1, 2, \dots, K$ , для функції  $d_{\Pi}$ .

### 1.5 Невирішені проблеми задачі класифікації. Типи задач класифікації

Залежно від обраних ознак, їх поєднання і процедури розподілу понять, класифікація може бути:

1) простою – розподіл родового поняття тільки за ознакою і тільки один раз до розкриття всіх видів. Прикладом такої класифікації є дихотомія, при якій членами поділу бувають тільки два поняття, кожне з яких суперечить іншому (тобто дотримується принцип: «А і не А»);

2) складною – застосовується для поділу одного поняття за різними основами і синтезу таких простих ділень в єдине ціле.

*Розрізняють:*

– допоміжну (штучну) класифікацію, яка виробляється за зовнішньою ознакою і служить для надання множині предметів (процесів, явищ) потрібного порядку;

– природну класифікацію, яка виробляється за істотними ознаками, що характеризують внутрішню спільність предметів і явищ. Вона є результатом і важливим засобом наукового дослідження, тому що передбачає і закріплює результати вивчення закономірностей об'єктів, що класифікуються.

Наведемо декілька актуальних прикладів невіршених задач класифікації (табл. 1.1).

Таблиця 1.1 – Невіршени задачі класифікації

№	Назва задачі	Мета класифікації	Опис задачі та її вирішення
1	Спам електронною	Визначити, чи є електронний лист спамом і чи повинен він	Метод, який тут використовується, базується на відносній частоті найбільш поширених слів та розділових знаків в електронних повідомленнях.

№	Назва задачі	Мета класифікації	Опис задачі та її вирішення
	поштою	бути доставлений у папку «Небажана пошта»	Набір із 57 таких слів і розділових знаків попередньо вибирають дослідники.
2	Розпізнавання рукописних цифр	Ідентифікувати зображення однозначних цифр 0 - 9.	Для комп'ютера зображення є матрицею, і кожен піксель на зображенні відповідає одному запису в матриці. Кожен запис є цілим числом, що починається від інтенсивності пікселів від 0 (чорний) до 255 (білий). Отже, необроблені дані можна подавати на комп'ютер безпосередньо без вилучення будь-яких функцій. Матрицю зображень сканували рядом за рядками, а потім розташовували у великий 256-мірний вектор. Це використовується як вхід для підготовки класифікатора.
3	Сегментація зображень	Ідентифікувати супутникові знімки у природних чи техногенних регіонах	Ці зображення мають $512 \times 512$ пікселів і знову ж таки тому, що це зображення в масштабах сірого, ми можемо представити інтенсивність пікселів цифрами від 0 до 255. у цій проблемі необхідне вилучення функції. Стандартний метод вилучення функцій у проблемі обробки зображень - розділити зображення на блоки пікселів або сформувати околиці кожного пікселя.
4	Розпізнавання мовлення	Ідентифікувати голос та слова у смартфоні, охоронних технологіях тощо	Необхідні дані включають вибірку амплітуди голосу в дискретних часових точках (часова послідовність), які можуть бути представлені у формах хвиль. У кожний момент часу обчислюється одна або кілька функцій, таких як частота. Мовний сигнал по суті стає послідовністю векторів частоти.
5	Опис мікроматриці експресії ДНК	Виявити типи захворювань або тканин на основі рівнів експресії генів	Для кожного зразка, взятого з тканини певного типу захворювання, вимірюється рівень експресії дуже великої колекції генів. Вхідні дані проходять через процес очищення даних.

## Висновки до розділу 1

Задача оптимальної класифікації природним чином виникає при створенні систем автоматизації проектування, автоматизованих систем планування ресурсів, при розв'язанні задач логістики, штучного інтелекту, робототехніки тощо. На жаль, задача оптимальної класифікації, як і більшість задач дискретної оптимізації є NP-важкою.

Для розв'язку задачі класифікації найчастіше застосовуються *методи*: найближчого сусіда, K-найближчого сусіда; байєсовські мережі; індукція дерев рішень; нейронні мережі; метод опорних векторів; статистичні методи, зокрема, лінійна регресія; класифікація *svm* – методом або за допомогою генетичних алгоритмів.

На разі залишилося багато прикладних невирішених задач класифікації, зокрема, розпізнавання спаму електронної пошти, зображень та мовлення в якості цифрового паролю, опис мікроматриці ДНК тощо.

## РОЗДІЛ 2. ДИСКРЕТНІ ЗАДАЧІ КЛАСИФІКАЦІЇ. АЛГОРИТМИ НА ОСНОВІ МЕТАЕВРИСТИК

### 2.1 Задача класифікації на дискретній множині

Основними з найбільш актуальних сьогодні завдань інтелектуального аналізу даних є кластеризація та класифікація на дискретній множині.

Кластеризація – об'єднання в групи схожих об'єктів – одне з фундаментальних завдань у галузі Data Mining. Список прикладних галузей, де вона застосовується, є досить широким: сегментація зображень, маркетинг, медицина, аналіз текстів тощо. На сучасному етапі кластеризація часто виступає першим кроком в аналізі даних [11].

Задача класифікації полягає в побудові моделі за деяким принципом на основі множини об'єктів навчальної вибірки, які мають подібні класифікаційні ознаки, та зарахування нових спостережень до одного з наперед заданих класів.

Перед проведенням класифікації на дискретній множині слід звернути увагу на такі питання.

1. Досить часто об'єкти в досліджуваних наборах даних характеризуються великою кількістю ознак, причому деякі ознаки не несуть значної інформації, а лише збільшують розмірність даних. Це ускладнює роботу алгоритмів, виникає потреба у великому обсязі пам'яті та значній кількості машинного часу, що робить деякі методи непридатними для таких наборів даних. Для розв'язання цієї проблеми користувач може сам обирати ознаки, спираючись на знання предмета, власні припущення та спостереження.

2. Якщо ознаки об'єктів вимірюються в різних одиницях або сильно відрізняються за значеннями, доцільно звести їх до єдиного масштабу. Найбільше

вживаною з таких процедур є стандартизація [11]. Нові стандартизовані значення розраховуються за формулою:

$$x'_{ij} = \frac{x_{ij} - \bar{x}_j}{\sigma_j}, i = \overline{1, n}, j = \overline{1, p}, \quad (2.1)$$

$$\text{де } \bar{x}_j = \frac{1}{n} \sum_{i=1}^n x_{ij}, \sigma_j = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_{ij} - \bar{x}_j)^2}.$$

Наведемо структуру алгоритму класифікації на рис. 2.1.

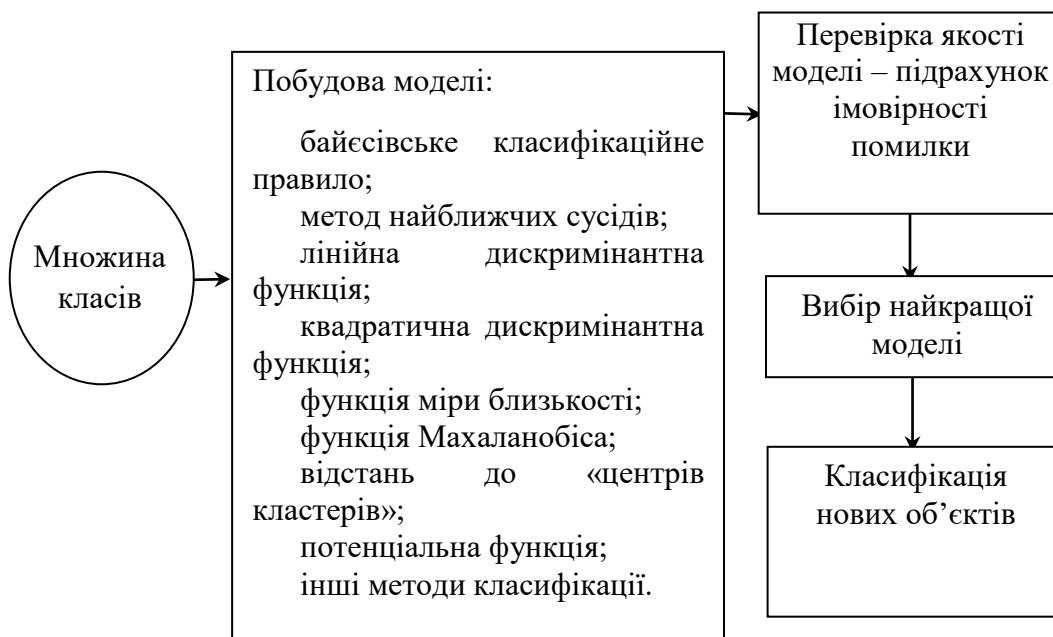


Рис. 2.1. Структура алгоритму класифікації

## 2.2 Зв'язок класифікації з задачею покриття графів

Продемонструємо, що задача класифікації за своєї суттю схожа на задачу покриття графів. Для цього розглянемо графовий адаптивний алгоритм пошуку

кількості та положення центрів кластерів (класів) та розрахункові залежності для визначення оптимальних початкових параметрів роботи алгоритму [4]. Схема роботи алгоритму представлена на рис. 2.2.

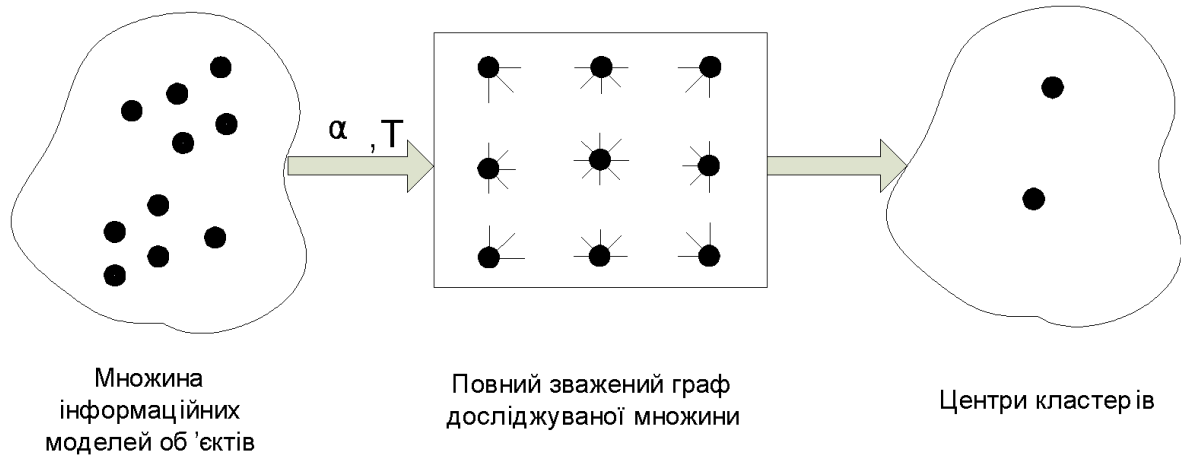


Рис. 2.2. Схема роботи графового адаптивного алгоритму пошуку кількості та положення центрів кластерів

Параметр  $\alpha$  визначає швидкість збіжності алгоритму. Параметр  $T$  задає порогову величину.

Для досліджуваної множини інформаційних моделей об'єктів в  $d$ -мірному просторі обчислюється матриця евклідових відстаней  $L = [l_{nm}]$ , де  $l_{ij} = |X_i - X_j|$ ,  $i = \overline{1, n}, j = \overline{1, m}$ .

Алгоритм працює таким чином, що вузли графу, які знаходяться на межі кластерних областей, отримують недодатню активність і не враховуються в подальших ітераціях. Ітераційний процес збігається до такого результату, коли в кожній області кластера залишається лише один активний інформаційний образ – центр кластера. Проте можливий такий варіант роботи алгоритму, коли виникають згустки центрів кластерів (кілька центрів кластерів в одній області). При цьому число вузлів з позитивною активністю не змінюється. Для таких випадків

розроблено блок адаптації для виявлення некоректних ситуацій, коректування параметрів алгоритму і отримання оптимального результату.

Робота алгоритму полягає у виконанні таких кроків:

Крок 1. Визначити початкову кількість активних вузлів графа

$$n = |D|, r = n, \quad (2.2)$$

де  $|D|$  – кількість елементів множини  $D$ .

Крок 2. Визначити максимальну відстань між точками вузлів графа

$$l_{\max} = \max \{l_{ij}\}, i, j = \overline{1, n}. \quad (2.3)$$

Крок 3. Визначити початкові параметри алгоритму

$$\alpha = 0,5, T = \frac{0,1l_{\max}}{2}, \Delta T = 0,1T, q_{\max} = m, q = 0, \quad (2.4)$$

де  $m$  – наперед задане натуральне число.

Крок 4. Визначити вагові коефіцієнти

$$w_{ij} = \begin{cases} \frac{T^2}{l_{ij} + T^2}, & l_{ij} \leq T \\ 0, & l_{ij} \geq T \end{cases}, \quad (2.5)$$

де  $i, j = \overline{1, n}$ .

Крок 5. Визначити початкові активності вузлів графа

$$s_i(0) = \sum_{j=1}^n w_{ij}, i, j = \overline{1, n}. \quad (2.6)$$

Крок 6. Встановити індекс ітерації  $k = 0$ .

Крок 7. Виконати розрахунок активностей вузлів графа

$$S_i(k+1) = S_i(k) + \alpha \cdot \sum_{j=1}^r w_{ij} \cdot (S_i(k) - S_j(k)), i = \overline{1, r}. \quad (2.7)$$

Крок 8. Визначити кількість вузлів графа, для котрих виконується умова  $s_i(k+1) \leq 0, i = \overline{1, r}$ , і присвоїти змінній  $p$ .

Крок 9. Зменшити кількість активних вузлів графа

$$r_1 = r - p, \quad (2.8)$$

впорядкувати множину індексів активних вузлів графа, враховуючи індекси видалених вузлів.

Крок 10. Якщо  $r = r_1$ , тоді перехід на крок 11, інакше  $r = r_1, k = k + 1$  і перехід на крок 7.

Крок 11. Обчислити відстані між точками активних вузлів графа

$$l_{ij} = |X_i - X_j|, i, j = \overline{1, n}. \quad (2.9)$$

Крок 12. Обчислити мінімальну відстань між точками активних вузлів графа

$$l_{\min} = \min\{l_{ij} \mid i, j = \overline{1, r}\}. \quad (2.10)$$

Крок 13. Якщо  $l_{\min} < 0,5l_{\max}, q \leq q_{\max}$ , тоді  $T = T + \Delta T, r = n, q = q + 1$  і перехід на крок 4, інакше – ЗУПИНКА.

Параметр  $\alpha \in (0;1)$ . Експериментальним шляхом встановлено, що найоптимальнішим є значення  $\alpha = 0,5$ , а найоптимальніше значення  $T$  становить

$$T = \frac{0,1 \max\{l_{ij}\}}{2}, i, j = \overline{1, n}.$$

### 2.3. Підходи до пошуку оптимальних рішень на основі метаевристик

Задача оптимальної класифікації на дискретних множинах мають велику кількість переваг. Зокрема, вони адекватно відображають нелінійні залежності, неподільність об'єктів, враховують логічні та технологічні обмеження, а також «якісні» вимоги. Але у прикладних задачах, як правило, є багато обмежень, які ускладнюють застосовність відомих алгоритмів. Тому для таких задач є виправданим застосування метаевристик.

Особливістю застосування подібних алгоритмів є те, що отримати будь-які гарантовані результати з їх допомогою не представляється можливим. Однак приклад живої природи показує, що подібні моделі містять у собі певну «родзинку» – властивості, які дозволяють отримувати дуже хороші результати при порівняно невеликих обчислювальних витратах. Саме тому інтерес до метаевристичних алгоритмів моделювання не слабшає і в найближчі роки варто очікувати лише активізації досліджень у цьому напрямку.

Перевагами застосування метаевристичних алгоритмів є:

- відносна простота побудови моделі. Фактично, для того щоб побудувати математичну модель оптимізаційної задачі достатньо уявити простір допустимих рішень в якому-небудь стандартному кодуванні;

- універсальність моделі. Такий алгоритм може застосовуватися практично для будь-якої оптимізаційної задачі;
- невелика кількість складових елементів дозволяє легко створювати програмні продукти;
- можливість розв'язання задач великої розмірності;
- можливість вести пошук глобального оптимуму на всьому просторі допустимих розв'язків.

Виділяють дев'ять властивостей метаевристик [39]:

1. Метаевристика є стратегією, яка управляє процесом пошуку.
2. Метою метаевристики є ефективне дослідження простору пошуку для знаходження (суб)оптимального рішення.
3. Методи, що використовуються метаевристичним алгоритмом, знаходяться в діапазоні від простого локального пошуку до складного процесу навчання.
4. Метаевристичний алгоритм є наближеним і зазвичай недетермінованим.
5. Метаевристика може використовувати механізм, що запобігає потраплянню в пастку в обмеженій області простору пошуку.
6. Основні положення метаевристики допускають абстрактний опис.
7. Метаевристика не є проблемно-орієнтованою.
8. Метаевристика може використовувати проблемно орієнтоване знання в формі евристик, керованих високорівневою стратегією.
9. Передові метаевристики використовують досвід, накопичений в процесі пошуку і представлений у вигляді пам'яті, для управління пошуком.

Множина локальних оптимумів може виявитися експоненціально великою, і на перший погляд здається, що такий варіант алгоритму не матиме переваг. Однак експериментальні дослідження розподілу локальних оптимумів свідчать про високу концентрацію їх в безпосередній близькості від глобального оптимуму [97 – 100]. Це спостереження відоме як гіпотеза про існування «великої долини» для

задач на мінімум або «центрального гірського масиву» для задач на максимум.

**Гіпотеза 2.1.** В середньому локальні оптимуми розташовані набагато ближче до глобального оптимуму ніж до випадково обраної точки. Їх розподіл в області допустимих рішень не є рівномірним. Вони концентруються в районі глобального оптимуму, займаючи область невеликого діаметру.

Ця гіпотеза частково пояснює працездатність метаевристичних алгоритмів. Якщо в популяції збираються локальні оптимуми, які відповідно до гіпотези сконцентровані в одному місці, і чергове рішення обирається десь між двома довільними локальними оптимуму, то такий процес має багато шансів знайти глобальний оптимум. У зв'язку з цим перевірка і теоретичне обґрунтування цієї гіпотези за допомогою алгоритмів метаевристики представляє безперечний інтерес. Також постає питання як зменшувати область пошуку.

## 2.4 Опуклі множини в метричних просторах

Одним зі способів зменшення області пошуку екстремуму є використання опуклих множин.

Нехай  $X$  – деяка непорожня множина точок. Функція  $\rho$ , яка ставить у відповідність кожній парі точок  $x, y$  множини  $X$  невід'ємне число  $\rho(x, y)$ , називається метрикою на  $X$ , якщо для довільних  $\{x, y, z\} \subseteq X$  виконуються умови:

1.  $\rho(x, y) = \rho(y, x)$ ;
2.  $\rho(x, y) = 0 \Leftrightarrow x = y$ ;
3.  $\rho(x, y) \leq \rho(x, z) + \rho(z, y)$  (нерівність трикутника).

**Означення 2.1.** Пара  $(X, \rho)$  називається метричним простором.

Зрозуміло, що на одній і тій же множині  $X$  можна задати різні метрики, утворивши різні метричні простори. Якщо на  $X$  задано фіксовану метрику  $\rho$ , то

простір  $(X, \rho)$  позначатимемо просто літерою  $X$ .

**Означення 2.2.** Нехай  $(X, \rho)$  – метричний простір. Відкритою (замкненою) кулею радіуса  $\varepsilon > 0$  з центром в точці  $x \in X$  називається множина

$$O_\varepsilon(x) = \{y \in X : \rho(x, y) < \varepsilon\} \quad (\bar{O}_\varepsilon(x) = \{y \in X : \rho(x, y) \leq \varepsilon\}). \quad (2.11)$$

**Означення 2.3.** Нехай  $(X, \rho)$  – метричний простір. Множина  $A \subseteq X$  називається відкритою, якщо для довільної точки  $x \in A$  існує відкрита куля  $O_\varepsilon(x) \subseteq A$ .

Система  $\tau_\rho$  усіх відкритих множин метричного простору  $(X, \rho)$  є топологією на множині  $X$ ; ця топологія називається топологією, що породжена метрикою  $\rho$ . Якщо  $d$  – інша метрика на  $X$  і  $\tau_\rho = \tau_d$ , то метрики  $\rho$  і  $d$  називають топологічно еквівалентними.

Якщо  $A \subseteq X$ , то метричний простір  $(A, \rho|_{A \times A})$  називається підпростором простору  $(X, \rho)$ .

Нехай  $x$  – довільна точка, а  $A$  – довільна підмножина метричного простору  $(X, \rho)$ . Число

$$\rho(x, A) = \inf \{ \rho(x, y) : y \in A \} \quad (2.12)$$

називається відстанню від точки до множини, а число

$$\rho(A, B) = \inf \{ \rho(x, y) : x \in A, y \in B \} \quad (2.13)$$

відстанню від множини  $A$  до множини  $B \subseteq X$ .

Підмножина  $A$  метричного простору  $X$  обмежена, якщо існує таке число  $M > 0$ , що  $\rho(x, y) < M$  для всіх  $x \in A, y \in A$ .

Діаметром множини  $A \subseteq X$  називається число

$$\text{diam}(A) = \sup\{\rho(x, y) : x \in A, y \in A\}. \quad (2.14)$$

**Означення 2.4.** *Послідовність  $(x_n)$  точок метричного простору  $(X, \rho)$  називається фундаментальною, якщо для довільного  $\varepsilon > 0$  існує такий номер  $n_0 \in \mathbb{N}$ , що  $\rho(x_n, x_m) < \varepsilon$  для всіх  $n > n_0$  і  $m > n_0$ .*

**Означення 2.5.** *Метричний простір  $(X, \rho)$  називається повним, якщо довільна фундаментальна послідовність його точок збігається до деякої точки простору  $(X, \rho)$ .*

Справедлива наступна фундаментальна теорема.

Теорема (принцип вкладених куль). *Метричний простір  $(X, \rho)$  повний  $\Leftrightarrow$  довільна послідовність непорожніх вкладених замкнених куль з прямыми до нуля радіусами має рівно одну спільну точку.*

Нехай  $(X, \rho_X), (Y, \rho_Y)$  – деякі метричні простори.

**Означення 2.6.** *Бієкція  $f : X \rightarrow Y$  називається ізометрією, або ізометричним відображенням, якщо  $\rho_Y(f(x), f(y)) = \rho_X(x, y)$  для довільної пари точок  $x \in X, y \in X$ . При цьому простори  $(X, \rho_X)$  і  $(Y, \rho_Y)$  називають ізометричними.*

Ізометричне вкладення одного метричного простору  $(X, \rho_X)$  в інший метричний простір  $(Y, \rho_Y)$  – це ізометрія простору  $(X, \rho_X)$  на деякий підпростір простору  $(Y, \rho_Y)$ .

**Означення 2.7.** *Поповненням метричного простору  $(X, \rho_X)$  називається*

повний метричний простір  $(\tilde{X}, \rho_{\tilde{X}})$  разом з таким ізометричним вкладенням  $i: X \rightarrow \tilde{X}$ , що множина  $i(X)$  скрізь щільна в  $\tilde{X}$ .

Довільний метричний простір має поповнення та всі поповнення одного й того ж метричного простору ізометричні між собою.

Поняття опуклості поєднує, на перший погляд, слабо пов'язані між собою області сучасної математики та інших наук. В математичному аналізі, теорії диференціальних рівнянь, математичній економіці, томографії, стереології знаходиться застосування різним аналогам опуклості. Найпростіший приклад – дослідження геометричних проблем аналізу функцій багатьох комплексних змінних. В цій тематиці в рівній мірі використовуються комплексний аналіз, топологія й опуклий аналіз.

**Означення 2.8** Множина  $M \subset R^n$  називається опуклою, якщо разом із двома своїми точками вона містить і відрізок, що їх з'єднує.

Іншими словами, множина опукла, якщо її перетин з довільною прямою зв'язний. Поруч із цим означенням існує поняття лінійної опуклості.

**Означення 2.9** Множина  $M \subset R^n$  називається лінійно опуклою, якщо через будь-яку точку доповнення до неї проходить гіперплощина, яка дану множину не перетинає.

Іншими словами, множина лінійно опукла, якщо доповнення до неї є об'єднанням гіперплощин.

**Означення 2.10** Множина  $M \subset R^n$  називається опуклою відносно множини  $W$  афінних лінійних підмноговидів в  $R^n$ , якщо її доповнення до  $R^n$  є об'єднанням елементів з  $W$ .

**Означення 2.11** Для довільної множини  $M \subset R^n$  назвемо підмножину  $M^* \subset W$  спряженою до множини  $E$ , якщо  $M^* = \{l \in W \mid l \text{ не перетинає } M\}$ , відповідно  $M^{**} = R^n \setminus \bigcup_{l \in M^*} l \subset R^n$  (остання формула коректна, оскільки кожен

елемент множини  $W$  є підмножиною  $R^n$ ).

Множина  $X$  називається опуклою, якщо вона містить всякий відрізок, кінці якого належать  $X$ , тобто  $\lambda x_1 + (1 - \lambda)x_2 \in X, x_1, x_2 \in X, \lambda \in [0; 1]$ .

## 2.5. Принципи побудови метаевристичних на основі поняття опуклості

Доведено [32-41], що для того щоб в метаевристичній моделі існувала повна система спадкових властивостей, необхідно і достатньо, щоб оператор кросоверу був геометричним, тобто, щоб на множині  $B$  існувала метрика  $\rho: B \times B \rightarrow R_+^1$  така, що для будь-яких розв'язків  $x, y \in B$  була справедливою рівність  $\rho(x, y) = \rho(x, K(x, y)) + \rho(y, K(x, y))$ .

Повернемося до визначення опуклої множини. *Опуклою підмножиною у метричному просторі називають підмножину, яка разом з будь-якими двома своїми точками містить всі точки геодезичної, що ці дві точки з'єднує.* З теореми випливає твердження, яке вказує на обмеженість еволюційного пошуку [85, 86]. Опуклими множинами є відрізок, пряма, півплощина, круг тощо.

Для еволюційно-фрагментарної моделі метаевристички з оператором кросоверу, що зберігає відносний порядок слідування фрагментів, та без оператора мутації будь-який розв'язок, отриманий у результаті застосування кросоверу, належить опуклій оболонці розв'язків з початкової популяції.

Таким чином, показано, що проблема пошуку початкових популяцій для реалізації еволюційно-фрагментарної моделі зводиться до *задачі покриття (або розбиття) простору перестановок опуклими множинами.*

Для оцінки якості метаевристичних алгоритмів використовують наступні методики, засновані на чисельних експериментах, а саме порівняння:

- 1) з відомим точним алгоритмом (для задач невеликої розмірності);
- 2) з відомими наближеними алгоритмами;

- 3) з методом випадкового пошуку на множині перестановок фрагментів;  
 4) з точними та наближеними розв'язками з відомих баз даних тестових задач.

## 2.6. Фрагментарні структури і їх властивості

Вимога застосовності генетичних, особливо «жадібних» алгоритмів для пошуку розв'язків дискретних задач призводить до поняття фрагментарної структури.

*Означення.* Фрагментарною структурою на скінченній множині  $X$  називається сімейство її підмножин  $E$  з такими властивостями:

1.  $\emptyset \in E$ ;
  2.  $\Delta Y \in E, Y \neq \emptyset, \exists y \in Y, Y \setminus \{y\} \in E$ .
- (2.15)

Елементи множини  $E$  називаються припустимими фрагментами.

Будь-який припустимий фрагмент фрагментарної структури може бути побудований з порожньої множини «жадібним» алгоритмом, на кожному кроці якого додається один елемент до вже побудованої підмножини таким чином, що знову утворена підмножина належить фрагментарній структурі.

Фрагментарні структури тісно пов'язані з такими комбінаторними об'єктами як матроїди та гридоїди. Зокрема, будь-який гридоїд та будь-який матроїд є фрагментарною структурою.

*Означення.* Рангом підмножини  $A \subseteq X$  називається величина  $rg(A)$ , що дорівнює максимальній з потужностей припустимих фрагментів, які містяться у множині  $A$ .

**Теорема 2.3.** Для того щоб фрагментарна структура була матроїдом, необхідно і достатньо, щоб для будь-яких підмножин  $A_1, A_2 \subseteq X$  виконувалась нерівність

$$rg(A_1 \cup A_2) + rg(A_1 \cap A_2) \leq rg(A_1) + rg(A_2).$$

Конструктивно фрагментарну структуру можна задавати інакше. Фрагментарною структурою будемо називати пару  $\Omega = (X, \diamond)$ , де  $X$  – множина,  $\diamond$  – бінарне відношення на множині  $X$ , що відповідає таким властивостям:

- 1)  $\emptyset \diamond \emptyset$ ;
- 2)  $\forall A_1, A_2 \subseteq X \quad A_1 \diamond A_2 \Rightarrow A_2 \diamond A_1$ ;
- 3)  $\forall A_1, A_2 \subseteq X, A_1 \diamond \emptyset, A_2 \diamond \emptyset \Rightarrow (A_1 \diamond A_2 \Leftrightarrow (A_1 \cup A_2) \diamond \emptyset)$ .

Такий підхід дозволяє будувати нові припустимі фрагменти з вже наявних шляхом їх об'єднання з урахуванням деяких умов приєднання.

Максимальним припустимим фрагментом фрагментарної структури є такий припустимий фрагмент, що не є власною підмножиною жодного іншого припустимого фрагменту.

«Жадібний» алгоритм на фрагментарній структурі  $\Omega = (X, \diamond)$  при заданому порядку фрагментів дозволяє отримати максимальний припустимий фрагмент за число кроків, що не перевищує числа елементів множини  $X$ .

Введення поняття фрагментарної структури дає змогу побудувати математичні моделі для великого класу прикладних задач, зокрема, погашення взаємозаборогованості, задачі симетричного розміщення рекламних блоків, задачі розкрою [35, 37]. Перевагою фрагментарних моделей є знаходження досить близьких до оптимальних субоптимальних рішень. Субоптимальні рішення – рішення, якість яких нижче по значенню (з точки зору їх природності для досягнення результату) ніж оптимальних [37].

## 2.7. Зведення задач оптимізації до задачі пошуку оптимальної перестановки

Для зведення задачі класифікації до пошуку оптимальної перестановки розглянемо рекурсивний метод побудови перестановок. Він полягає у тому, що один елемент фіксується в кінці, а останні перебираються.

Метод переміщення максимального елемента генерування елементів комбінаторної конфігурації передбачає вибір максимального елемента в множині з якої генеруються елементи комбінаторного об'єкта і його переміщення справа наліво. Переваги даного методу пояснюються зручним представленням елементів комбінаторного об'єкта для застосування методу направленої структуризації.

Основна ідея перекликається з відомим методом послідовного аналізу варіантів, але спеціально застосованого для розв'язування комбінаторних задач. Як відомо, у більшості задач на комбінаторних конфігураціях постає необхідність перераховувати велику кількість варіантів, порівняну з факторіальною величиною.

Метод об'єднує засоби комбінаторного аналізу та теорії графів і передбачає послідовне виконання трьох стадій.

1. Генерування у певній послідовності всіх елементів заданої комбінаторної конфігурації.

2. Побудову на її елементах орієнтованого графа, де дуга відповідає спаданню значень цільової функції.

3. Побудову поліноміального алгоритму розв'язку задачі на частково упорядкованих вершинах графа.

Цей метод було застосовано для розв'язання задач з лінійною та дробово-лінійною цільовими функціями на перестановках, розбиттях, сполученнях та розміщеннях.

Важливою задачею, що виникає при побудові залежності між елементами

перестановки за значеннями лінійної цільової функції, є задача організації впорядкування і вибору елемента перестановки. Цільову функцію для множини перестановок  $P_n = P(N_n)$  будемо позначати  $f(x)$ .

Для графів многогранників досить часто виникає наступна задача локалізації значень лінійної функції: знайти множину перестановок, в яких значення цільової функції рівно заданому значенню, тобто

$$x^* = \arg f(x), f(x^*) = y. \quad (2.16)$$

За умови, що перестановки, в яких цільова функція приймає задане значення, існують не завжди, вищесформульована проблема може бути інтерпретована як **задача**: визначити множину пар перестановок  $\bar{x}, \underline{x}$ , для яких при заданому  $y$  визначається

$$\bar{x} = \arg \min f(x), \underline{x} = \arg \max f(x). \quad (2.17)$$

Розглянемо підхід до розв'язання задачі (2.16), (2.17) на основі властивостей графа перестановок – **алгоритм локалізації значення лінійної функції на перестановках**.

Встановлення гамільтонового шляху в графі перестановок дозволяє розглянути початкову впорядкованість перестановок – вершин графа за значеннями цільової функції і застосувати метод направленої структуризації, при цьому для генерування елементів перестановки використовується рекурсивний метод.

## 2.8 Еволюційні метаевристики

Стимулом виникнення *еволюційних обчислень* свого часу стали кілька відкриттів у біології. Самі еволюційні алгоритми поділяються на декілька видів:

- генетичний алгоритм, призначений для оптимізації функції дискретних змінних, що акцентує увагу на рекомбінації генів;
- еволюційне програмування, орієнтоване на оптимізацію неперервних функцій без використання рекомбінації;
- еволюційні стратегії, орієнтовані на оптимізацію неперервних функцій з використанням рекомбінації;
- генетичне програмування, що використовує еволюційний метод для оптимізації комп'ютерних програм.

Ідея генетичних алгоритмів запозичена у живої природи і полягає в організації еволюції, ціль якої є отримання оптимального рішення у складній комбінаторній задачі. Загальна схема роботи генетичного алгоритму наступна.

Спочатку генерується початкова популяція особин (індивідуумів), тобто деякий набір рішень задачі. Як правило, це робиться випадковим чином. Потім моделюється розмноження всередині цієї популяції. Для цього випадково відбирається декілька пар індивідуумів, відбувається схрещування між хромосомами в кожній парі, а отримані нові хромосоми втілюються в популяцію нового покоління. У генетичному алгоритмі зберігається основний принцип природнього відбору – чим пристосованіший індивідуум (чим більше відповідне йому значення цільової функції), тим з більшою ймовірністю він буде брати участь у схрещуванні. При такому відборі члени популяції з більш високою пристосованістю з більшою ймовірністю будуть частіше вибиратися, ніж особини з низькою пристосованістю. Для кожного індивідуума може застосовуватися кросинговер, після якого отримані нащадки заміняють собою батьків і переходять

до мутації. Тоді моделюються мутації – у декількох випадково обраних особинах нового покоління змінюються деякі гени. Потім стара популяція частково або цілком знищується, здійснюється перехід до розгляду наступного покоління. Популяція наступного покоління в більшості реалізацій генетичних алгоритмів містить стільки ж особин, скільки початкова, але в силу відбору пристосованість у ній у середньому вища. Після чого процеси відбору, схрещування й мутації повторюються вже для цієї популяції, тощо. У кожному наступному поколінні спостерігається виникнення зовсім нових рішень задачі, що розв’язується. Серед них будуть як погані, так і хороші, але завдяки відбору число прийнятних рішень буде зростати. Робота генетичного алгоритму є ітераційним процесом, що продовжується, поки не виконається задане число ітерацій або інший заданий критерій зупинки. Таким чином, генетичні алгоритми є однією з важливих парадигм широкої області алгоритмів пошуку оптимальних рішень, що активно розвиваються, а з розвитком методів комп’ютерної підтримки ухвалення рішень вони набувають все більшого значення.

Програмні продукти в цій області діляться на декілька великих категорій. Перша категорія програм – це пакети, що реалізують класичний генетичний алгоритм з можливим відлагодженням параметрів керування основними операторами генетичних алгоритмів. Модель хромосоми в таких пакетах має, як правило, стандартну бінарну структуру, а функція відбору задана одним математичним виразом. До таких програм відносяться: пакет Evolver 4.0 компанії Palisade Corp, пакет GeneHunter компанії Ward System Group, тощо. Істотним недоліком цих методів є прив’язка до бінарної або числової моделі хромосоми (хромосома не може мати складної структури). До другої категорії програм відносяться спеціалізовані програми, призначені для вирішення конкретних завдань. Їх генетичні алгоритми розроблені і оптимізовані для вирішення вузької, чітко визначеної проблеми. Ефективність даної категорії програм залежить від

багатьох чинників і, зокрема, від таких, як генетичні оператори і вибір відповідних значень параметрів, а також від способу представлення рішення на хромосомі. Оптимізація цих чинників приводить до підвищення швидкості і стійкості пошуку, що істотно впливає на застосування генетичних алгоритмів. Третя категорія розробок по генетичних алгоритмах включає наукові дослідження, що полягають в дослідженні властивостей і характеристик різних генетичних алгоритмів, їх збіжності і виродженості (MATLAB 7.0, Evolvica, E2K).

Останніми роками одним із основних напрямків підвищення ефективності прикладних алгоритмів задачі оптимізації є розробка і дослідження *метаевристичних алгоритмів*, в яких деяка ключова ідея (метаевристика) слугує базисом для організації покрокових обчислень – ітераційного процесу, який породжує послідовність точок у відповідності із вбудованою процедурою. Якщо вбудована процедура – якийсь оптимізаційний алгоритм, то такі алгоритми називаються гібридними. Переважна більшість відомих гібридних методів розроблялася на базі двох концепцій: еволюційних обчислень та методів стохастичного локального пошуку. Тому саме на їх основі в роботі розроблені гібридні алгоритми, які використали позитивні якості, притаманні зазначеним підходам.

Існує два основних способи поєднання переваг ГА та  $G$ -алгоритмів. У першому із цих гібридних алгоритмів, названому модифікованим ГА (МГА) [], в схемі еволюційних обчислень як оператор мутації була використана модифікація  $G$ -алгоритмів. В частинному випадку цей алгоритм охоплює відомий клас меметичних алгоритмів, які почали розвиватися останнім часом.

Використання  $G$ -алгоритмів в еволюційних обчисленнях як процедури підсилення масштабу мутації, яке було здійснене в МГА, дозволило підвищити ефективність пошуку варіантів розв'язку порівняно з автономним застосуванням  $G$ -алгоритмів та ГА. При цьому аналіз результатів використання такого підходу

для розв'язання багатьох задач КО показав, що застосування в комбінованому методі  $G$ -алгоритму у повному обсязі на початкових етапах ГА можна уникнути, якщо основні його параметри, які визначають умови переходу до наступного кроку при локальному пошуку на кожній ітерації, змінювати узгоджено з динамікою змін параметрів ГА та поведінкою цільової функції.

В іншому алгоритмі, названому  $G$ -ГА, на відміну від поширених способів гібридизації комбінаторних алгоритмів, ГА виступає не як загальна схема (метаевристика), а як вбудований алгоритм. Алгоритм  $G$ -ГА дозволяє здійснювати автоматичну адаптацію своїх параметрів в ході обчислювального процесу: за рахунок послаблення на початковій стадії пошуку умов прийняття до розгляду варіантів розв'язку збільшується кількість точок простору  $X$ , що можуть проглядатися за певний проміжок часу, та підвищується ймовірність потрапляння до «зони притягання» більш точного варіанту розв'язку.

## 2.9 Ройові метаевристики

Ройовий інтелект (Swarm Intelligence) описує колективну поведінку децентралізованої системи, яка здатна до самоорганізації. Інколи ройовий інтелект називають колективним інтелектом і він розглядається в теорії штучного інтелекту як метод оптимізації.

Системи ройового інтелекту як правило складаються з множини агентів (мультиагентна система), які локально взаємодіють між собою, а також з навколишнім середовищем. Самі агенти зазвичай досить прості, але всі разом, взаємодіючи, вони створюють ройовий інтелект. Прикладом у природі може служити колонія мурах, рій бджіл, зграя птахів, косяк риб.

Особливість ройових метаевристик полягає в тому, що еволюційні оператори не застосовуються безпосередньо до рішень. Натомість простір рішення

модифікується для заохочення вибору значень змінних рішень, які призвели до гарних рішень у попередніх ітераціях. В результаті генеруються абсолютно нові рішення в кожній ітерації, а не модифікуються рішення з попередніх ітерацій.

Алгоритм колонії мурах (мурашиний алгоритм, алгоритм оптимізації мурашиної колонії, ant colony optimization, ACO) – один з ефективних поліноміальних ройових алгоритмів для знаходження наближених розв'язків задачі комівояжера, а також завдань пошуку маршрутів на графах. Алгоритм можна описати у 3 етапи (рис. 2.3):

1. Перша мураха знаходить джерело їжі (F) через якийсь шлях (a). Потім повертається до гнізда (N), залишивши за собою слід з феромонів (b).
2. Мурахи вибирають будь-який шлях, але шлях, на якому міститься феромон робить його більш привабливим як найкоротший шлях.
3. Мурахи вибирають коротший шлях, а з часом інші шляхи втрачають щільність феромонового сліду.

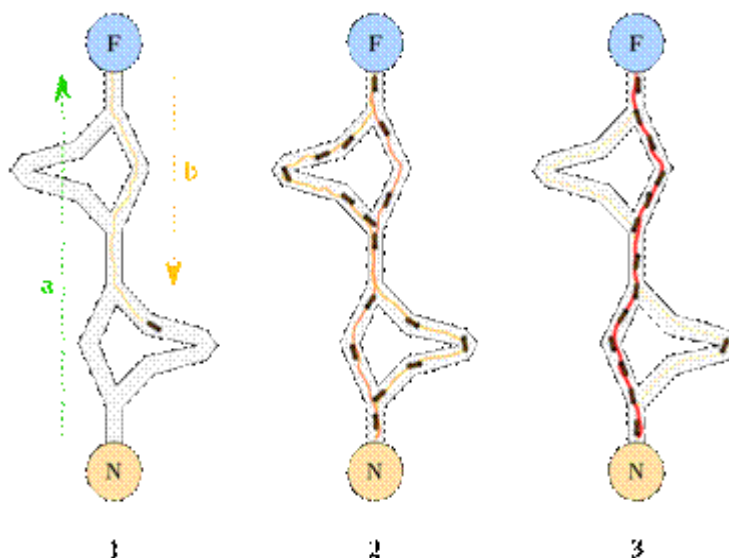


Рис. 2.3. Поведінка мурах в колонії

Бджолиний алгоритм (алгоритм оптимізації наслідуванням бджолиної колонії, *artificial bee colony optimization*, ABC) – один з поліноміальних евристичних алгоритмів для розв’язку оптимізаційних задач. Зокрема, використовується для розв’язку задачі календарного планування з множиною робіт, кожна з яких складається з однієї або більше операцій.

Метод рою частинок, МРЧ (*Particle Swarm Optimization*, PSO) – метод чисельної оптимізації, для використання якого не потрібно знати точного градієнта оптимізованої функції. МРЧ оптимізує функцію, підтримуючи популяцію можливих розв’язків, названих частками, і переміщаючи ці частки в просторі розв’язків згідно із простою формулою. Переміщення підпорядковуються принципу найкращого знайденого в цьому просторі положення, що постійно змінюється при знаходженні частками вигідніших положень.

Метод рою частинок можна застосовувати для задачі пошуку максимуму та мінімуму функцій, задачі календарного планування, задач машинного навчання (найчастіше, навчання нейронних мереж і розпізнання зображень), а також для задач, типових для генетичних алгоритмів.

Розглянемо модель пересування агентів рою.

Припустимо, існує безліч  $D$  – агентів рою  $d_i (i=1, \dots, n)$ , спільне взаємодія яких забезпечує вирішення певного завдання  $P=p_1, \dots, p_m$ .

Всі агенти  $d_i (i=1, \dots, n)$  однакові. Агент  $d_i \in D$  може здійснювати обмін повідомленнями з деякою підмножиною агентів  $d_i \in D$  яка знаходяться в межах певної зони, обмеженої радіусом  $R$ , яку зазвичай називають «зоною видимості». За допомогою такого інформаційного обміну агенту  $d_i$  може бути доступна інформація про стан агентів підмножини  $D_i$ .

Агент  $d_i$  рухається на відстань  $r$  від своїх найближчих сусідів.

Для вирішення поставлених завдань  $p_i \in P$  оператор надає кожному агенту  $d_i$  карту потенціалів, яка визначає перспективність руху в певному напрямку.

Однак, у агентів немає відомостей про можливі перешкоди на їх шляху. У міру руху у кожного агента формується власна потенційна картина світу.

Рух кожного агента  $d_i$  характеризується напрямком  $x_i$  і значенням потенціалу обраного шляху  $q_i$ .

Кожен агент володіє базою знань про навколишній світ, яку він поповнює по шляху проходження (на основі показників датчиків, методів комп'ютерного зору, інформації від сусідів, перерахунку значення потенціалу

Потенційна функція характеризує можливість досягнення мети  $s$  при збереженні руху в напрямку  $x_t^i$ . Наприклад, при відомому напрямку на ціль.  $s_t^i$  для агента  $i$  в момент часу  $t$  можливо прокласти шлях  $q_t^i = q_t^i(x_t^i) = \{x_t^i, s_t^i\}$ .

Кожен агент при остаточному визначенні напрямку руху намагається вибрати маршрут так, щоб уникнути зіткнень. Наприклад, можна задати функцію  $\varphi(xti)$ , яка відхиляє рух у випадковому напрямку в тому випадку, якщо агент  $i$  виявляє перешкоду в напрямку  $x_t^i$  на відстані ближче ніж  $r$ .

Зазначену модель можна використовувати при розв'язку задачі оптимальної класифікації.

## Висновки до 2 розділу

Задача оптимальної класифікації має широке практичне застосування. На жаль, вона є NP-важкою і, крім того, містить велику кількість змінних, які в свою чергу можуть мати невизначений (непрямий) характер. До того ж у прикладних задачах, як правило, є багато обмежень, які ускладнюють застосовність відомих алгоритмів. Тому для задач оптимальної класифікації є виправданим застосування метаевристик.

При пошуку екстремуму справедливим є гіпотеза великої долини, що частково пояснює працездатність метаевристичних алгоритмів. Якщо в популяції

збираються локальні оптимуми, які відповідно до гіпотези сконцентровані в одному місці, і чергове рішення обирається десь між двома довільними локальними оптимуму, то такий процес має багато шансів знайти глобальний оптимум. Для зменшення долини доцільно використовувати опуклі множини.

Вимога використання метаевристичних алгоритмів, а також опуклих множин, призводить до поняття фрагментарної структури. Введення поняття фрагментарної структури дає змогу побудувати математичні моделі для великого класу прикладних задач, зокрема, розміщення виробництва.

Наслідком застосування фрагментарних структур є зведення задач оптимізації до задачі пошуку оптимальної перестановки. Для цього було розглянуто рекурсивний метод побудови перестановок. Він полягає у тому, що один елемент фіксується в кінці, а останні перебираються.

Такі дискретні екстремальні задачі, як задачі оптимальної класифікації, допускають формулювання, як задачі на фрагментарних структурах. Отже для них є доцільним використання метаевристичних алгоритмів. При цьому фрагментарний алгоритм має властивість досяжності для заданого класу задач, якщо для будь-якої індивідуальної задачі класу можна знайти таке впорядкування елементарних фрагментів, за якого застосування фрагментарного алгоритму призводить до точного оптимального розв'язку задачі.

Якщо фрагментарний алгоритм має властивість досяжності, то точний оптимальний розв'язок екстремальної задачі може бути знайдено шляхом перебору всіх перестановок елементарних фрагментів і застосування фрагментарного алгоритму для кожної такої перестановки. Однак, такий алгоритм, безумовно, є складним. Якщо обмежитися не всіма перестановками, а якоюсь їх частиною, обраною за певними правилами, то можна отримати різні варіанти евристичних алгоритмів пошуку.

Фрагментарний алгоритм для таких задач дозволяє відшукати допустимий розв'язок і, відповідно, отримувати значення критерію на цьому розв'язку. Однак допустимий розв'язок, отриманий шляхом застосування фрагментарного

алгоритму, взагалі кажучи, не є оптимальним. Для пошуку наближених оптимальних розв'язків подібних задач можна запропонувати модель, що поєднує метаевристичні та генетичні алгоритми, яка буде розглянута в розділі 4.

## РОЗДІЛ 3. ЗАДАЧА РОЗМІЩЕННЯ ВИРОБНИЦТВА

### 3.1 Задача покриття графу зірками

Розглянемо тепер задачу покриття графа зірками, яка часто виникає в теорії автоматичного проектуванні, в теорії синтезу схем, в задачах землекористування та розміщення виробництва [36].

Зіркою називається граф, що ізоморфний повному дводольному графу  $K_{1,n}$  де  $i = 1, 2, \dots$  (рис. 3.1).

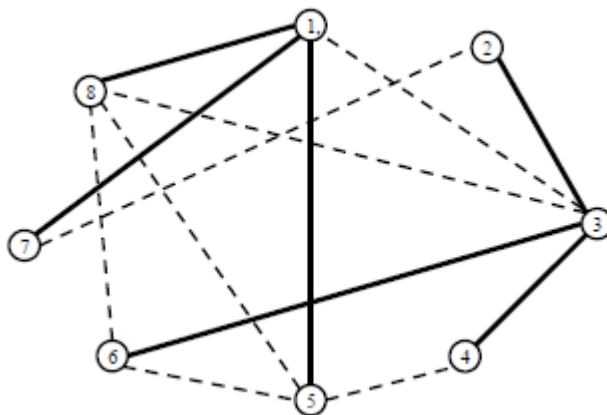


Рис. 3.1. Граф, що є зіркою

Задача покриття може бути сформульована таким чином. Заданий граф  $G = (V, E)$ , ребра якого  $E$  зважені функцією  $p: E \rightarrow R_+^1$ . Знайти підмножину мінімальної ваги з неперетинних по вершинах зірок у цьому графі, об'єднання яких містить всі вершини графу  $G$ .

$$G = (V, E) : T = \{T_1, T_2, \dots, T_i\}, T_i = (V_i, E_i), E_i \neq \emptyset$$

$$\begin{aligned} & \forall i, j = 1, 2, \dots, t \\ & V_i \cap V_j = \emptyset \text{ за умов } i \neq j \\ & \bigcup_{i=1}^t V_i = V \\ & \rho(T) = \sum_{i=1}^t \sum_{e \in E_i} \rho(e) \rightarrow \min \end{aligned} \quad (3.1)$$

Відомо, що задача покриття графу зірками є *NP*-складною [36].

Побудуємо фрагментарну модель задачі. Множиною елементарних фрагментів є множина  $E$  всіх ребер графу, елементи якого нумеруються числами  $1, 2, \dots, m$ .

Фрагментом цієї моделі є будь-який підграф в графі, кожна компонента зв'язності якого є зіркою.

Умова приєднання елементарного фрагменту є наступною. Або ребро  $e_i \in E$  має рівно одну спільну вершину з однією з зірок фрагмента і ця вершина – центр зірки, або це ребро не має загальних вершин з уже обраними ребрами. Розглядаються довільні впорядкування ребер графу, кожне з яких описується перестановкою з групи перестановок  $S_m$ .

Розглянемо фрагментарний алгоритм розміщення.

Складається список елементарних фрагментів (ребер графу), упорядкований відповідно до деякої перестановки чисел  $1, 2, \dots, m$ . За розв'язок на початковому кроці вибирається порожня множина. На кожному кроці алгоритму вибираємо черговий по порядку фрагмент (ребро)  $e_i \in E$  зі списку. Якщо виконується умова приєднання, обране ребро приєднується до поточного фрагменту як промінь відповідної зірки. В іншому випадку відбувається перехід до наступного ребра зі списку.

Алгоритм закінчує роботу, коли список елементарних фрагментів (ребер) буде порожнім.

Результатом роботи фрагментарного алгоритму є допустиме покриття графу зірками.

За допомогою фрагментарного алгоритму за рахунок вибору нумерації ребер можна отримати будь-яке наперед задане покриття графу зірками. Тому має місце теорема досяжності [36].

**Теорема 3.1.** Будь-який допустимий розв'язок задачі покриття зваженого графу зірками досяжний в рамках фрагментарної моделі.

*Доведення.* Розглянемо допустиме покриття графу зірками. Впорядкуємо зірки, в будь-якому порядку, а потім будемо нумерувати ребра послідовно по зірках. Раніше нумеруємо ребра першої зірки, потім другої, і так далі. Тоді фрагментарний алгоритм для заданої нумерації ребер побудує потрібне покриття.

Критерієм в задачі є сумарна вага знайденого покриття графа зірками, отриманого шляхом застосування фрагментарного алгоритму.

Цікавим є окремий випадок задачі покриття графа зірками – *задача про регулярне покриття графа  $k$ -зірками*, тобто зв'язними графами, у кожного з яких одна вершина має степінь  $k$ , а всі інші 1.

Будемо розглядати задачу регулярного покриття зваженого графа  $k$ -зірками в наступному формулюванні: знайти набір неперетинаних підграфів в розглянутому графі з  $n$  вершинами і  $m$  ребрами, кожна компонента якого ізоморфна зірці з  $k$  променями, причому набір містить всі вершини початкового графу. Зрозуміло, що необхідною умовою існування такого покриття є умова подільності числа  $n$  на число  $k+1$ .

Формально задача про регулярне покриття  $k$ -зірками має наступний вигляд.

У заданому графі  $G = (V, E)$ , ребра якого  $E$  зважені функцією  $p: E \rightarrow R^1$ , знайти підмножину мінімальної ваги неперетинаних по вершинах  $k$ -зірок,

об'єднання яких містить всі вершини графа  $G$ . Зрозуміло, допустимий розв'язок задача матиме лише в тому випадку, коли число вершин графа  $G$  кратне  $k+1$ , тобто має місце рівність  $n = (k + 1) \cdot t$ , де  $t$  – число зірок в покритті.

У цьому випадку формальна постановка задачі має вигляд:

$$\begin{aligned}
 & \forall i, j = 1, 2, \dots, t \\
 & V_i \cap V_j = \emptyset \text{ за умов } i \neq j \\
 & \bigcup_{i=1}^t V_i = V \\
 & \forall i = 1, 2, \dots, t \\
 & \rho(T) = \sum_{i=1}^t \sum_{e \in E_i} \rho(e) \rightarrow \min \tag{3.2}
 \end{aligned}$$

Для розглянутої задачі пропонується фрагментарна модель [20-23], в якій множина елементарних фрагментів є множина всіх ребер графу, елементи якого нумеруються числами  $1, 2, \dots, m$ . Фрагментом в цій моделі є будь-який підграф у графі, кожна компонента зв'язності якого є зіркою з числом вершин не більше  $k$ . Умова приєднання елементарного фрагмента є такою. Або ребро  $e_i \in E$  має тільки одну спільну вершину з однією з зірок фрагмента, число променів якої не перевершує  $k$ , і ця вершина – центр зірки, або це ребро не має загальних вершин з уже обраними ребрами. Розглядаються довільні впорядкування ребер графу. Кожне з цих упорядкувань описується перестановкою з групи перестановок  $S_m$ .

Фрагментарний алгоритм розміщення полягає в наступному. Складається список елементарних фрагментів (ребер), упорядкований відповідно до деякої перестановки чисел  $1, 2, \dots, m$ . За розв'язок на початковому кроці вибирається порожня множина. На кожному кроці алгоритму вибираємо черговий по порядку фрагмент (ребро)  $e_i \in E$  зі списку. Якщо виконується умова приєднання, обране

ребро приєднується до поточного фрагменту в якості променів. В іншому випадку відбувається перехід до наступного ребра зі списку. Після побудови чергової  $k$ -зірки, всі не вибрані ребра інцидентні центру зірки видаляються зі списку.

Алгоритм закінчує роботу, коли список елементарних фрагментів (ребер) буде порожнім. Якщо після цього не всі вершини графа  $G$  опинилися в покритті, то додаються бракуючі ребра, яким приписується вага  $+\infty$  (штраф).

Результатом роботи фрагментарного алгоритму є допустиме покриття графа  $k$ -зірками. Очевидно, що всякий допустимий розв'язок є досяжним в рамках даної моделі. Таким чином має місце наступна теорема [36].

**Теорема 3.2** Будь-який допустимий розв'язок задачі покриття зваженого графа  $k$ -зірками може бути отриманий шляхом належного вибору перестановки елементарних фрагментів.

Доведення теореми аналогічно доведенню теореми 3.2.

Критерієм в задачі є сумарна вага знайденого покриття графа  $k$ -зірками, отриманого шляхом застосування фрагментарного алгоритму з урахуванням штрафів.

Задача покриття графа зірками має фрагментарну структуру та, відповідно, будь-яке допустимий розв'язок задачі може бути отримане шляхом застосування фрагментарного алгоритму [31,46]. Отже, для задачі покриття графу є справедливими всі твердження Розділу 2. Крім того, задача покриття графу належить до задач дискретної оптимізації, і являє собою по суті задачу класифікації [87].

### 3.2 Найпростіші варіанти задачі розміщення виробництва

Задачі територіального розміщення різноманітних об'єктів є поширеними та актуальними в наш час і мають масовий характер. Наприклад, підприємства

торгівлі, громадського харчування, побутового обслуговування, поштові відділення тощо повинні розташовуватися так, щоб населення могло отримувати відповідні послуги з певним ступенем доступності, і, з цього погляду, що більше їх, то краще. З іншого боку, чим менше таких підприємств в регіоні обслуговування, тим вони більші і, отже, собівартість їх утримання нижча, тобто існує певна суперечність між доступністю та рентабельністю. Подібні завдання розв'язують місцеві, селищні, сільські органи державного управління. Аналогічні задачі виникають також у керівників комерційних фірм: проблеми розміщення мережі бензозаправок, торгових відділень великих торгових фірм, філій банків, будівельних, транспортних компаній тощо. Ці приклади доповнює ряд завдань, пов'язаних з розміщенням виробничих потужностей нафтогазовидобувних підприємств, металургійних комплексів, транспортних вузлів, і всі вони потребують розв'язання задач розміщення з подібними критеріями якості.

Як і багато інших математичних моделей, задача розміщення виробництва допускає різні інтерпретації, що виправдовує академічний інтерес до її дослідження.

Представимо формалізацію задачі розміщення виробництва у наступному вигляді.

Задано повний дводольний граф  $K_{n,m}$ , ребра якого  $\{(i, j)\}_{i=1, j=1}^{i=n, j=m}$  зважені числами  $\{w_{ij}\}_{i=1, j=1}^{n, m}$ .

Дві групи попарно несуміжних вершин графа будемо називати відповідно споживачі і пункти обслуговування. Вага ребра інтерпретується як вартість обслуговування відповідного клієнта. Крім того, кожній вершині-пункту обслуговування  $j$  приписана вага  $\alpha_j$ .

Задача полягає у знаходженні часткового графа мінімальної ваги, який містив би всі вершини-споживачі і частину вершин-пунктів обслуговування, причому кожен споживач з'єднаний лише з одним пунктом обслуговування. При

цьому ступінь вершини-пункту обслуговування  $j$  не повинен перевищувати величини  $K$  (обмеження на кількість клієнтів). Будемо шукати розв'язок у вигляді булевої матриці  $\{x_{ij}\}_{i=1, j=1}^{n,m}$ .

Розглянемо детальніше процес задання виробників та споживачів.

Нехай множина  $I = \{1, \dots, I\}$  задає перелік можливих пунктів розміщення підприємств з виробництва деякого однорідного продукту. У будь-якому з пунктів  $i \in I$  можна відкрити підприємство. Величина  $c_j \geq 0$  задає відповідні витрати. Відкрите підприємство може виробляти продукцію для споживачів в необмеженій кількості.

Перелік споживачів задається множиною  $J = \{1, \dots, J\}$ . Для кожної пари  $ij$  відома величина  $g_{ij} \geq 0$  витрат на виробництво і доставку продукції споживачеві. Завдання полягає в тому, щоб знайти таку множину  $S \in I, S \neq \emptyset$  підприємств, що відкриваються, щоб з мінімальними витратами задовольнити потреби всіх споживачів. З використанням введених позначень оптимізаційна постановка задачі може бути записана наступним чином:

$$F(S) = \sum_{i \in S} c_i + \sum_{j \in J} \min_{i \in S} g_{ij} \rightarrow \min_{S \in I} \quad (3.3)$$

Сформульована задача є узагальненням відомої задачі про покриття множинами [23], тому відноситься до числа NP-важких в сильному сенсі. Для розв'язку найпростішої задачі розміщення виробництва розроблено точні алгоритми, наближені алгоритми з гарантованими оцінками точності, лагранжевої евристики, імовірнісні ітераційні алгоритми локального пошуку. Знайдено поліноміально розв'язані класи задач.

Задача розміщення з обмеженнями стосовно потужностей підприємств є узагальненням найпростішої задачі розміщення. На відміну від останньої в ній передбачається, що кожне підприємство може виробляти продукцію тільки в обмежених кількостях. Таке природне і важливе припущення злегка змінює математичну модель і сильно ускладнює методи розв'язку оптимізаційної задачі.

Досить розповсюдженими є задача розміщення виробництва з обмеженнями стосовно потужностей, задача про  $p$ -медіану,  $p$ -медіану з уподобаннями клієнтів, про  $(r|p)$ -центроїд і конкурентна задача розміщення, задача розкрою та упаковки, задача про постачання, а також задача складання розкладу [14, 22, 28].

### **3.3 Метаевристичні алгоритми пошуку оптимального розв'язку складних задач розміщення виробництва**

Незважаючи на те, що алгоритми метаевристики не гарантують оптимальності знайденого розв'язку, ці методи є досить потужними інструментами. Це обумовлено тим, що при знаходженні оптимального розв'язку метаевристичні алгоритми, на відміну від традиційних методів оптимізації, забезпечують паралелізм, тобто пошук розв'язку ведеться серед множин альтернативних варіантів. Отже, з метою забезпечення практичності, коли потрібно за певний час знайти одне або кілька субоптимальних розв'язків, використання таких алгоритмів є дуже ефективних.

Розглянемо особливості метаевристичних алгоритмів для розв'язку задачі розміщення виробництва. Застосування метаевристичних алгоритмів для задачі класифікації має три основні характеристики:

- 1) орієнтація на популяцію та наявні дані. Оптимізація виконується таким чином, що поточні «погані» розв'язки використовуються для створення нових кращих розв'язків;

2) використання додаткових «штучних» функцій. Враховуючи те, що задача розміщення виробництва є варіацією задачі класифікації, в якій важливим є недопущення віднесення того чи іншого елементу до неіснуючого класу, доцільним є введення штрафної функції.

Використання штрафної функції допомагає швидкому знаходженню розв'язку і уникати передчасної збіжності метаевристичного алгоритму.

Ідея методу штрафної функції полягає в погіршенні значень функції пристосованості (fitness function) наприклад, при появі неприпустимих хромосом у генетичному алгоритмі. Іншими словами, якщо розглядається задача мінімізації, то введена штрафна функція повинна різко збільшувати значення fitness function. І навпаки, якщо вирішується задача максимізації, то штрафну функцію слід побудувати таким чином, щоб вона різко зменшувала значення fitness function.

3) орієнтація на змінні моделі. Якщо в поточній сукупності немає прийняттого розв'язку враховуючи штрафну функцію, необхідно модифікувати останню.

### **3.4 Фрагментарний алгоритм розв'язку задачі розміщення виробництва**

Множиною елементарних фрагментів є множина ребер дводольного графу  $K_{n,m}$ , які нумеруються числами  $1,2,\dots,nt$ . Фрагментом задачі є будь-який частковий граф графу  $K_{n,m}$ , кожна компонента якого є зіркою з числом променів не переважаючим  $K$ .

Фрагментарний алгоритм розміщення полягає в наступному:

- складається список елементарних фрагментів, упорядкований відповідно до деякої перестановки ребер;
- вибирається чергове по порядку ребро  $(i, j)$  зі списку. Клієнт з номером  $i$  закріплюється за пунктом обслуговування з номером  $j$ ;

- всі інші ребра, що інцидентні вершині і видаляються зі списку;
- алгоритм закінчує роботу, коли список стане порожнім.

Результатом роботи фрагментарного алгоритму буде розподіл клієнтів між пунктами обслуговування, яке є допустимим розв'язком розглянутої задачі.

Має місце теорема досяжності в рамках заданої моделі.

**Теорема 3.3.** Будь-який допустимий розв'язок задачі розміщення виробництва може бути отримано шляхом належного вибору перестановки елементарних фрагментів.

Доведення теореми аналогічно доведенню теореми 3.2.

Для пошуку наближеного оптимального розв'язку задачі пропонується використовувати еволюційно-фрагментарну модель на перестановках [23].

### **3.5 Ройовий алгоритм для задачі розміщення виробництва**

Виділимо загальні для всіх алгоритмів ройового інтелекту поняття на підставі системного опису [36].

1. Елемент. Під елементом прийнято розуміти найпростішу неподільну частина системи. Елементом рою є частка.

2. Підсистема. Концепція ройового інтелекту допускає виділення груп часток. Наприклад, можливі такі ситуації:

- зв'язок між частками в групі є сильнішим, ніж між частками з різних груп (островкова модифікація алгоритму рою часток);

- виділяється група часток з особливою поведінкою, які, наприклад, не використовують інформацію про попередній досвід для запобігання передчасної збіжності алгоритму (розвідники в методі рою бджіл).

3. Структура. Структура ройових систем задається правилами поведінки часток і обміном інформацією. Особливістю таких систем є відсутність центру, який управляв би частками безпосередньо.

4. Зв'язок між частками відбувається шляхом непрямого обміну досвідом.

5. Стан. Під станом (зрізом, описом системи в окремий момент) розуміється стан всіх часток, стан об'єкту для непрямого обміну досвідом.

6. Поведінка. Поведінка рою задається алгоритмом роботи рою, який переводить множину часток з одного стану в інший.

7. Зовнішнє середовище. Під зовнішнім середовищем розуміється множина елементів, які не входять до системи, але зміна їх стану викликає зміну поведінки системи.

8. Рівновага, стійкість і розвиток. З точки зору ройового інтелекту ці поняття пов'язані зі збіжністю алгоритмів. Збіжність алгоритмів ройового інтелекту є окремою галуззю досліджень. Висока стійкість може привести до швидкої збіжності пошуку як до оптимального розв'язку, так і до деякого локального екстремуму, далекому від оптимального. Низька стійкість призводить до підвищення ймовірності потрапити в область глобального екстремуму, але при цьому підвищується ймовірність і передчасного виходу з цієї області через недостатньо докладне дослідження.

Робота ройового алгоритму включає в себе наступні кроки.

1. Генерація початкової множини часток. Деяким чином, як правило, з використання рандомізації, в просторі пошуку розподіляються множина часток  $S$ . Номер ітерації  $j = 1$ .

2. Обчислення критерію для кожної частки як функції від позиції частки  $f(X_{ij})$ . Для окремих алгоритмів, таких як, алгоритми колонії мурах, обчислення критерію виконується після переміщення частинок, тому для них другий крок на першій ітерації пропускається.

3. Переміщення часток (міграція). На цьому кроці реалізується головна особливість алгоритмів ройового інтелекту – виконання кожною частинкою своїх дій на підставі:

- індивідуальних правил і досвіду;
- непрямого обміну інформацією з іншими частинками рою;
- стохастичних властивостей.

4. Перевірка умови завершення. Якщо умова виконана, процес завершується, значення  $X^{final}_{best}$  буде кінцевим результатом, інакше відбувається перехід до кроку 2 (зі збільшенням номера ітерації  $j$  на одиницю).

Сформулюємо задачу розміщення виробництва для зручності розв'язку алгоритмом рою часток наступним чином. Дано  $n$  підприємств, кожне з яких випускає певний однотипний продукт. Вартість випуску  $p_i$  одиниць продукту  $i$ -тим підприємством визначається за співвідношенням:

$$f_i = a_i p_i^2 + b_i p_i + c_i \quad (3.3)$$

Випуск продукту кожним з підприємств має задовольняти такому обмеженню:

$$p \min_i \leq p_i \leq p \max_i. \quad (3.4)$$

Необхідно випустити рівно  $s$  одиниць продукту, мінімізувавши при цьому загальні витрати.

У наведених вище формулах  $a_i$ ,  $b_i$ ,  $c_i$ ,  $pMin_i$ ,  $pMax_i$  – деякі константи, що характеризують  $i$ -те підприємство.

Додатково можна ввести додаткову величину, що характеризує негативний вплив випуску  $p_i$  одиниць продукту  $i$ -тим підприємством на навколишнє середовище чи економіку:

$$g_i = \lambda_i p_i^2 + \beta_i p_i + \gamma_i, \quad (3.5)$$

де  $\lambda_i$ ,  $\beta_i$ ,  $\gamma_i$  – коефіцієнти, що характеризують негативний вплив  $i$ -ого підприємства на стан навколишнього середовища чи економіки.

Сумарні затрати для кожного підприємства тепер вираховуються за формулою:

$$F_i = f_i + h g_i, \quad (3.6)$$

де  $h$  – коефіцієнт, що показує вагу фактору негативного впливу на навколишнє середовище чи економіку у загальній функції затрат.

Визначення коефіцієнту  $h$  не є тривіальною задачею. Очевидно, що в залежності від різних значень коефіцієнтів, які описують кожне підприємство, значення параметра  $h$  має змінюватись. Зазвичай використовується наступний алгоритм визначення коефіцієнту  $h$ :

1) Обраховується значення індивідуального коефіцієнту врахування впливу на навколишнє середовище:

$$h_i = \frac{a_i p \max_i^2 + b_i p \max_i + c}{\lambda_i p \max_i^2 + \beta_i p \max_i + \gamma_i}. \quad (3.7)$$

2) Підприємства сортуються за зростанням величини  $h_i$ .

3) У порядку сортування по черзі додаються значення  $p_{max}x_i$  до тих пір, поки отримана сума не стане більше або рівною за значення  $s$  (очевидно, такий момент колись настане, інакше задача не має розв'язку).

4) У якості глобального коефіцієнта  $h$  приймається значення  $h_i$  останнього розглянутого підприємства [36].

### 3.6 Алгоритм мурашиної колонії для задачі розміщення виробництва

Представимо мурашиний алгоритм у вигляді наступного набору команд:

1. Створення агентів.
2. Пошук відповідного розв'язку.
3. Зміна феромону.
4. Допоміжні дії (не обов'язково).

Розглянемо детальніше ці кроки.

#### 1. Створення агентів.

Початкове розташування, де розміщується агент, залежить від початкових умов і обмежень задачі. Агенти можуть або бути в одній точці, або в різних з повтореннями, або в різних без повторень.

Вказується початкове значення феромона таким чином, щоб значення не були нульовими.

#### 2. Пошук відповідного розв'язку.

Імовірність того, що відбудеться перехід з вершини  $i$  в  $j$ , можна визначити за формулою:

$$P_{ij}(t) = \frac{\tau_{ij}(t)^\alpha \left(\frac{1}{d_{ij}}\right)^\beta}{\sum \tau_{ij}(t)^\alpha \left(\frac{1}{d_{ij}}\right)^\beta}, \quad (3.8)$$

де  $\tau_{ij}(t)$  – рівень феромону,

$d_{ij}$  – евристична відстань,  
 $\alpha, \beta$  – константні параметри.

Якщо  $\alpha=0$ , з більшою ймовірністю вибереться найближче місце. Якщо  $\beta=0$ , вибір буде ґрунтуватися лише на феромони. Знаходить компроміс між цими двома величинами слід в експериментальний спосіб.

### 3. Зміна феромону .

Рівень феромону змінюється за формулою:

$$\tau_{ij}(t+1) = (1 - \rho)\tau_{ij}(t) + \sum \frac{Q}{L_k}, \quad (3.9)$$

де  $\rho$  – інтенсивність випаровування,

$L_k(t)$  – вартість поточного розв'язку  $k$ -го мурашки,

$Q$  – параметр, який має значення порядку вартості оптимального розв'язку,

$\frac{Q}{L_k}$  – феромон, який відкладається  $k$ -м мурахою, що використовує ребро.

### 4. Допоміжні дії.

В основному використовують алгоритми локального пошуку.

## Висновки до 3 розділу

У даному розділі розглянуто постановку задачі покриття графів типовими підграфами. Показано, що ці задачі можуть бути сформульовані як задачі на множині з фрагментарною структурою, а також задача класифікації. Для всіх задач розглянутих класів фрагментарний алгоритм пошуку допустимого розв'язку по заданій перестановці елементарних фрагментів має поліноміальну

трудомісткість. Таким чином, для всіх розглянутих в дисертації задач наближений розв'язок може бути знайденим стандартним еволюційно-фрагментарним алгоритмом, базова множина якого є множиною перестановок.

Розглянуто постановку задачі розміщення виробництва у термінах задачі покриття графів. Для розглянутої задачі наведено особливості алгоритму ройового та еволюційного алгоритму пошуку оптимального розв'язку.

## РОЗДІЛ 4. ПРОГРАМНІ РЕАЛІЗАЦІЇ АЛГОРИТМІВ ПОШУКУ ЗАДАЧ ОПТИМАЛЬНОЇ КЛАСИФІКАЦІЇ

### 4.1 Генерація випадкових графів

Розглянемо докладні приклади графів (програмна реалізація вказана в Додатках А, Б, В), якими пізніше скористаємося для тестування досліджуваних алгоритмів. Наведемо два випадкових графи.

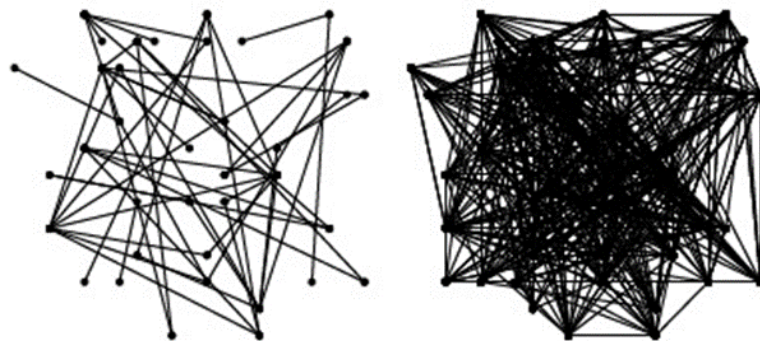


Рисунок 4.1. Два випадкових графи

Обидва наведених графи містять по 50 вершин. Розріджений граф у верхній частині рисунку містить 50 ребер, а насичений граф в нижній частині рисунку – 500 ребер. Розріджений граф не є зв'язковим, оскільки кожна його вершина з'єднана тільки з невеликою кількістю інших вершин; насичений граф, безсумнівно, є зв'язковим, тому що кожна його вершина пов'язана в середньому з 20 іншими вершинами. Ці діаграми демонструють складність розробки алгоритмів креслення довільних графів (на рисунку вершини розміщені в випадково вибраних місцях).

Розглянемо детальніше реалізацію створення випадкових ребер. Для заданої кількості вершин  $V$  генеруються довільні ребра, тобто пари випадкових чисел від  $0$  до  $V-1$ . Результатом, швидше за все, буде довільний мультиграф з петлями. Будь-яка пара може містити два однакових числа (тобто можливі петлі); і будь-яка пара може повторитися кілька разів (тобто можливі паралельні ребра). Програма генерує ребра до тих пір, поки не набереться  $E$  ребер; рішення про видалення паралельних ребер залишається за реалізацією. Якщо видаляти паралельні ребра, то в насичених графах кількість генеруються ребер буде значно більше, ніж кількість використаних ребер ( $E$ ); тому даний метод зазвичай використовується для розріджених графів.

Необхідно зазначити, що графи, які моделюють карти, електронні схеми, розкладу, транзакції, мережі та інші реальні ситуації, зазвичай є не тільки розрідженими, а й локальними – ймовірність того, що задана вершина пов'язана з однією з вершин конкретного безлічі вершин, вище, ніж з іншими вершинами. Як показують такі приклади, існує багато різних способів моделювання локальності. Наприклад, випадкові графи з сусідніми зв'язками.

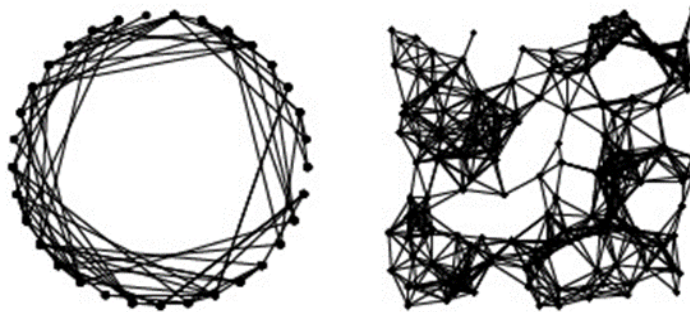


Рисунок 4.2. Випадкові графи з сусідніми зв'язками

Тут наведені приклади двох моделей розріджених графів. Граф з сусідніми зв'язками у верхній частині рисунку містить 33 вершини і 99 ребер, і кожне ребро

може з'єднувати одну вершину з іншого, якщо їх індекси відрізняються не більше ніж на 10 (по модулю  $V$ ). Евкліда граф з сусідніми зв'язками в нижній частині рисунку моделює графи, які зустрічаються в додатках, де вершини прив'язані до конкретних геометричних точкам. Для вершин обрані випадкові точки на площині, а ребра з'єднують будь-яку пару вершин, відстань між якими не перевищує  $d$ .

Цей граф відноситься до категорії розріджених (177 вершин і 1001 ребро). Змінюючи значення  $d$ , можна побудувати граф будь-якого ступеня насиченості.

Граф, показаний в нижній частині рис. 4.2, викреслений генератором, який вибирає на площині  $V$  точок з випадковими координатами від 0 до 1, а потім генерує ребра, що з'єднують будь-які дві точки, відстань між якими не перевищує  $d$ . Це – Евкліда граф з сусідніми зв'язками. Якщо  $d$  невелика, то граф виходить розрідженим, а якщо  $d$  велике, то граф насичений. Такий граф моделює графи, з якими ми стикаємося при роботі з картами, електронними схемами або іншими додатками, де вершини прив'язані до певних геометричних точкам. Їх неважко уявити наочно, вони дозволяють наочно побачити властивості алгоритмів, характерні для подібних додатків.

Один з можливих недоліків цієї моделі полягає в тому, що розріджені графи за просто можуть виявитися незв'язними; ще одна складність полягає в низькій вірогідності появи вершин високого ступеня, а також у відсутності довгих ребер. При бажанні можна ввести в ці моделі зміни для усунення цих недоліків.

Розглянемо детальніше граф транзакцій. Такі послідовності пар чисел можуть представляти список телефонних викликів в місцевій телефонній станції, або фінансові операції між рахунками, або будь-яку подібну ситуацію, якщо виконуються транзакції між двома елементами, яким присвоєно унікальні ідентифікатори. Такі графи не можна розглядати як чисто випадкові, адже деякі

телефонні номери використовуються набагато частіше, ніж інші, а деякі рахунки набагато активніші, ніж інші.

На рис. 4.3 показаний всього лише невеликий фрагмент графа, який можна виявити в комп'ютерах телефонної компанії.

900-435-5100	201-332-4562
415-345-3030	757-995-5030
757-310-4313	201-332-4562
747-511-4562	609-445-3260
900-332-3162	212-435-3562
617-945-2152	408-310-4150
757-995-5030	757-310-4313
212-435-3562	803-568-8358
913-410-3262	212-435-3562
401-212-4152	907-618-9999
201-232-2422	415-345-3120
913-495-1030	802-935-5112
609-445-3260	415-345-3120
201-310-3100	415-345-3120
408-310-4150	802-935-5113
708-332-4353	803-777-5834
413-332-3562	905-828-8089
815-895-8155	208-971-0020
802-935-5115	408-310-4150
708-410-5032	212-435-3562
201-332-4562	408-310-4150
815-511-3032	201-332-4562
301-292-3162	505-709-8080
617-833-2425	208-907-9098
800-934-5030	408-310-4150
408-982-3100	201-332-4562
415-345-3120	905-569-1313
413-435-4313	415-345-3120
747-232-8323	408-310-4150
802-995-1115	908-922-2239

Рисунок 4.3. Граф транзакцій

У цьому графі кожному телефонному номеру відповідає вершина, а кожне ребро, що з'єднує пару  $i$  і  $j$ , відповідає телефонним дзвінком від  $i$  до  $j$  протягом деякого фіксованого проміжку часу. Це безліч ребер є мультиграф величезних розмірів. Він, природно, розріджене, оскільки кожен абонент дзвонить лише в

мізерну частину всіх доступних телефонів. Цей граф характерний і для багатьох інших додатків. Аналогічну інформацію може, наприклад, містити кредитна картка фінансової установи і записи кредитної історії.

Вивчення продуктивності алгоритмів на графах, згенерованих на базі однієї з розглянутих імовірнісних моделей, не завжди дає досить точну інформацію для передбачення продуктивності на реальних графах. Однак генератори графів, які були розглянуті в даному пункті, зручні для тестування реалізацій і попередньої оцінки продуктивності алгоритмів. Перш ніж прогнозувати ефективність програми, необхідно хоча б перевірити правильність усіх припущень про взаємини між різними даними додатка з усіма використовуваними моделями або вибірками. Така перевірка доцільна при роботі в будь-якій прикладній області і вже тим більше важлива при обробці графів в силу великої різноманітності можливих видів графів.

## **4.2 Еволюційні алгоритми**

Відомо, що в області еволюційних алгоритмів можна виділити два напрямки: теоретичний та експериментальний.

Теоретичні дослідження еволюційних алгоритмів спрямовані в основному на побудову математичних моделей, виходячи з яких видаються рекомендації з вибору тих чи інших обчислювальних схем та налаштування внутрішніх параметрів, а також обчислюються оцінки точності і швидкості роботи алгоритмів.

В рамках експериментального напрямку розробляються алгоритми, призначені для розв'язання прикладних задач. Особливе місце тут займають методи гібридизації еволюційних алгоритмів, методів локального пошуку і алгоритмів, розроблених для розв'язання конкретного типу задач з урахуванням їх

специфіки. Питання про порівняння алгоритмів та налаштування параметрів вирішуються шляхом обчислювальних експериментів. Важливу роль тут відіграє створення загальнодоступних бібліотек тестових задач, розміщених у мережі Інтернет, які дозволяють досліднику порівнювати свої результати з роботами інших авторів. Для прикладу можна навести наступні ресурси:

- 1) DIMACS Challenge II (<http://dimacs.rutgers.edu>);
- 2) бібліотеку OR Library (<http://mscmga.ms.ic.ac.uk/infoMtm1>);
- 3) бібліотеку тестових задач Інституту математики ім. С.П. Соболева (<http://math.nsc.ru/AP/benchmarks/>).

#### **4.3 Бібліотека тестових задач, що була використана для дослідження ефективності еволюційних методів**

Для формування вихідних даних задачі розроблено генератор, який дозволяє за допомогою датчика псевдовипадкових чисел створювати матрицю транспортних витрат  $\{g_{ij}\}$  і матрицю потреб  $\{p_{ij}\}$  розмірності  $100 \times 100$ . Послідовність псевдовипадкових чисел створюється за допомогою кодового числа, винесеного на панель генератора. Сформовані матриці можуть бути записані у вигляді текстового файлу.

Для кожного підприємства в файл записуються відповідний рядок матриць  $\{p_{ij}\}$  і  $\{g_{ij}\}$ . Рядки, що відповідають різним підприємствам, поділяються символом нового рядка.

Транспортні витрати  $\{g_{ij}\}$  вибираються з інтервалу  $[1 \sim 100]$ . Величини  $\{p_{ij}\}$  вибираються з множини  $\{1,2,3,4,5\}$ . Витрати на відкриття підприємств однакові і рівні  $100$ .

Представимо три задачі: найпростішу задачу розміщення, задачу розміщення з обмеженнями потужності підприємств та багатостадійну задачу розміщення (додаток Г).

В свою чергу всі задачі розміщення можуть відрізнятися між собою способом породження матриці транспортних витрат. Вони розбиті на наступні класи:

1. Приклади на досконалих кодах (PCodes).
2. Приклади на шахівниці (Chess).
3. Приклади на кінцевих проектних площинах (FPP).
4. Приклади з великим розривом подвійності (Gap -A, Gap -B, Gap -C).
5. Приклади з рівномірним розподілом (Uniform).
6. Приклади на евклідової площини (Eucl).

Розглянемо *найпростішу задачу розміщення* виробництва. Для всіх класів  $|I| = |J|$ , вартість відкриття будь-якого підприємства дорівнює 3000 і матриці транспортних витрат складені таким чином, щоб число підприємств, що відкриваються, в оптимальному рішенні не сильно змінювалося при переході від одного класу до іншого. Класи **PCodes** і **Chess** характерні тим, що для них число строгих локальних мінімумів щодо околиці *додати - видалити* - зростає експоненціально з ростом розмірності задачі. Клас **FPP** є поліноміально вирішуваним. Класи **Gap-A**, **Gap-B**, **Gap-C** мають великий розрив подвійності, а класи **Uniform** і **Eucl** найбільш популярні при побудові наближених алгоритмів з гарантованими оцінками точності. У табл. 4.1 наводяться усереднені характеристики поведінки точного методу гілок і меж на даних класах.

Таблиця 4.1 – Характеристики поведінки точного методу гілок і меж на даних класах

Класи	Число ітерацій	Номер ітерації, на якій знайдено рішення	Кількість змін рекорду	Час роботи ( чч: мм: сс )
PCodes	374 264	371 646	11	00:00:15
Chess	138 674	136 236	17	00:00:06
FPP	6 656 713	6 652 295	27	00:05:20
Gap -A	10 105 775	3 280 342	10	00:04:52
Gap -B	30 202 621	14 656 960	22	00:12:24
Gap -C	541 320 830	323 594 521	30	1:42:51
Uniform	9 834	2748	8	00:00:00
Eucl	1 084	552	5	00:00:00

Розрахований час приведено для PC Pentium -IV, 1 800 MHz.

Для прикладів з кожного класу проведено наступний експеримент. Випадковим образом з рівномірним розподілом на всьому просторі допустимих рішень генерується 9000 початкових точок. Для кожної з них застосовується стандартна процедура локального спуску щодо околиці *додати - видалити – замінити*.

У підсумку отримаємо множину локальних оптимумів. Для неї пораховано число різних локальних оптимумів (**N1**) і нижня оцінка на діаметр області, займаної локальними оптимуму (**N2**). Крім того для кожного локального оптимуму підраховано число інших локальних оптимумів, що знаходяться від даного на відстані не більше  $r = 1, 2, 3, \dots$ . Ця величина (**Nr**) є середньою для всіх локальних оптимумів. У таблиці 4.2 наведено дані про середні значення цих показників для зазначених класів вихідних даних.

Таблиця 4.2 – Середній час, необхідний для пошуку локального оптимуму для різних класів задач

Class	PCodes	Chess	FPP	Gap -A	Gap -B	Gap -C	Uniform	Eucl
N1	8868	8009	8987	6022	8131	8465	1018	40
N2	55	50	51	36	42	41	33	21
Nr , r = 10	3	3	1,1	53	18	14	31	13

На рис. 4.4 наведено результати експерименту для  $r = 1, \dots, 100$ . Вони свідчать про те, що дані класи помітно відрізняються один від одного не тільки за кількістю локальних оптимумів, а й за середньою щільністю, і слід очікувати, що клас **Eucl** виявиться найбільш простим для алгоритмів локального пошуку, а класи **PCodes**, **FPP** і **Gap -C** - найбільш важкими.

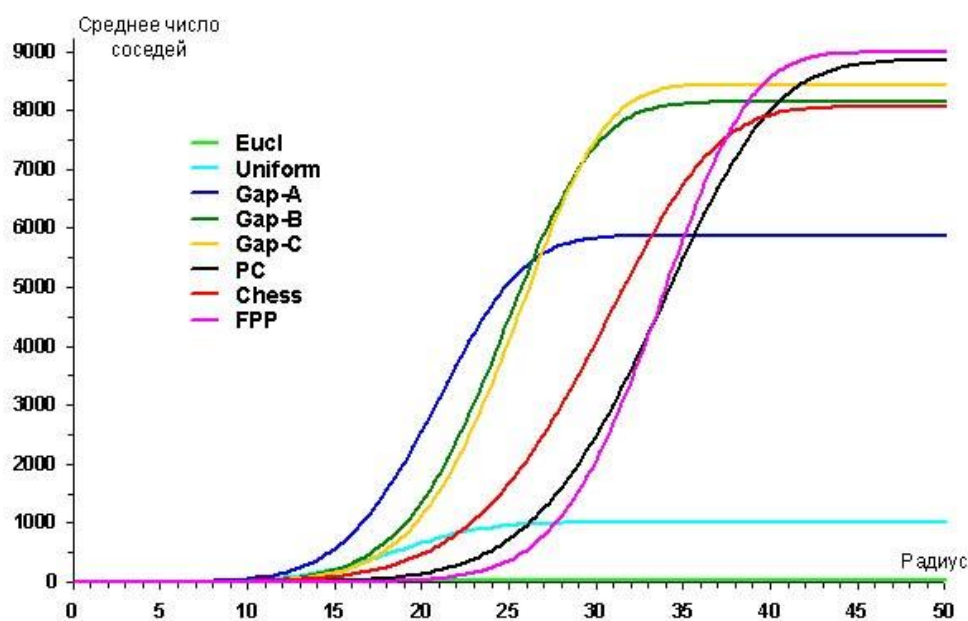


Рис. 4.4. Результати експерименту пошуку локального екстремуму

Перейдемо до задачі розміщення виробництва з обмеженнями потужності підприємства. Розглядається 5 класів вихідних даних. Представники цих класів

мають однакові матриці  $\{p_{ij}\}$  і  $\{g_{ij}\}$  і відрізняються тільки обмеженнями на обсяги виробництва. Обмеження для всіх підприємств в кожному класі однакові і рівні 10, 20, 30, 40 і 50 умовних одиниць продукції відповідно.

Результати тестових розрахунків наведені в таблицях додатку Г. Вихідні дані у вигляді текстових файлів доступні в першому стовпчику таблиці. Оптимальні значення цільової функції отримані І.Л. Васильєвим (Іркутськ).

Розглянемо *багатостадійну задачу розміщення виробництва*. Один з найбільш важких класів вихідних даних може бути побудований за аналогією з **класом Gap -A** для найпростішої задачі розміщення. Специфіка цього класу полягає в тому, що кожен споживач може обслуговуватися однією з 10 технологічних ланцюжків. До складу кожного ланцюжка входить рівно 4 підприємства. Початкові витрати на відкриття кожного підприємства дорівнюють 3000. Розмірність тестових прикладів є наступною: 50 підприємств, 100 технологічних ланцюжків, 100 споживачів.

У табл. 4.3 наведено вихідні дані та результати розрахунків для 30 тестових прикладів. У першому стовпчику таблиці дані стартові числа (коди) для датчика псевдовипадкових чисел, що дозволяють формувати дані приклади. Другий стовпець містить оптимальні значення цільової функції, отримані методом гілок і меж. У третьому стовпці наведено оцінки розриву подвійності. В останньому стовпчику буде подано оптимальні рішення (номера відкриваються підприємств).

#### 4.4 Алгоритм мурашиної колонії

Алгоритм мурашиної колонії – це один з ефективних поліноміальних алгоритмів для розв’язання задач на графах, в тому числі, задачі класифікації. Програма, яка була створена для реалізації мурашиного алгоритму з

Таблиця 4.3 – Результати тестування тестового прикладу багатостадійної задачі розміщення

код	оптимум	Розрив подвійності (%)	Номери підприємств, що відкриваються
471	69122	57,0	3, 4, 7, 8, 10, 13, 17, 21, 22, 23, 24, 26, 27, 30, 31, 32, 36, 37, 38, 40, 46, 48, 49
472	69140	55,9	1, 2, 3, 5, 7, 15, 16, 18, 19, 20, 22, 23, 26, 31, 32, 37, 38, 39, 41, 45, 46, 47, 49
473	72110	58,2	1, 4, 7, 10, 11, 17, 18, 23, 24, 25, 27, 29, 30, 31, 32, 34, 35, 36, 38, 39, 40, 43, 46, 48
474	69119	56,5	2, 3, 6, 9, 13, 14, 17, 19, 21, 22, 26, 27, 28, 32, 34, 40, 41, 44, 45, 46, 47, 48, 49
475	69121	56,5	3, 6, 7, 8, 9, 14, 15, 17, 20, 22, 23, 25, 29, 30, 32, 33, 34, 35, 37, 39, 40, 42, 45
476	69141	57,0	5, 11, 13, 17, 18, 19, 24, 26, 28, 30, 31, 32, 33, 37, 39, 40, 41, 43, 44, 45, 47, 49, 50
477	66109	54,7	3, 4, 7, 9, 14, 15, 16, 20, 21, 22, 23, 24, 32, 34, 36, 37, 40, 43, 45, 46, 47, 49
478	72104	59,0	1, 3, 5, 6, 9, 12, 14, 16, 17, 18, 19, 20, 23, 24, 27, 31, 32, 33, 37, 39, 42, 44, 45, 49
479	69123	56,5	2, 3, 6, 8, 11, 13, 15, 16, 18, 23, 25, 29, 30, 34, 37, 39, 40, 41, 42, 43, 44, 46, 50
480	72132	58,1	6, 7, 9, 13, 14, 17, 18, 19, 20, 21, 24, 28, 29, 33, 34, 36, 37, 38, 40, 41, 42, 44, 47, 49
481	69132	55,4	1, 4, 7, 8, 10, 11, 13, 16, 20, 21, 23, 24, 25, 30, 31, 33, 34, 36, 43, 44, 45, 46, 49
482	69141	55,6	3, 5, 6, 7, 13, 15, 16, 17, 20, 22, 23, 27, 30, 31, 36, 38, 39, 41, 42, 44, 45, 47, 50
483	72139	57,5	4, 7, 8, 9, 11, 16, 18, 20, 23, 24, 27, 28, 31, 33, 34, 35, 36, 37, 42, 43, 44, 45, 47, 48
484	72131	56,9	1, 2, 4, 7, 8, 10, 16, 18, 19, 21, 22, 24, 26, 27, 30, 32, 33, 35, 36, 39, 42, 44, 45, 49
485	75105	60,1	1, 2, 9, 10, 13, 16, 17, 18, 19, 20, 22, 24, 25, 27, 28, 29, 32, 33, 39, 40, 41, 42, 43, 44, 48
486	72121	57,0	1, 3, 8, 10, 11, 12, 18, 20, 21, 22, 23, 24, 26, 29, 30, 31, 32, 34, 36, 39, 41, 43, 49, 50
487	69138	55,6	3, 4, 7, 8, 9, 11, 12, 15, 18, 20, 21, 24, 27, 30, 32, 33, 34, 36, 37, 39, 43, 47, 50
488	69137	56,3	3, 4, 7, 8, 9, 11, 12, 15, 18, 20, 21, 24, 27, 30, 32, 33, 34, 36, 37, 39, 43, 47, 50
489	69107	56,2	1, 2, 3, 4, 5, 6, 11, 12, 13, 15, 16, 17, 23, 24, 27, 30, 31, 32, 34, 35, 36, 42, 46

код	оптимум	Розрив подвійності (%)	Номери підприємств, що відкриваються
490	72112	58,1	2, 7, 10, 11, 14, 16, 18, 19, 22, 24, 26, 31, 32, 33, 34, 40, 41, 42, 44, 45, 46, 47, 49, 50
491	72133	58,2	3, 4, 5, 6, 7, 11, 17, 19, 20, 21, 23, 25, 26, 28, 31, 32, 33, 35, 36, 38, 39, 41, 44, 47
492	69112	53,7	7, 9, 11, 12, 15, 16, 19, 21, 22, 23, 26, 29, 31, 34, 35, 37, 40, 41, 44, 47, 48, 49, 50
493	72137	57,8	4, 5, 6, 7, 10, 12, 13, 16, 18, 19, 20, 21, 22, 23, 24, 27, 29, 31, 33, 35, 36, 37, 39, 46
494	72127	58,1	1, 2, 3, 4, 8, 9, 12, 17, 20, 22, 23, 24, 30, 31, 34, 35, 36, 40, 42, 43, 45, 46, 47, 48
495	69142	56,3	1, 2, 5, 7, 8, 13, 14, 16, 17, 19, 21, 23, 24, 25, 27, 29, 31, 32, 35, 38, 39, 49, 50
496	69122	55,9	1, 3, 4, 5, 8, 14, 15, 18, 21, 22, 25, 26, 28, 29, 30, 32, 33, 34, 38, 42, 45, 46, 48
497	66138	53,4	3, 7, 12, 13, 15, 16, 17, 19, 21, 22, 27, 32, 36, 38, 39, 40, 41, 42, 43, 46, 47, 50
498	72112	56,6	1, 4, 6, 7, 8, 11, 14, 17, 23, 26, 28, 32, 33, 34, 35, 38, 39, 41, 43, 44, 46, 47, 48, 49
499	69135	55,6	3, 5, 7, 8, 9, 10, 11, 13, 15, 17, 20, 21, 24, 25, 26, 28, 29, 30, 33, 34, 40, 42, 44
500	69136	55,9	3, 5, 6, 8, 9, 13, 14, 19, 21, 22, 23, 26, 28, 32, 34, 37, 38, 40, 43, 45, 46, 48, 49

метафорою агрегацією феромонів для вирішення задачі класифікації, була написана на мові C++, та складається з таких структурних елементів:

1) об'єкт AntAlg() – головний об'єкт програми, ініціалізується наступними параметрами:

1. n\_alg – кількість вимірів цільової функції,
2. k\_alg – кількість рішень,
3. k\_del – параметр, який визначає кількість нових розв'язків, що виникають на кожній ітерації,
4. q\_alg – параметр алгоритму,
5. xi – параметр алгоритму,
6. max\_count – максимальна кількість ітерацій;

- 2) `struct component { vector args` – масив змінних;
- 3) `double func` – значення цільової функції;
- 4) `double weight` – вага цього розв'язку;
- 5) `double posib` – ймовірність вибору цього розв'язку; };
- 6) `vector T_mas` – масив, який представляє собою сукупність усіх розв'язків, що існують на даний момент;
- 7) метод `gaussian_random (double mue, double sigma)` – відповідно до нормального розподілу щільності ймовірностей (функція Гауса) обирає випадкове число типу `double`, параметрами цього методу є `mue` – математичне очікування, `sigma` - стандартне відхилення;
- 8) метод `gauss_func (double mue, double sigma, double x)` – повертає розраховану функцію Гауса, де `mue` – математичне очікування, `sigma` - стандартне відхилення;
- 9) метод `make_T(double hypercube_a, double hypercube_b, int func_numb)` – створює початкову конфігурацію алгоритму, тобто заповнює масив `T_mas` таким чином, що присвоює змінним кожного набору випадкові значення в межах гіперкубу між визначеними параметрами (`hypercube_a`, `hypercube_b`), також першочергово обчислює задану параметром `func_numb` цільову функцію для кожного набору і сортує рішення;
- 10) метод `make_weight()` – обчислює ваги всіх знайдених розв'язків згідно за формулою;
- 11) метод `random(double min, double max)` – генерує випадкове значення типу `double` в інтервалі (`min`, `max`);
- 12) метод `T_sort()` – сортує масив розв'язків згідно значень цільової функції;
- 13) метод `make_choice()` – виконує відбір рішення для формування нового набору згідно з вище означеним алгоритмом;
- 14) метод `make_sigma(double xi, int l, int i)` – формує стандартне відхилення для функції `gaussian_random(double mue, double sigma)` за вище означеною формулою;

15) метод `run()` – запускає алгоритм.

#### 4.5 Порівняння ефективності використаних алгоритмів

Метаевристичні алгоритми передусім відрізняються за збіжністю та точністю. Крім того, їх притаманні різні ступені участі ОПР, рівень формалізації, трудомісткість та час роботи програми за умов реалізації алгоритму в машинному коді.

Ефективність метаевристичних алгоритмів визначається ступенем виконання вимог до показників точності і обчислювальних витрат. Точність характеризується похибкою отриманого результату:

$$\delta(X^*) = (F(X^*) - F(X_{exact}))/F(X^*), \quad (4.1)$$

де  $X^*$  – рішення, отримане за допомогою метаевристичного алгоритму;

$F(X_{exact})$  – точне значення екстремуму;

$F(X^*)$  – значення цільової функції в точці. Оскільки в практично вирішуваних завданнях точка невідома, оцінку точності різних алгоритмів виконують на тестових завданнях і в якості приймають найкраще з відомих рішень.

Збіжність метаевристичних алгоритмів гарантується відомою теоремою шим (теоремою шаблонів, the schema theorem [15]). Однак теорема шим лише в якійсь мірі виправдовує застосування таких алгоритмів, але не дає ніяких чисельних оцінок збіжності. Тому розумним видається порівняння роботи метаевристичних алгоритмів з іншими алгоритмами. Звичайно, хотілося б отримати чисельні оцінки збіжності алгоритму, порівняння швидкості і точності одержуваного розв'язку. На жаль, для задач, в яких виправдане застосування метаевристичного алгоритму, це не завжди представляється можливим.

Оцінка якості метаевристичного алгоритму дається порівнянням результатів роботи цього алгоритму з іншими алгоритмами на досить великій серії задач.

Порівняння алгоритмів здійснювалася за наступними напрямками:

1) порівняльна якість – число, що показує скільки разів в серії задач результат, отриманий за цим типом алгоритмів, був не гірше результатів, отриманих за іншими типами;

2) рейтинг за правилом Борда – сума балів, набраних алгоритмом по всіх задачах вибірки. За перше місце алгоритм отримує 2 бали, за друге – 1 бал, за третє – 0 балів.

Але варто зазначити, що порівняти ефективність двох алгоритмів є досить важким завданням. На конкретній проблемі різні алгоритми можуть показувати різні результати, але на множині всіх задач вони нерозрізнені. Це розповсюджується на властивість алгоритмів платити за першість у розв'язку одних задач значно гіршими результатами на інших задачах. В такому сенсі, у пошуку немає безкоштовних сніданків. З іншого боку, згідно з Шаффером, ефективність пошуку зберігається. Зазвичай пошук інтерпретується як оптимізація, і це веде до спостереження, що безкоштовних сніданків немає також в оптимізації.

«Теорема про відсутність безкоштовних сніданків Вульперта та Маккріді» [101], як дослівно зазначено самими Вульпертом і Маккріді, полягає в тому, що «ефективність будь-яких двох алгоритмів у середньому однакова при розв'язанні всіх можливих задач». Наслідки відсутності безкоштовного сніданку свідчать про те, що підбір специфічного алгоритму для кожної проблеми дає в середньому кращі результати, ніж застосування одного й того самого алгоритму до всієї множини задач. Айгель, Гуссон та Інглиш уклали спільну угоду про відсутність безкоштовного сніданку взагалі. Вважаючи, що фізично це можливо, теорема не

завжди виконується точно. Дросте, Джансес та Вегенер довели теорему, яку вони інтерпретують як «Безкоштовних сніданків (майже) не існує», на практиці.

Для конкретизації суті можна розглянути оптимізатора-виконавця, якому була поставлена задача. Маючи деяку інформацію про виникнення цієї проблеми, оптимізатор може використовувати ці знання для вибору достатньо продуктивного алгоритму для розв'язання задачі. Якщо виконавець не може застосувати ці знання при виборі алгоритму, або взагалі не має таких знань, перед ним постає питання, чи існують алгоритми, які загалом показують найкращі результати при розв'язанні реальних задач. Автори теореми про (майже) відсутність безкоштовних сніданків дають негативну по суті відповідь на це питання, але залишають місце для деяких резервацій, зважаючи на те, наскільки теорема застосована до виконавця.

*Універсальний майже загальний оптимізатор теоретично можливий.* Будь-який алгоритм пошуку виявляє добрі результати майже на всіх цільових функціях.

*Алгоритм може перевершити інший у випадку, коли жоден з них не спеціалізується на задачі.* Можливий варіант, коли обидва алгоритми найгірші для розв'язання проблеми. Вульперт та Маккріді розробили систему оцінки «збігу» між алгоритмом та задачею. Сказати, що один алгоритм кращий за інший у розв'язанні задачі — це не обов'язково має на увазі, що один з алгоритмів спеціалізується на цій задачі.

*На практиці деякі алгоритми переоцінюють допустимі розв'язки.* Перевага алгоритму, що ніколи не переоцінює розв'язки, над тим, що переоцінює, не обов'язково означає спеціалізацію одного з алгоритмів на проблемі.

*Для майже всіх цільових функцій, спеціалізація цілком випадкова.* Через Колмогорівську складність, цільові функції не мають постійності, яку міг би використати алгоритм. Маючи нестисливу цільову функцію, нема підстав для вибору певного алгоритму. Якщо обраний алгоритм більшість часу показує добру

ефективність — це просто збіг.

На практиці, тільки значно стисні (далекі від випадковості) функції вміщуються в пам'ять комп'ютера, і немає випадку, коли один алгоритм видає добрі результати на всіх цільових функціях. Є загальна перевага у закладанні знань про проблему в логіку алгоритму. В той час, коли результати NFL встановлюють, у звуженому сенсі, теореми повної зайнятості для професіоналів в оптимізації, важливо не приймати цей термін буквально. По-перше, люди зазвичай мають недостатньо знань для роботи. По-друге, привнесення початкових знань не дає значної переваги для деяких задач. Нарешті, людський час значно дорожче комп'ютерного часу. Існує багато випадків, коли компанія обирає повільну оптимізацію за допомогою цілком комп'ютерної програми, а не швидко оптимізацію, яка залучає людей. Наслідки NFL не означають, що не варто здійснювати «постріли всліпу» при розв'язку задач з неспеціалізованими алгоритмами. Ніхто не виявив долю задач, в якій алгоритм зможе знайти гарне рішення швидко. Також існує практично безкоштовний сніданок, що не конфліктує з загальною теорією. Виконання реалізації алгоритму на комп'ютері значно дешевше людської праці, навіть заради гарного розв'язку. Якщо алгоритм здатен знайти задовільне рішення за прийнятний проміжок часу — маленький вклад приніс значний прибуток. Якщо ж розв'язок не знайдено — понесені втрати

Для оцінки ефективності метаевристик на фрагментарних структурах було розроблено програму оцінки ефективності для різних модельних задач, в якій для багатьох дискретних задач, що допускають фрагментарну модель, були реалізовані універсальні алгоритми ряду метаевристик.

Зокрема, реалізовані метод випадкового пошуку, метод ітеративного локального пошуку, метод імітації відпалу, еволюційно-фрагментарний алгоритм, метод перемішаних стрибаючих жаб (інтерфейс програми представлено на рис 4.5).

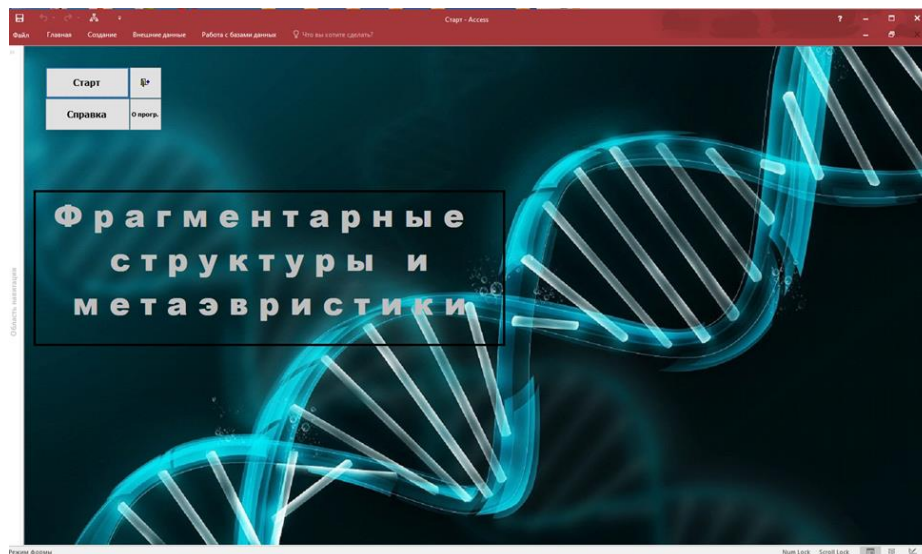


Рис. 4.5. Интерфейс програми оцінки якості метаевристик

У програмі реалізовані генератор випадкових завдань з різними обмеженнями, база даних завдань і програма порівняння ефективності алгоритмів.

Для задачі покриття графа зірками проведено чисельний експеримент на базі 53 випадково згенерованих завдань. Розглядалися зв'язкові графи з числом вершин від 20 до 50 і з щільністю ребер 0,5-0,8. Для кожної з задач було побудовано фрагментарну модель і застосовувалася група алгоритмів (випадковий пошук, еволюційно-фрагментарний алгоритм, метод імітації відпалу тощо). Параметри алгоритмів підбиралися таким чином, щоб трудомісткість обчислень була приблизно однаковою.

Результати розрахунків порівнювалися за такими правилами.

Правило 1. Порівнювати число завдань в серії, де алгоритм отримав перше місце з точки зору значення цільової функції.

Правило 2. Розраховувався рейтинг алгоритмів за правилом Борда. За останнє місце в результатах давалося 0 очок, за передостаннє 1 очко тощо. За перше місце давалося  $K-1$  очко, де  $K$  число алгоритмів, які брали участь в порівнянні. Далі кількості очок, набраних алгоритмом, підсумовувалися по всіх

завдань серії. Алгоритми впорядковувалися за спаданням рейтингу (рис. 4.6).

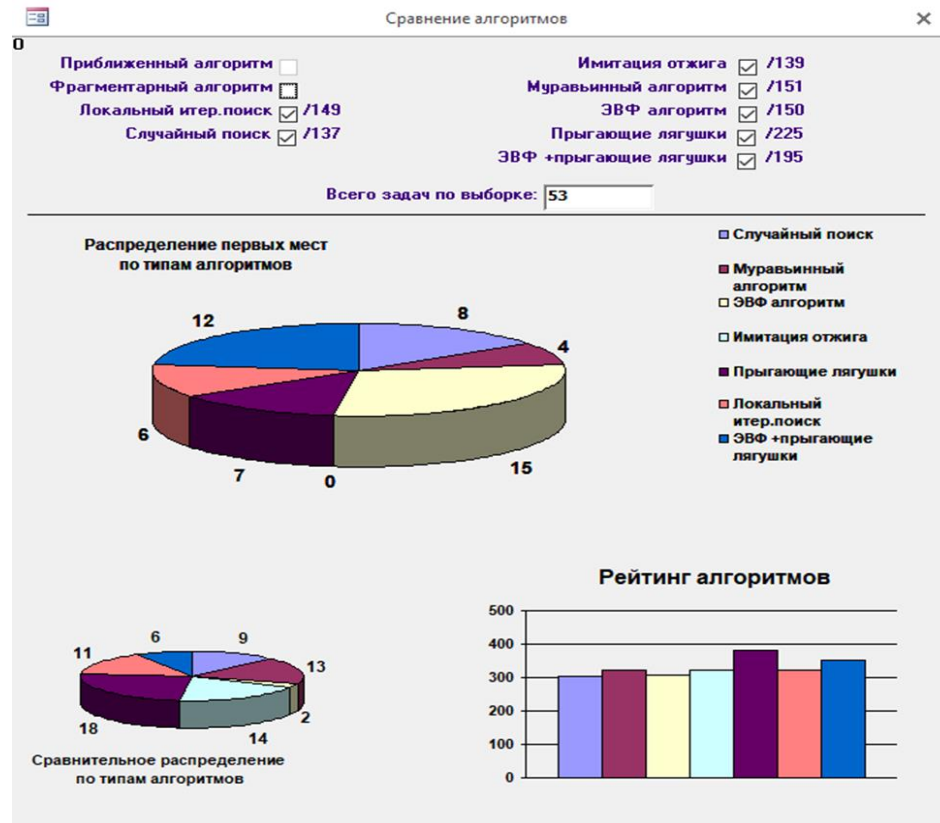


Рис. 4.6. Побудова рейтингу алгоритмів

Результати порівняння різних алгоритмів показують, що жоден з них не володіє явною перевагою перед іншими. Це побічно підтверджує відому теорему «про відсутність безкоштовних обідів» для метаевристик. Таким чином, в умовах реальної експлуатації розумно застосовувати не одну, а кілька метаевристик і вибирати найкращий результат. Розглянутий в дисертаційній роботі метод використання фрагментарних моделей для пошуку субоптимальних рішень задач класифікації, дозволяє порівняно просто побудувати універсальну комп'ютерну систему для таких завдань. Причому універсальними будуть програми реалізації метаевристик, а індивідуальними методи побудови фрагментарних моделей і алгоритми розрахунку значень критеріїв.

## Висновки до розділу 4

Ефективність алгоритмів була досліджена за допомогою згенерованих ймовірнісних графових моделей. В межах експериментального напрямку розробляються алгоритми на базі методів гібридизації еволюційних алгоритмів, методів локального пошуку і алгоритмів, розроблених для розв'язання конкретного типу задач з урахуванням їх специфіки. Порівняння алгоритмів та налаштування параметрів проводиться за допомогою обчислювальних експериментів.

При цьому однозначно зробити висновок про перевагу одного алгоритму над іншими зробити не можна, що підтверджує теорему «про відсутність безкоштовних обідів». Найкращим рішенням є використання фрагментарних моделей для пошуку субоптимальних рішень задач класифікації.

## ВИСНОВКИ

У дисертаційній роботі проведено узагальнення і розробку теоретичних основ математичного апарату для побудови та дослідження фрагментарних моделей та метаевристичних методів для розв'язку задачі класифікації. Розглянуті в дисертації моделі та методи пройшли широку апробацію. Вони можуть використовуватися під час пошуку розв'язків у багатьох наукових та практичних задачах, в тому числі, задачі розміщення виробництва.

У межах проведених досліджень отримано такі основні результати:

1. Проведений аналітичний огляд існуючих задач класифікації дозволив зробити висновок про те, що для розв'язку задачі класифікації найчастіше застосовуються методи: найближчого сусіда,  $K$ -найближчого сусіда; байєсовські мережі; індукція дерев рішень; нейронні мережі; метод опорних векторів; статистичні методи, зокрема, лінійна регресія; класифікація  $sbr$  – методом або за допомогою генетичних алгоритмів. На разі залишилося багато прикладних невирішених задач класифікації, зокрема, розпізнавання спаму електронної пошти, зображень та мовлення в якості цифрового паролю, опис мікроматриці ДНК тощо.

2. Аналіз особливостей задачі класифікації на дискретній множині виявив те, що задача оптимальної класифікації природним чином виникає при створенні систем автоматизації проектування, автоматизованих систем планування ресурсів, при розв'язанні задач логістики, штучного інтелекту, робототехніки тощо.

3. Встановлено зв'язок задачі класифікації із задачею покриття графу зірками та задачею розміщення виробництва. Показано, що ці задачі можуть бути сформульовані як задачі на множині з фрагментарною структурою. Для всіх задач розглянутих класів фрагментарний алгоритм пошуку допустимого розв'язку по

заданій перестановці елементарних фрагментів має поліноміальну трудомісткість. Таким чином, для всіх розглянутих в дисертації задач наближений розв'язок може бути знайденим стандартним еволюційно-фрагментарним алгоритмом, базова множина якого є множиною перестановок.

4. Виявлено, що задача оптимальної класифікації, як і більшість задач дискретної оптимізації є NP-важкою. Крім того, вона містить велику кількість змінних, які в свою чергу можуть мати невизначений (непрямий) характер. До того ж у прикладних задачах, як правило, є багато обмежень, які ускладнюють застосовність відомих алгоритмів. Тому для задач оптимальної класифікації є виправданим застосування метаевристик.

5. Виявлено, що при пошуку екстремуму справедливим є гіпотеза великої долини, що частково пояснює працездатність метаевристичних алгоритмів. Головною їх особливістю є наступна: якщо в популяції збираються локальні оптимуми, які відповідно до гіпотези сконцентровані в одному місці, і чергове рішення обирається десь між двома довільними локальними оптимумом, то такий процес має багато шансів знайти глобальний оптимум.

6. Розроблено фрагментарну модель для пошуку субоптимальних рішень задачі оптимальної класифікації, що дозволяє відшукати допустимий розв'язок і, відповідно, отримувати значення критерію на цьому розв'язку. Однак допустимий розв'язок, отриманий шляхом застосування фрагментарного алгоритму, взагалі кажучи, не є оптимальним. Для пошуку наближених оптимальних розв'язків подібних задач запропоновано модель, що поєднує метаевристичні та генетичні алгоритми.

7. Доведено властивість досяжності для розроблених фрагментарних моделей, оскільки результатом роботи фрагментарного алгоритму є допустиме покриття графу зірками з можливістю вибору нумерації.

8. Виявлено, що ефективність алгоритмів може бути досліджена за

допомогою згенерованих ймовірнісних графових моделей. В межах експериментального напрямку розробляються алгоритми на базі методів гібридизації еволюційних алгоритмів, методів локального пошуку і алгоритмів, розроблених для розв'язання конкретного типу задач з урахуванням їх специфіки.

9. Розроблено програмну реалізацію генерації випадкових графів та розв'язку задачі класифікації за допомогою метаевристичних алгоритмів. Порівняння алгоритмів та налаштування параметрів проводилося за допомогою обчислювальних експериментів.

10. Проведені математичні експерименти для оцінки якості метаевристичних алгоритмів не дають можливість однозначно зробити висновок про перевагу одного алгоритму над іншими, що підтверджує теорему «про відсутність безкоштовних обідів». Найкращим рішенням є використання фрагментарних моделей для пошуку субоптимальних рішень задач класифікації.

## ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Айвазян С. А., Бухштабер В. М., Енюков И. С., Мешалкин Л. Д. Прикладная статистика: Классификация и снижение размерности. Справочное издание / Под ред. С. А. Айвазяна. – М. : Финансы и статистика, 1989. – 607 с.
2. Айвазян С. А. Классификация многомерных наблюдений / Айвазян С. А., Бежаева З. И., Староверов О. В. – М., 1974. – 240 с.
3. Алескеров Ф. Т. Бинарные отношения, графы и коллективные решения / Ф. Т. Алескеров, Э. Л. Хабина, Д. А. Шварц. – М. : ГУ ВШЭ, 2006. – 300 с.
4. Алефельд Г., Херцбергер Ю. Введение в интервальные вычисления. – М. : Мир, 1987. – 356 с.
5. Алтунин А. Е., Семухин М. В. Модели и алгоритмы принятия решений в нечетких условиях. – Тюмень: Издательство Тюменский Государственный Университет, 2000. – 352 с.
6. Ахо А., Хопкрофт Дж. Построение и анализ вычислительных алгоритмов. – М. : Мир, 1979. – 536 с.
7. Ащепков Л. Т., Косогорова И. Б. Минимизация квадратичной функции с интервальными коэффициентами // Журнал вычислительной математики и математической физики. – 2002. – Т. 42, № 5. – С. 654–664.
8. Бабак О. В. Об одном подходе к оптимизации решения задач обучения распознаванию образов на основе метода опорных векторов // Кибернетика и системный анализ. – 2004. – № 2. – С. 179–185.
9. Бабак О. В., Гасанов А. С. Новый подход к решению задач кластеризации данных // Кибернетика и системный анализ. – 2001. – № 6. – С. 126–133.

10. Бабак О. В., Татаринев А. Э. Об одном подходе к решению задач классификации в условиях неполноты информации // Кибернетика и системный анализ. – 2005. – № 6. – С. 116–123.
11. Батищев Д. И. Генетические алгоритмы решения экстремальных задач / Д. И. Батищев. – Воронеж: Воронеж. техн. ун-т, 1995. – 65 с.
12. Баум Д., Коваленко И. Графовые модели коммуникации мобильных устройств в зонах доступа // Кибернетика и системный анализ. – 2003. – № 5. – С. 107–121.
13. Берж К. Теория графов и ее применения / К. Берж. – М.: Изд-во иностранной лит., 1962. – 320 с.
14. Борисовский П. А. Генетические алгоритмы для задачи о поставках продукции / П. А. Борисовский // Динамика систем, механизмов и машин: материалы V Междунар. науч.-техн. конф. – Омск: Изд-во ОмГТУ, 2004 – Кн. 2. – С. 255-258.
15. Гладков Л. А. Генетические алгоритмы / Л. А. Гладков, В. В. Курейчик, В. М. Курейчик. – М.: Физматлит, 2006. – 319 с.
16. Глибовец Н. Н., Медвидь С. А. Генетические алгоритмы и их использование для решения задачи составления расписания // Кибернетика и системный анализ. – 2003. – № 1. – С. 95-108.
17. Гуляницький Л. Ф. Застосування метаевристичних алгоритмів для розв'язання одного класу задач розміщення прямокутників на напівнескінченній стрічці / Л. Ф. Гуляницький, В. В. Турінський // System Analysis and Information Techns: Proc. 15-th Int. Conf., (May 27-31, 2013, Kyiv, Ukraine). – SAIT, 2013.
18. Гуляницький Л. Ф. Один підхід до синтезу генетичних алгоритмів та алгоритмів прискореного ймовірнісного моделювання / Л. Ф. Гуляницький, О. Я. Турчин // Вісник Нац. ун-ту «Львівська політехніка». Сер.: Комп'ютерні системи проектування. Теорія і практика. – 2003. – № 471. – С. 75-80.

19. Гэри М. Вычислительные машины и труднорешаемые задачи / М. Гэри, Д. Джонсон ; пер. с англ. – М.: Мир, 1982. – 416 с.
20. Донец А. Г. Об одном подходе к решению задачи Штейнера / А. Г. Донец // Математические машины и системы. – 1997. – № 2. – С. 41-42.
21. Донец Г. А. Задачи комбинаторного распознавания / А. Г. Донец // Проблемы теоретической кибернетики: материалы XVI Междунар. конф., 20-25 июня 2011 г. – Н. Новгород: Изд-во Нижегородского ун-та, 2011. – С. 142-144.
22. Донец Г. А. Экстремальные покрытия графов / Г. А. Донец, А. Я. Петренюк. – Кировоград: Комбінаторні конфігурації, 2009. – 172 с.
23. Донець Г. П. Екстремальні задачі на перестановках зі спеціальною генерацією / А. Г. Донец // Комбінаторні конфігурації та їх застосування: матеріали XI Міжвузівського наук.-практ. семінару – Кіровоград, 2011 – С. 41-48.
24. Дюк В. Data Mining : учебный курс / В. Дюк, А. Самойленко. – СПб. : Питер, 2001. – 368 с.
25. Емеличев В. А. Асимптотический подход к многокритериальной задаче покрытия графа звездами / В. А. Емеличев, В. А. Перепелица, Х. Д. Шунгаров // ДАН БССР. – 1986. – Т. XXXI, № 5. – С. 430-433.
26. Емеличев В. А. К вычислительной сложности дискретных многокритериальных задач / В. А. Емеличев, В. А. Перепелица // Известия АН СССР. Сер.: Техническая кибернетика. – 1988. – № 1. – С. 78-85.
27. Емельянов В. В. Теория и практика эволюционного моделирования / В. В. Емельянов, В. В. Курейчик, В. М. Курейчик. – М.: Физматлит, 2003. – 432 с.
28. Емец О. А. Задача цветной упаковки прямоугольников с учетом погрешностей исходных данных и ее решение / О. А. Емец, Л. Г. Евсеева, Н. Г. Романова // Экономика и математические методы. – 2000. –Т. 36, № 3. – С. 149-152.

29. Ємець О. О. Результати численних експериментів по розв'язуванню задачі кольорового упакування прямокутників з урахуванням похибок початкових даних / О. О. Ємець, Л. Г. Євсєєва, Н. Г. Романова // Вісник Полтавського державного педагогічного інституту ім. В. Г. Короленка. – Полтава, 1998. – Вип. 3. – С. 16-18.

30. Зайченко Ю. П. Исследование операций / Ю. П. Зайченко. – К.: Вища школа, 1988. – 552 с.

31. Заховалко Т. В. Задача покриття графа типовими подграфами с заданными свойствами симметрии / Т. В. Заховалко // Питання прикладної математики і математичного моделювання: зб. наук. праць. – Дніпропетровськ, 2005. – С. 87-89.

32. Перепелиця В. О., Козін І. В., Терещенко Е. В. Задачі класифікації: підходи, методи, алгоритми : монографія / Запоріжжя: Поліграф, 2008. – 188 с.

33. Козин И. В. Использование ЭВФ-алгоритмов для задачи прямоугольного раскроя / И. В. Козин, С. И. Полюга // Питання прикладної математики і математичного моделювання: зб. наук. праць. – Дніпропетровськ, 2009. – С. 199-208.

34. Козин И. В. Наследственные структуры в метрических пространствах / И. В. Козин, С. И. Полюга // Питання прикладної математики і математичного моделювання: зб. наук. праць. – Дніпропетровськ, 2012. – С. 166-177.

35. Козин И. В. О свойствах фрагментарных структур / И. В. Козин, С. И. Полюга // Вісник Запорізького національного університету. Сер.: Фізико-математичні науки. – 2012. – № 1. – С. 99-106.

36. Козин И. В. Фрагментарные модели для некоторых экстремальных задач на графах / И. В. Козин, С. И. Полюга // Математичні машини і системи. – 2014. – № 1. – С. 143-150.

37. Козин И. В. Фрагментарные структуры и эволюционные алгоритмы / И. В. Козин // Питання прикладної математики і математичного моделювання: зб. наук. праць. – Дніпропетровськ, 2008. – С. 138-146.
38. Козин И. В. Фрагментарный алгоритм для задачи симметричного размещения / И. В. Козин // Радиоэлектроника, информатика, управление. – 2005. – № 1. – С. 76-83.
39. Козин И. В. Эволюционная модель упаковки многомерных объектов / И. В. Козин, С. И. Полюга // Вісник Запорізького національного університету. Сер.: Фізико-математичні науки. – 2010. – № 1. – С. 61-67.
40. Козин И. В. Эволюционно-фрагментарная модель упаковки пентамино / И. В. Козин, С. И. Полюга // Дискретный анализ и исследование операций. – 2014. – Т. 21, № 6. – С. 35-50.
41. Козин И. В. Фрагментарные алгоритмы в системах поддержки принятия решений / И. В. Козин // Питання прикладної математики і математичного моделювання: зб. наук. праць. – Днепропетровск, 2006. – С. 131-137.
42. Кочетов Ю. А. Локальный поиск с чередующимися окрестностями / Ю. А. Кочетов, Н. Младенович, П. Хансен // Дискретный анализ и исследование операций. Сер. 2. – 2003. – Т. 10, № 1. – С. 11-43.
43. Курапов С. В. Векторная алгебра и рисунок графа / С. В. Курапов, В. В. Савин. – Запорожье: ЗГУ, 2003. – 200 с.
44. Курейчик В. М. Генетические алгоритмы / В. М. Курейчик. – Таганрог: изд-во ТРТУ, 1998. – 242 с.
45. Курейчик В. М. Генетические алгоритмы. Состояние. Проблемы. Перспективы / В. М. Курейчик // Известия РАН. Сер.: Теория и системы управления. – 1999. – № 1. – С. 144-160.

46. Максишко Н. К. Моделі та методи розв'язання прикладних задач покриття на графах та гіперграфах / Н. К. Максишко, Т. В. Заховалко. – Запоріжжя: Полиграф, 2009. – 244 с.

47. Мухачева А. С. Генетический алгоритм поиска минимума в задачах двумерного гильотинного раскроя / А. С. Мухачева, А. В. Чиглинец // Информационные технологии. – 2001. – № 3.– С. 27-32.

48. Мухачева Э. А. Генетический алгоритм блочной структуры в задачах двумерной упаковки / Э. А. Мухачева, А. С. Мухачева, А. В. Чиглинец // Информационные технологии. –1999. – № 11. – С. 12-17.

49. Панченко Т. В. Генетические алгоритмы: учебно-методическое пособие / Т. В. Панченко ; под ред. Ю. Ю. Тарасевича. – Астрахань: АГУ, 2007. – 87 с.

50. Перепелица В. А. Многокритериальные задачи теории графов / В. А. Перепелица. – К.: УМК, 1989. – 67 с.

51. Полюга С. И. Эволюционно-фрагментарная модель задачи регулярного покрытия взвешенного графа  $k$ -звездами / С. И. Полюга // Вісник Запорізького національного університету. Сер.: Фізико-математичні науки. – 2011. – № 2. – С. 105-110.

52. Пятецкий-Шапиро Г. Data Mining и перегрузка информацией / Г. Пятецкий-Шапиро // Анализ данных и процессов / А. А. Барсегян, М. С. Куприянов, И. И. Холод и др. – 3-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2009. – 512 с.

53. Рябенко А. Є. Математичні моделі та методи для векторних задач оптимізації організаційних структур та землекористування: Автореф. дис.... канд. ф.-м. наук: 01.05.02 / Дніпропетровський національний університет. – Дніпропетровськ, 2003. –18 с.

54. Сергеева Л. Н. Моделирование поведения экономических систем методами нелинейной динамики (теории хаоса). – Запорожье: ЗГУ, 2002. – 227 с.
55. Сергеева Л. Н. Моделирование структуры экономических систем и процессов. – Запорожье: ЗГУ, 2002. – 88 с.
56. Сергиенко И. В. Математические модели и методы решения задач дискретной оптимизации. – К. : Наук. думка, 1988. – 472 с.
57. Сергиенко И. В., Парасюк И. Н., Каспшицкая М. Ф. Об одной задаче многопараметрического выбора оптимальных решений // Кибернетика и системный анализ. – 2003. – №2. – С.3–15.
58. Сергиенко И. В., Шило В. П. Задачи дискретной оптимизации. Проблемы, методы решения, исследования. – Киев: Наук. думка, 2003. – 260 с.
59. Сигорский В. П. Математический аппарат инженера. – Киев: Техніка, 1977. – 768 с.
60. Скобцов Ю. О. Основы эволюционных вычислений: учеб. пособие / Ю. О. Скобцов. – Донецк: ДонНТУ, 2008. – 326 с.
61. Турчина В. А. Наближені алгоритми побудови оптимальних паралельних упорядкувань заданої довжини / В. А. Турчина, Н. К. Федоренко // Питання прикладної математики і математичного моделювання: зб. наук. праць. – Дніпропетровськ: ДНУ, 2010. – С. 312-319.
62. Чубукова И. А. Data Mining : учебное пособие / Чубукова И. А. – М. : Интернет-университет информационных технологий: БИНОМ: Лаборатория знаний, 2006. – 382 с.
63. Уилсон Р. Введение в теорию графов / Р. Уилсон. – М.: Мир, 1977. – 208 с.
64. Харари Ф. Теория графов / Ф. Харари. – 2-е изд., стер. – М.: Едиториал УРСС, 2003. – 296 с.

65. Шило В. П. Метод глобального равновесного поиска решения задачи о максимальном взвешенном разрезе графа / В. П. Шило, О. В. Шило, В. А. Рошин // Кибернетика и системный анализ. – 2012. – № 4. – С. 101-105.
66. Шиханович Ю. А. Введение в современную математику. – М. : Наука, 1965. – 376 с.
67. Au W-H., Keith C.C.C., Wong A.K.C., Wang Y. Attribute clustering for grouping, selection, and classification of gene expression data // IEEE/ACM Transactions on Computational Biology and Bioinformatics. – April-June 2005. – Vol. 2, № 2. – P. 83-101.
68. Chakrabarti S. Data mining for hypertext: a tutorial survey // ACM SIGKDD Explorations. – 2000. – Vol. 1, № 2. – P. 1-11.
69. Chakrabarti S. Mining the web: discovering knowledge from hypertext data. – San Francisco. Morgan Kaufman, 2002. – 352 p.
70. Deza Michel M., Deza Elena Encyclopedia of Distances. – Springer-Verlag Berlin Heidelberg, 2009.
71. Desai M., Spink A. An algorithm to cluster documents based on relevance // Information Processing and Management. – 2005. – Vol. 41, № 5. – P. 1035-1049.
72. Jain A. K. Data Clustering: A Review / A. K. Jain, M. N. Murty, P. J. Flunn // ACM Computing Surveys. – September 1999. – Vol. 31. – No. 3. – P. 265–323.
73. Halkidi M. On Clustering Validation Techniques / M. Halkidi, Y. Batistakis, M. Vazir-giannis // Journal of Intelligent Information Systems. – 2001. – 17:2/3. – P. 107–145.
74. Han J., Kamber M. Data mining: concepts and techniques. – San Francisco: Morgan Kaufman, 2000. – 550 p.
75. Holland J. H. Adaptation in Natural and Artificial Systems / J. H. Holland. – Boston: MIT Press, 1992. – 288 p.

76. Grabmeier J., Rudolph A. Techniques of cluster algorithms in data mining // *Data Mining and Knowledge Discovery*. – 2002. – Vol. 6, № 4. – P. 303-360.
77. Fayyad U.M., Piatetsky-Shapiro G., Smyth P., Uthurusamy R. *Advances in Knowledge Discovery and Data Mining*. – Cambridge: AAAI/MIT Press, 1996. – 625 p.
78. Nahm U.Y., Mooney R.J. A mutually beneficial integration of data mining and information extraction // *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-00)*. – Austin. (USA). – July 31-August 2. – 2000. – P. 627-632.
79. Neto J.L., Santos A.D., Kaestner C.A.A., Freitas A.A. Document clustering and text summarization // *Proceedings of the 4th International Conference on Practical Applications of Knowledge Discovery and Data Mining (PADD-2000)*. – Manchester. (UK). – April 11-13. – 2000. – P. 41-55.
80. Milligan G. An examination of procedures for determining the number of clusters in a data set / G. Milligan, M. Cooper // *Psychometrika*. – June 1985. – Vol. 50. – No. 2. – P. 159–179.
81. Pan H., Zhu J., Han D. Genetic algorithms applied to multi-class clustering for gene expression data // *Genomics Proteomics Bioinformatics*. – 2003. – Vol. 1, № 4. – P. 279-287.
82. Sebastiani F. Machine learning in automated text categorization // *ACM Computing Surveys*. – 2002. – Vol. 34, № 1. – P. 1-47.
83. Soderland S. Learning information extraction rules for semi-structured and free text // *Machine Learning*. – 1999. – Vol. 34, № 1-3. – P. 233-272.
84. Steinbach M., Karypis G., Kumar V. A comparison of document clustering techniques // *Technical Report TR 00-034*. Department of Computer Science and Engineering. – USA. University of Minnesota, May 2000. – 20 p.
85. Yang Y., Slattery S., Ghani R. A study of approaches to hypertext categorization // *Journal of Intelligent Information Systems*. – 2002. – Vol. 18, № 2. – Pages 219-241.

86. Borenstein Y., Moraglio A. "Theory and Principled Methods for the Design of Metaheuristics", due 2011, Springer.
87. Vanneschi L., Gustafson S., Moraglio A., Falco I. De, Ebner M. "Genetic Programming: 12th European Conference", EuroGP 2009 Tubingen, Germany, April, 15-17, 2009, Springer.
88. Zuhba A.V. NP-Completeness of the Classification Problem // Pattern Recognition and Image Analysis. — 2010. — Vol. 20, №.4. — P. 484–494.
89. Dialog graphical system of classification with the help of distance function / [Stekh Y., Faisal M.E. Sardieh, Kernytskyy A., Nykyforchyn R.]// Proc. of the XVI Ukrainian-Polish Conference on “CAD in Machinery Design. Implementation and Education Problems” CADMD. - Lviv,2008 – P. 88-89.
90. Some trends in knowledge discovery and data mining / [Lobur M., Stekh, Y., Kernytskyy A., Faisal M.E. Sardieh] // Proc. of the IVth International Conference MEMSTECH. – Lviv-Polyana, 2008. – P. 95-97
91. Stekh Y. Neural network based clustering algorithm / Stekh Y., Faisal M.E. Sardieh, Lobur M. // Proc. of the Vth International Conference MEMSTECH. – Lviv-Polyana, 2009. – P. 168-169.
92. Graphical System for Pattern Recognition / [Stekh Y., Faisal M.E. Sardieh, Kernytskyy A., Lobur M.] // Proc. of the Xth International Conference CADSM. – Lviv-Polyana, 2009. – P. 483.
93. Stekh Y. Estimation of optimal clustering results / [Stekh Y., Faisal M.E. Sardieh, Kernytskyy A., Dombrova M.] // Proc. of the Vth International Conference on Computer Science and Information Technologies. – Lviv, 2010. – P. 173-174.
94. Stekh Y. Algorithm for clustering Web documents / [Stekh Y., Faisal M. E. Sardieh, Lobur M., Dombrova M.]// Proc. of the VIth International Conference MEMSTECH. – Lviv-Polyana, 2010. – P. 187.

95. Stekh Y. System for a cluster analysis / Stekh Y., Fajsal M.E. Sardieh, Lobur M. // Proc. of the Xth International Conference TCSET. – Lviv-Polyana, 2010. – P. 236.
96. Development and study of clustering algorithms for large sets of data / [Stekh Y., Lobur M., Fajsal M.E. Sardieh, Dombrova M., Artsibasov V.] // Proc. of the XIth International Conference CADSM. – Lviv-Polyana, 2011. – P. 202-204.
97. Research and development of methods and algorithms non-hierarchical clustering / [Stekh Y., Lobur M., Fajsal M.E. Sardieh, Dombrova M., Artsibasov V.] // Proc. of the XIth International Conference CADSM. – Lviv-Polyana, 2011. – P. 205-207.
98. Balas E., Niehaus W. Optimized crossover-based genetic algorithms for the maximum cardinality and maximum weight clique problems. *J. Heuristics*. v4 (1998), N4, pp 107–122.
99. Boese K. D., Kahng A. B., Muddu S. A new adaptive multi-start technique for combinatorial global optimizations. *Oper. Res. Lett.* v16 (1994), N2, pp 101-114.
100. Kirkpatrick S., Toulouse G. Configuration space analysis of traveling salesman problems. *J. de Phys.* v46 (1985) , pp 1277-1292.
101. Wolpert, D.H., and Macready, W.G., "Coevolutionary Free Lunches", *IEEE Transactions on Evolutionary Computation*, 9, 721-735, 2005.

**ДОДАТОК А**  
**ПРОГРАМНА РЕАЛІЗАЦІЯ НАЙПРОСТІШОЇ ГЕНЕРАЦІЇ**  
**ВИПАДКОВИХ ГРАФІВ**

```
using System;
using System.Text;
using Utils;

namespace SimpleGen
{
    class Program
    {
        static SymmetricalMatrix matrix;

        static int getNumberFromKeyboard()
        {
            int n;
            string s = Console.ReadLine();
            while (!int.TryParse(s, out n))
            {
                Console.WriteLine("Неверный формат числа\n");
                s = Console.ReadLine();
            }
            return n;
        }

        static void Main(string[] args)
        {
            Console.WriteLine("-----");
            Console.WriteLine("Программа генерации случайных связанных кубических графов");
            Console.WriteLine("Примитивный алгоритм");
            Console.WriteLine("-----");
            Console.WriteLine("");

            int size; //число вершин графа
            int maxMatrixCount; //число генерируемых графов

            Console.Write("Введите четное число (4 и более) вершин графа: ");
            size = getNumberFromKeyboard();

            while (size % 2 != 0 || size < 4)
            {
                if (size % 2 != 0)
```

```

        Console.WriteLine("Число вершин графа должно быть четным. ");
        if (size < 4)
            Console.WriteLine("\nЧисло вершин кубического графа должно быть больше 3.");

        Console.WriteLine("\nВведите число вершин еще раз: ");
        size = getNumberFromKeyboard();
    }

    Console.WriteLine("Введите количество графов: ");
    maxMatrixCount = getNumberFromKeyboard();

    Console.WriteLine(String.Format("\nВыполняется генерация {0} кубических графов с {1}
    вершинами\n",
        maxMatrixCount, size));

    //необходимое число единиц в верхнем треугольнике матрицы смежности
    int count = size * 3 / 2;

    //создаем матрицу смежности, заполняя её необходимым количеством единиц
    matrix = new SymmetricalMatrix(size);
    for (int i = 0; i < count; i++)
    {
        matrix.data[i] = 1;
    }

    DateTime dStart = DateTime.Now;
    //генерируем кубические графы
    int matrixCount = 0;
    while (matrixCount < maxMatrixCount)
    {
        matrixCount++;
        Console.WriteLine("Матрица №" + matrixCount + ": ");
        Console.WriteLine(RandomBuild());
    }

    DateTime dStop = DateTime.Now;

    TimeSpan timeInfo = dStop.Subtract(dStart);

    Console.WriteLine(String.Format("Генерация завершена за {0} сек",
timeInfo.TotalSeconds));
    Console.ReadLine();
}

static Random rand = new Random();
static SymmetricalMatrix RandomBuild()
{

```

```
while (true)
{
  for (int i = 0; i < matrix.data.Length - 1; i++)
  {
    int ind = rand.Next(i, matrix.data.Length - 1);

    byte t = matrix.data[i];
    matrix.data[i] = matrix.data[ind];
    matrix.data[ind] = t;
  }

  if (matrix.IsCubicGraphMatrix() && matrix.IsConnectedGraphMatrix())
    return matrix;
} } } }
```

## ДОДАТОК Б

### ПРОГРАМНА РЕАЛІЗАЦІЯ ВИПАДКОВИХ КУБІЧНИХ ГРАФІВ

```

using System;
using System.Text;
using Utils;

namespace GBJG//от первых букв имен создателей оригинальной версии
{
    class Program
    {
        public static Random Rand = new Random();

        //для считывания числа с клавиатуры
        static int getNumberFromKeyboard()
        {
            int n;
            string s = Console.ReadLine();
            while (!int.TryParse(s, out n))
            {
                Console.WriteLine("Неверный формат числа. Введите число еще раз: ");
                s = Console.ReadLine();
            }

            return n;
        }

        static void Main(string[] args)
        {
            Console.WriteLine("-----");
            Console.WriteLine("Программа генерации случайных связанных кубических графов");
            Console.WriteLine("-----");
            Console.WriteLine("");

            int size; //число вершин графа
            int maxMatrixCount; //число генерируемых графов

            Console.WriteLine("Введите четное число (4 и более) вершин графа: ");
            size = getNumberFromKeyboard();

            while (size % 2 != 0 || size < 4)
            {
                if (size % 2 != 0)
                    Console.WriteLine("Число вершин графа должно быть четным. ");
            }
        }
    }
}

```

```

    if (size < 4)
        Console.WriteLine("\nЧисло вершин кубического графа должно быть больше 3.");

    Console.WriteLine("\nВведите число вершин еще раз: ");
    size = getNumberFromKeyboard();
}

Console.WriteLine("Введите количество графов: ");
maxMatrixCount = getNumberFromKeyboard();

Console.WriteLine(String.Format("\nВыполняется генерация {0} кубических графов с {1}
вершинами\n",
    maxMatrixCount, size));

DateTime dStart = DateTime.Now;

//генерируем кубические графы
int matrixCount = 0;
while (matrixCount < maxMatrixCount)
{
    Graph g = new Graph(size);

    matrixCount++;
    Console.WriteLine("Матрица №" + matrixCount + ": ");
    Console.WriteLine(g);
}

DateTime dStop = DateTime.Now;

TimeSpan timeInfo = dStop.Subtract(dStart);

Console.WriteLine(String.Format("Генерация {0} кубических графов с {1} вершинами
завершена за {2} сек",
    maxMatrixCount, size, timeInfo.TotalSeconds));
Console.ReadLine();
}
}

//Кубический случайный граф
class Graph
{
    SymmetricalMatrix matrix;//матрица смежности графа

    public Graph(int size)
    {
        currSize = 4;
        matrix = new SymmetricalMatrix(size);
    }
}

```

```

//начальный граф - это 4-х вершинная клика
matrix[0, 1] = 1;
matrix[0, 2] = 1;
matrix[0, 3] = 1;

matrix[1, 2] = 1;
matrix[1, 3] = 1;

matrix[2, 3] = 1;

while (!GBJGOperation()) ;//преобразуем начальный граф, пока это возможно
}

int currSize;
private bool GBJGOperation()
{
    if (matrix.size == currSize)
        return true; //исходный граф по построению кубический

    if (matrix.size - currSize == 2)
    {
        op2();
        return true;
    }
    else if (matrix.size - currSize == 4)
    {
        int p = Program.Rand.Next(0, 99);

        if (p <= 49)
        {
            op2(); op2();
        }
        else
            op4();

        return true;
    }
    else if (matrix.size - currSize == 6)
    {
        selectOp();
        return true;
    }

    selectOp();
    return false;
}

```

```

// случайный выбор операции, когда можно добавить 6 и более вершин к графу
private void selectOp()
{
    int p = Program.Rand.Next(0, 99);

    if (p <= 19)
    {
        op2(); op2(); op2();
    }
    else if (p <= 39)
    {
        op4(); op2();
    }
    else if (p <= 59)
    {
        op2(); op4();
    }
    else if (p <= 79)
        op61();
    else
        op62();
}

//случайный выбор двух различных ребер
private int[] select2Edge()
{
    int[] e = selectEdge();

    int i2 = Program.Rand.Next(1, currSize - 1);
    int j2 = Program.Rand.Next(1, currSize - 1);

    while (matrix[i2, j2] == 0 || //по случайному адресу i2, j2 ребро действительно должно
    существовать
        (i2 == e[0] && j2 == e[1]) ||
        (i2 == e[1] && j2 == e[0]) ||
        i2 == j2)
    {
        i2 = Program.Rand.Next(1, currSize);
        j2 = Program.Rand.Next(1, currSize);
    }

    return new int[] { e[0], e[1], i2, j2 };
}

//случайный выбор одного ребра
private int[] selectEdge()

```

```

{
    int i1 = Program.Rand.Next(1, currSize);
    int j1;
    while ((j1 = Program.Rand.Next(1, currSize)) == i1) ;

    while (matrix[i1, j1] == 0)//по случайному адресу i1, j1 ребро действительно должно
    существовать
    {
        i1 = Program.Rand.Next(1, currSize);
        while ((j1 = Program.Rand.Next(1, currSize)) == i1) ;
    }

    return new int[] { i1, j1 };
}

private void op2()
{
    int[] e = select2Edge();

    //удаляем старые ребра
    matrix[e[0], e[1]] = 0;
    matrix[e[2], e[3]] = 0;

    //добавляем новые ребра
    matrix[currSize, currSize + 1] = 1;

    matrix[e[0], currSize] = 1;
    matrix[e[1], currSize] = 1;
    matrix[e[2], currSize + 1] = 1;
    matrix[e[3], currSize + 1] = 1;

    currSize = currSize + 2;
}

private void op4()
{
    int[] e = selectEdge();

    //удаляем старое ребро
    matrix[e[0], e[1]] = 0;

    //добавляем подграф
    matrix[currSize, currSize + 1] = 1;
    matrix[currSize, currSize + 2] = 1;
    matrix[currSize, currSize + 3] = 1;
    matrix[currSize + 1, currSize + 2] = 1;
    matrix[currSize + 2, currSize + 3] = 1;
}

```

```

//соединяем новый подграф с текущим
matrix[e[0], currSize + 1] = 1;
matrix[e[1], currSize + 3] = 1;

currSize = currSize + 4;
}

private void op61()
{
    int[] e = selectEdge();

    //удаляем старое ребро
    matrix[e[0], e[1]] = 0;

    //добавляем подграф
    matrix[currSize, currSize + 1] = 1;
    matrix[currSize, currSize + 2] = 1;
    matrix[currSize, currSize + 3] = 1;
    matrix[currSize + 1, currSize + 3] = 1;
    matrix[currSize + 1, currSize + 4] = 1;
    matrix[currSize + 2, currSize + 3] = 1;
    matrix[currSize + 2, currSize + 4] = 1;
    matrix[currSize + 4, currSize + 5] = 1;
    //соединяем новый подграф с текущим
    matrix[e[0], currSize + 5] = 1;
    matrix[e[1], currSize + 5] = 1;

    currSize = currSize + 6;
}

private void op62()
{
    int[] e = select2Edge();

    //удаляем старые ребра
    matrix[e[0], e[1]] = 0;
    matrix[e[2], e[3]] = 0;

    //добавляем подграф
    matrix[currSize, currSize + 1] = 1;
    matrix[currSize, currSize + 2] = 1;
    matrix[currSize, currSize + 3] = 1;
    matrix[currSize + 1, currSize + 2] = 1;
    matrix[currSize + 1, currSize + 5] = 1;
    matrix[currSize + 2, currSize + 3] = 1;
    matrix[currSize + 3, currSize + 4] = 1;
}

```

```

//соединяем новый подграф с текущим
matrix[e[0], currSize + 4] = 1;
matrix[e[1], currSize + 4] = 1;
matrix[e[2], currSize + 5] = 1;
matrix[e[3], currSize + 5] = 1;

currSize = currSize + 6;
}

public override string ToString()
{
    return matrix.ToString();
}
}
}

```

Реализация на C++:

Класс Graph:

```

#include "SymmetricalMatrix.h"

extern "C" int rand();
int randint(int min, int max) // диапазон min..max
{
    return (rand() % (max-min+1) + min);
}

//Кубический случайный граф
class Graph
{
public:

    SymmetricalMatrix* matrix;//матрица смежности графа

    void ToString()
    {
        matrix->ToString();
    }

    Graph()
    {}

    Graph(int size)
    {
        matrix = new SymmetricalMatrix(size);
    }
}

```

```

}

void Generate()
{
    currSize = 4;
    memset(matrix->data, 0, sizeof(int) * matrix->size * (matrix->size - 1) / 2);

    //начальный граф - это 4-х вершинная клика
    matrix->setData(0, 1, 1);
    matrix->setData(0, 2, 1);
    matrix->setData(0, 3, 1);

    matrix->setData(1, 2, 1);
    matrix->setData(1, 3, 1);

    matrix->setData(2, 3, 1);

    while (!GBJGOperation()) ;//преобразуем начальный граф, пока это возможно
}

```

private:

```

int currSize;

bool GBJGOperation()
{
    if (matrix->size == currSize)
        return true; //исходный граф по построению кубический

    if (matrix->size - currSize == 2)
    {
        op2();
        return true;
    }
    else if (matrix->size - currSize == 4)
    {
        int p = randint(0, 99);

        if (p <= 49)
        {
            op2(); op2();
        }
        else
            op4();

        return true;
    }
}

```

```

else if (matrix->size - currSize == 6)
{
    selectOp();
    return true;
}

selectOp();
return false;
}

// случайный выбор операции, когда можно добавить 6 и более вершин к графу
void selectOp()
{
    int p = randint(0, 99);

    if (p <= 19)
    {
        op2(); op2(); op2();
    }
    else if (p <= 39)
    {
        op4(); op2();
    }
    else if (p <= 59)
    {
        op2(); op4();
    }
    else if (p <= 79)
        op61();
    else
        op62();
}

//случайный выбор двух различных ребер
void select2Edge(int* m)
{
    int e[2];
    selectEdge(e);

    int i2 = randint(1, currSize - 2);
    int j2 = randint(1, currSize - 2);

    while (matrix->getData(i2, j2) == 0 || //по случайному адресу i2, j2 ребро
действительно должно существовать
        (i2 == e[0] && j2 == e[1]) ||
        (i2 == e[1] && j2 == e[0]) ||
        i2 == j2)

```

```

{
    i2 = randint(1, currSize - 1);
    j2 = randint(1, currSize - 1);
}

    m[0] = e[0];
    m[1] = e[1];
    m[2] = i2;
    m[3] = j2;

}

//случайный выбор одного ребра
void selectEdge(int* m)
{
    int i1 = randint(1, currSize - 1);
    int j1;
    while ((j1 = randint(1, currSize - 1)) == i1);

    while (matrix->getData(i1, j1) == 0)//по случайному адресу i1, j1 ребро действительно
должно существовать
    {
        i1 = randint(1, currSize - 1);
        while ((j1 = randint(1, currSize - 1)) == i1);
    }

    m[0] = i1;
    m[1] = j1;

}

void op2()
{
    int e[4];
    select2Edge(e);

    //удаляем старые ребра
    matrix->setData(e[0], e[1], 0);
    matrix->setData(e[2], e[3], 0);

    //добавляем новые ребра
    matrix->setData(currSize, currSize + 1, 1);

    matrix->setData(e[0], currSize, 1);
    matrix->setData(e[1], currSize, 1);
    matrix->setData(e[2], currSize + 1, 1);
    matrix->setData(e[3], currSize + 1, 1);
}

```

```

    currSize = currSize + 2;
}

void op4()
{
    int e[2];
        selectEdge(e);

    //удаляем старое ребро
    matrix->setData(e[0], e[1], 0);

    //добавляем подграф
    matrix->setData(currSize, currSize + 1, 1);
    matrix->setData(currSize, currSize + 2, 1);
    matrix->setData(currSize, currSize + 3, 1);
    matrix->setData(currSize + 1, currSize + 2, 1);
    matrix->setData(currSize + 2, currSize + 3, 1);
    //соединяем новый подграф с текущим
    matrix->setData(e[0], currSize + 1, 1);
    matrix->setData(e[1], currSize + 3, 1);

    currSize = currSize + 4;
}

void op61()
{
    int e[2];
        selectEdge(e);

    //удаляем старое ребро
    matrix->setData(e[0], e[1], 0);

    //добавляем подграф
    matrix->setData(currSize, currSize + 1, 1);
    matrix->setData(currSize, currSize + 2, 1);
    matrix->setData(currSize, currSize + 3, 1);
    matrix->setData(currSize + 1, currSize + 3, 1);
    matrix->setData(currSize + 1, currSize + 4, 1);
    matrix->setData(currSize + 2, currSize + 3, 1);
    matrix->setData(currSize + 2, currSize + 4, 1);
    matrix->setData(currSize + 4, currSize + 5, 1);
    //соединяем новый подграф с текущим
    matrix->setData(e[0], currSize + 5, 1);
    matrix->setData(e[1], currSize + 5, 1);
}

```

```

    currSize = currSize + 6;
}

void op62()
{
    int e[4];
    select2Edge(e);

    //удаляем старые ребра
    matrix->setData(e[0], e[1], 0);
    matrix->setData(e[2], e[3], 0);

    //добавляем подграф
    matrix->setData(currSize, currSize + 1, 1);
    matrix->setData(currSize, currSize + 2, 1);
    matrix->setData(currSize, currSize + 3, 1);
    matrix->setData(currSize + 1, currSize + 2, 1);
    matrix->setData(currSize + 1, currSize + 5, 1);
    matrix->setData(currSize + 2, currSize + 3, 1);
    matrix->setData(currSize + 3, currSize + 4, 1);

    //соединяем новый подграф с текущим
    matrix->setData(e[0], currSize + 4, 1);
    matrix->setData(e[1], currSize + 4, 1);
    matrix->setData(e[2], currSize + 5, 1);
    matrix->setData(e[3], currSize + 5, 1);

    currSize = currSize + 6;
}

};

```

Основная программа:

```

#include "stdafx.h"
#include "Graph.h"
#include <ctime>
#include <conio.h>

//ввод с клавиатуры количества вершин
void getCountVertex(int* n)
{
    char czC[255];
    while ( !*n || *n %2 !=0 || *n < 4)
    {
        printf("Input count vertex: ");
        scanf( "%s", czC );
    }
}

```

```

        *n = atoi( czC );

        if(*n %2 !=0)
            printf("The number of vertices should be even\n");
        if(*n < 4 )
            printf("number of vertex must be greater than or equal to 4\n");
    }
}

//ввод с клавиатуры количества вершин
void getCountGraph(int* n)
{
    char czC[255];
    while ( ( *n = atoi( czC ) ) == 0 || !*n )
    {
        printf("Input count graph: ");
        scanf( "%s", czC );
    }
}

int main(int argc, char* argv[])
{
    printf("-----\n");
    printf("ZAS110 kniazkov_maxim@mail.ru\n");
    printf("-----\n\n");

    int size = 0; //число вершин графа
    int maxMatrixCount = 0; //число генерируемых графов

    getCountVertex(&size);
    getCountGraph(&maxMatrixCount);

    //генерируем кубические графы
    clock_t t0 = clock();

    int matrixCount = 0;
    Graph g(size);
    while (matrixCount < maxMatrixCount)
    {
        g.Generate();

        matrixCount++;
        printf("Matrix #d:\n", matrixCount);
        g.ToString();
    }
}

```

```
clock_t t1 = clock();

printf("\nGenerate of %d graphs with %d vertex completed of %f sec\n",
      maxMatrixCount,
      size, (double)(t1 - t0) / CLOCKS_PER_SEC);

getch();
}
```

## ДОДАТОК В

### ПРОГРАМНА РЕАЛІЗАЦІЯ СИМЕТРИЧНОЇ МАТРИЦІ ВИПАДКОВОГО ГРАФУ

```

using System;
using System.Text;

namespace Utils
{
    /// <summary>
    /// Симметричная бинарная матрица, с нулями по диагонали
    /// </summary>
    public class SymmetricalMatrix
    {
        //размер матрицы
        public readonly int size;

        //верхний треугольник
        public readonly byte[] data;

        public SymmetricalMatrix(int size)
        {
            this.size = size;
            data = new byte[(size * (size - 1)) / 2];
        }

        //определение индекса в data по обычным индексам
        //i и j квадратной матрицы
        private int getDataIndex(int i, int j)
        {
            int k = i + 1;
            int l = (k * k + k) / 2;
            return i * size + j - l;
        }

        //индексируется как обычная квадратная матрица
        public byte this[int i, int j]
        {
            get
            {
                if (i == j)
                    return 0;
            }
        }
    }
}

```

```

    if (i < j)
        return data[getDataIndex(i, j)];
    else
        return data[getDataIndex(j, i)];
}
set
{
    if (i == j)
        return;

    if (i < j)
        data[getDataIndex(i, j)] = value;
    else
        data[getDataIndex(j, i)] = value;
}
}

```

//это действительно корректная матрица смежности для кубического графа ?

```

public bool IsCubicGraphMatrix()
{
    //считаем суммы по столбцам, они должны быть все равны 3
    for (int i = 0; i < size; i++)
    {
        int sum = 0;
        for (int j = 0; j < size; j++)
        {
            if (this[j, i] != 0)
                sum++;
        }
        if (sum != 3) return false;
    }

    return true;
}

public bool IsConnectedGraphMatrix()
{
    return true; //заглушка
}

```

```

public override string ToString()
{
    StringBuilder sb = new StringBuilder();
    for (int j = 0; j < size; j++)
    {
        for (int i = 0; i < size; i++)

```

```

        sb.AppendFormat("{0} ", this[i, j]);
        sb.AppendLine();
    }
    return sb.ToString();
}
}
}

```

Реализация на C++:

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

class SymmetricalMatrix
{
public:
    int size;//размер матрицы

    //верхний треугольник
    int* data;

    //индексируется как обычная квадратная матрица
    int getData(int i, int j)
    {
        if (i == j)
            return 0;

        if (i < j)
            return data[getDataIndex(i, j)];
        else
            return data[getDataIndex(j, i)];
    }

    void setData(int i, int j, int value)
    {
        if (i == j)
            return;

        if (i < j)
            data[getDataIndex(i, j)] = value;
        else
            data[getDataIndex(j, i)] = value;
    }
}

```

```

void ToString()
{
    for (int j = 0; j < size; j++)
    {
        for (int i = 0; i < size; i++)
            printf("%d ", getData(i, j) );
        printf("\n");
    }
    printf("\n");
}

SymmetricalMatrix()
{}

SymmetricalMatrix(int s)
{
    size = s;

    data = new int[ size * (size - 1) / 2 ];
    memset(data, 0, sizeof(int) * size * (size - 1) / 2);
}

~SymmetricalMatrix()
{
    delete[] data;
}

private:
//определение индекса в data по обычным индексам
//i и j квадратной матрицы
int getDataIndex(int i, int j)
{
    int k = i + 1;
    int l = (k * k + k) / 2;
    return i * size + j - l;
}
};

```

## ДОДАТОК Г

## БІБЛІОТЕКА ТЕСТОВИХ ЗАДАЧ РОЗМІЩЕННЯ ВИРОБНИЦТВА

## Клас 1

## 20 ЗАВДАНЬ, ОБСЯГИ ВИРОБНИЦТВА = 10

Стовпець LP – нижня межа, отримана методом лінійного програмування

Стовпець MLP – нижня межа, отримана релаксацією умови цілочисельності змінних  $x_{ij}$

Стовпець OPT – оптимальне значення цільової функції

Стовпець UB – верхня оцінка

код	LP	MLP	OPT	UB	Допустиме рішення (множина обраних підприємств)
1	2769,04	2796,0	2842,8	2894,7	1, 5, 11, 13, 17, 20, 23, 25, 26, 29, 30, 31, 33, 42, 51, 53, 58, 72, 78, 79, 86, 94
2	2893,92	2916,4	2969,6	3016,6	1, 5, 8, 13, 19, 21, 22, 23, 29, 40, 54, 58, 60, 61, 64, 72, 80, 81, 82, 91, 98
3	2960,90	2995,8	3053,3	3147,7	3, 11, 17, 28, 34, 36, 43, 46, 52, 53, 55, 56, 57, 64, 66, 68, 73, 76, 81, 82, 85, 86, 90
4	2841,61	2866,4	2943,1	3004,2	8, 9, 10, 22, 25, 27, 28, 30, 33, 34, 36, 52, 60, 63, 68, 74, 75, 76, 79, 80, 81, 100
5	2897,74	2911,4	2974,5	3077,1	2, 4, 5, 6, 7, 18, 19, 22, 25, 27, 34, 36, 39, 57, 58, 71, 82, 85, 87, 92, 96, 98
6	2967,66	2982,2	3040,1	3100,2	12, 14, 15, 19, 24, 29, 36, 39, 54, 56, 61, 62, 63, 64, 66, 67, 68, 74, 75, 78, 79, 80, 86, 91
7	2922,90	2960,6	3013,7	3114,9	1, 4, 6, 7, 9, 10, 18, 21, 23, 25, 33, 54, 55, 62, 66, 82, 86, 87, 88, 91, 99, 100
8	2825,37	2849,9	2891,4	2966,5	7, 9, 22, 29, 40, 44, 47, 50, 51, 55, 57, 58, 63, 64, 66, 67, 76, 78, 80, 86, 88, 94
9	2888,47	2924,4	2984,9	3035,1	2, 4, 8, 17, 20, 22, 27, 33, 34, 37, 39, 40, 46, 52, 55, 65, 67, 76, 84, 88, 93, 94, 97
10	2926,47	2967,9	3029,3	3112,0	10, 15, 16, 17, 25, 29, 32, 35, 41, 47, 50, 55, 61, 69, 72, 75, 76, 78, 79, 86, 94, 95, 99
11	2924,95	2944,9	3016,4	3085,1	3, 4, 5, 11, 16, 18, 21, 29, 49, 56, 65, 67, 68, 71, 74, 76, 77, 78, 80, 81, 84, 90, 91
12	2906,06	2938,6	3003,5	3090,0	3, 8, 19, 24, 30, 35, 39, 43, 49, 62, 69, 72, 75, 76, 77, 78, 82, 85, 86, 88, 91, 93, 96

код	LP	MLP	OPT	UB	Допустиме рішення (множина обраних підприємств)
13	2866,65	2895,6	2954,6	3032,0	3, 4, 6, 10, 19, 28, 30, 34, 35, 36, 39, 42, 43, 51, 66, 68, 80, 85, 87, 89, 90, 93, 97, 98
14	2859,76	2884,3	2936,4	3016,8	1, 5, 9, 11, 15, 17, 19, 40, 42, 46, 48, 56, 62, 66, 69, 74, 75, 77, 78, 83, 85
15	2796,68	2827,9	2886,1	2939,6	2, 16, 19, 32, 35, 37, 42, 45, 46, 47, 57, 61, 62,, 64, 67, 77, 78, 81, 87, 92, 93
16	2862,12	2869,0	2926,4	2986,7	3, 6, 13, 16, 19, 23, 24, 43, 49, 50, 51, 56, 61, 64, 65, 67, 73, 75, 86, 89, 91
17	2910,78	2939,8	2992,6	3045,2	5, 7, 9, 10, 13, 16, 23, 28, 35, 36, 45, 47, 58, 60, 69, 74, 78, 79, 80, 90, 96, 99
18	2812,08	2830,4	2897,3	2945,7	2, 6, 8, 9, 15, 17, 24, 28, 33, 41, 48, 52, 56, 61, 63, 77, 81, 82, 84, 89, 92
19	2856,11	2879,7	2938,3	2969,3	2, 8, 11, 21, 23, 25, 27, 33, 34, 42, 44, 46, 47, 50, 59, 75, 76, 84, 86, 88, 95, 98, 100
20	2855,65	2882,3	2940,0	3023,2	3, 4, 6, 7, 12, 15, 19, 21, 24, 26, 32, 39, 42, 48, 52, 58, 61, 63, 66, 80, 87, 93, 99

## Клас 2

### 20 ЗАВДАНЬ, ОБСЯГИ ВИРОБНИЦТВА = 20

Стовпець LP – нижня межа, отримана методом лінійного програмування

Стовпець OPT – оптимальне значення цільової функції

Стовпець UB – верхня оцінка

код	LP	OPT	UB	Допустиме рішення (множина обраних підприємств)
1	2049,18	2110,839	2130,4	1, 5, 11, 16, 17, 23, 25, 29, 44, 53, 73, 75, 85, 86, 94
2	2151,964	2196,111	2224,2	5, 6, 21, 22, 24, 29, 40, 43, 51, 58, 61, 66, 80, 88, 97
3	2186,485	2250,047	2279,9	10, 18, 26, 31, 52, 53, 55, 66, 68, 69, 73, 84, 85, 86, 90
4	2095,343	2164,932	2183,0	3, 8, 9, 10, 19, 22, 25, 27, 28, 33, 38, 56, 60, 75, 93
5	2089,726	2145,423	2176,2	2, 4, 7, 12, 18, 19, 21, 22, 40, 80, 81, 82, 85, 87, 92, 94
6	2204,322	2273,843	2303,1	12, 19, 23, 29, 39, 56, 57, 60, 61, 62, 68, 76, 78, 80, 84, 91

код	LP	OPT	UB	Допустиме рішення (множина обраних підприємств)
7	2170,993	2224,963	2265,1	1, 2, 7, 10, 18, 21, 25, 28, 33, 37, 49, 59, 73, 87, 91, 98
8	2098,142	2140,044	2156,4	7, 20, 29, 40, 47, 50, 51, 55, 57, 63, 71, 76, 86, 88
9	2157,882	2216,528	2246,4	22, 24, 26, 29, 33, 34, 39, 40, 44, 46, 55, 60, 63, 77, 90, 97
10	2142,529	2216,789	2240,4	2, 6, 10, 12, 29, 35, 50, 53, 55, 60, 61, 69, 70, 76, 79, 94
11	2168,987	2230,958	2261,5	3, 5, 7, 8, 11, 16, 18, 44, 49, 67, 68, 77, 78, 81, 84, 88
12	2160,239	2225,163	2254,0	13, 14, 19, 21, 40, 52, 69, 72, 74, 75, 77, 78, 82, 85, 86, 93
13	2090,554	2160,767	2175,0	3, 4, 19, 30, 39, 43, 68, 70, 80, 85, 87, 89, 90, 93, 95, 98
14	2160,221	2226,947	2254,1	1, 9, 15, 19, 20, 42, 43, 45, 46, 54, 59, 62, 69, 74, 79, 96
15	2092,174	2156,231	2184,9	2, 6, 10, 12, 16, 24, 26, 35, 39, 42, 47, 51, 57, 61, 93, 99
16	2154,244	2210,485	2235,6	6, 13, 19, 20, 33, 38, 43, 61, 67, 68, 71, 73, 88, 91, 98
17	2172,755	2239,593	2263,2	2, 5, 8, 9, 16, 30, 35, 36, 47, 64, 66, 69, 79, 82, 85, 90
18	2105,997	2175,144	2198,9	1, 6, 8, 12, 15, 24, 33, 41, 49, 59, 77, 82, 84, 91, 96
19	2096,486	2153,092	2183,4	3, 16, 23, 33, 34, 42, 44, 47, 50, 68, 73, 76, 86, 92 100
20	2119,986	2178,856	2218,2	4, 6, 7, 15, 22, 24, 32, 39, 52, 57, 58, 66, 80, 84, 99

### Клас 3

#### 20 ЗАВДАНЬ, ОБСЯГИ ВИРОБНИЦТВА = 30

Стовпець LP – нижня межа, отримана методом лінійного програмування

Стовпець OPT - оптимальне значення цільової функції

Стовпець UB – верхня оцінка

код	LP	OPT	UB	Допустиме рішення (множина обраних підприємств)
-----	----	-----	----	--

код	LP	OPT	UB	Допустиме рішення (множина обраних підприємств)
1	1822.6	1923.9	1924.2	5, 11, 16, 25, 29, 40, 44, 50, 73, 86, 94 100
2	1914.9	2015.7	2032.1	5, 6, 8, 19, 22, 40, 44, 54, 58, 61, 88, 91
3	1939.3	2053.4	2084.1	21, 31, 35, 36, 42, 46, 57, 66, 68, 69, 72, 76, 90
4	1876.7	2000.8	2015.9	12, 25, 26, 28, 38, 52, 59, 60, 68, 75, 76, 81
5	1833.0	1942.8	1968.8	2, 8, 18, 21, 34, 39, 40, 72, 80, 85, 94
6	1976.9	2102.1	2123.5	9, 12, 23, 24, 29, 36, 54, 61, 76, 78, 84, 91, 94
7	1928.2	2041.0	2088.1	5, 17, 21, 27, 28, 33, 38, 39, 42, 43, 67, 76
8	1870.4	1978.4	1990.8	4, 7, 40, 43, 50, 51, 55, 57, 63, 75, 76, 88
9	1890.1	1992.5	2005.5	17, 18, 24, 27, 33, 34, 43, 46, 60, 76, 77, 87
10	1918.2	2062.5	2066.6	2, 12, 29, 35, 50, 61, 68, 69, 76, 79, 80, 89, 94
11	1927.8	2024.8	2036.8	3, 5, 8, 11, 18, 29, 44, 67, 78, 81, 84, 91
12	1934.1	2050.9	2084.5	3, 6, 8, 14, 19, 35, 46, 53, 59, 72, 77, 91
13	1853.1	1979.0	1996.9	3, 4, 23, 25, 34, 42, 58, 61, 68, 85, 87, 90, 98
14	1941.9	2053.7	2075.2	4, 15, 20, 26, 36, 50, 55, 56, 62, 66, 69, 74, 77
15	1855.8	1944.9	1944.9	2, 8, 24, 26, 29, 39, 42, 47, 61, 72, 93, 99
16	1927.6	2032.1	2042.2	19, 22, 28, 38, 48, 61, 67, 68, 71, 73, 88, 91
17	1954.2	2059.7	2063.7	1, 6, 7, 9, 10, 16, 28, 35, 45, 47, 60, 66, 69
18	1883.5	2015.0	2021.6	1, 6, 15, 17, 24, 41, 74, 75, 77, 82, 91, 93
19	1856.6	1962.1	1962.1	2, 16, 33, 34, 44, 47, 60, 69, 73, 86, 92 100
20	1886.9	1961.3	1973.3	4, 7, 10, 15, 24, 27, 32, 39, 44, 52, 57, 58, 84

#### Клас 4

#### 20 ЗАВДАНЬ, ОБСЯГИ ВИРОБНИЦТВА = 40

Стовпець LP – нижня межа, отримана лінійним програмуванням

Стовпець OPT – оптимальне значення цільової функції

Стовпець UB – верхня оцінка

код	LP	OPT	UB	Допустиме рішення (множина обраних підприємств)
1	1760.5	1879.9	1879.9	11, 25, 29, 40, 44, 50, 73, 86, 94, 100
2	1851.1	1959.9	1959.9	6, 8, 19, 22, 40, 54, 61, 66, 68, 88, 91
3	1874.1	1990.3	2002.7	3, 11, 21, 36, 42, 57, 69, 72, 80, 82, 90
4	1829.6	1977.2	1978.0	3, 7, 12, 20, 25, 28, 33, 38, 59, 60, 93
5	1757.4	1869.4	1881.8	2, 18, 34, 39, 40, 48, 80, 85, 91, 92
6	1919.0	2045.8	2053.8	9, 12, 19, 29, 36, 61, 75, 76, 78, 84, 94
7	1844.6	1976.3	1993.7	1, 7, 8, 17, 21, 28, 30, 33, 38, 39, 42, 91
8	1794.2	1905.3	1942.0	4, 7, 22, 40, 43, 50, 55, 57, 62, 63, 76
9	1810.5	1931.1	1931.1	17, 18, 33, 39, 45, 49, 60, 67, 74, 76, 84
10	1878.0	1989.9	2019.5	10, 12, 31, 61, 66, 67, 68, 69, 76, 79, 80
11	1852.0	1956.2	1978.5	3, 5, 8, 18, 29, 44, 67, 78, 81, 84, 91
12	1867.0	1983.9	2007.6	6, 10, 14, 19, 32, 53, 59, 72, 78, 82, 91
13	1802.0	1919.9	1919.9	3, 4, 25, 42, 58, 61, 62, 68, 85, 87, 98
14	1880.0	1998.2	2002.7	4, 9, 15, 20, 42, 43, 50, 59, 74, 77, 100
15	1795.1	1890.4	1890.4	2, 8, 24, 29, 39, 42, 72, 74, 79, 93, 99
16	1850.6	1966.6	1966.6	4, 8, 20, 28, 33, 38, 43, 48, 61, 68, 71, 73
17	1887.1	2015.8	2037.0	1, 2, 8, 9, 10, 16, 17, 35, 66, 69, 79
18	1836.5	1976.9	1987.6	1, 15, 17, 46, 74, 75, 77, 82, 91, 92, 93
19	1801.3	1923.1	1923.1	16, 33, 34, 44, 47, 53, 70, 73, 76, 92, 100
20	1824.0	1939.2	1939.2	4, 7, 10, 15, 24, 27, 32, 39, 44, 57, 58, 84

**Клас 5**

**20 ЗАВДАНЬ, ОБСЯГИ ВИРОБНИЦТВА = 50**

Стовпець LP – нижня межа, отримана лінійним програмуванням.

Стовпець OPT – оптимальне значення цільової функції.

Стовпець UB – верхня оцінка

код	LP	OPT	UB	Допустиме рішення (множина обраних підприємств)
1	1748.0	1879.9	1879.9	11, 25, 29, 40, 44, 50, 73, 86, 94, 100
2	1740.2	1950.5	1950.5	5, 6, 8, 21, 22, 32, 54, 61, 91, 97
3	1863.8	1989.9	1989.9	3, 4, 21, 36, 38, 69, 72, 73, 82, 90, 99
4	1821.3	1974.0	1974.0	3, 7, 12, 20, 25, 28, 33, 38, 59, 60, 93
5	1742.1	1860.2	1860.2	2, 8, 20, 34, 39, 40, 48, 72, 80, 85
6	1904.3	2037.8	2044.1	9, 12, 23, 24, 29, 36, 54, 61, 76, 78, 84
7	1821.6	1975.6	1975.6	1, 7, 8, 21, 28, 30, 33, 38, 39, 42, 91
8	1773.2	1897.7	1897.7	4, 7, 22, 40, 43, 50, 55, 57, 62, 63
9	1796.5	1915.7	1915.7	17, 18, 33, 39, 45, 49, 60, 67, 74, 84
10	1875.0	1989.9	1998.4	10, 12, 31, 61, 66, 67, 68, 69, 76, 79, 80
11	1839.2	1938.5	1938.5	2, 8, 18, 38, 45, 67, 78, 84, 88, 91
12	1852.1	1965.3	1967.3	1, 3, 14, 15, 21, 35, 37, 46, 53, 82
13	1791.8	1918.5	1918.5	3, 4, 30, 39, 42, 58, 61, 63, 68, 87, 95
14	1866.1	1995.8	1995.8	4, 9, 15, 20, 42, 43, 50, 59, 74, 77, 100
15	1786.3	1879.7	1879.7	2, 8, 24, 29, 39, 42, 74, 79, 93, 99
16	1836.1	1953.7	1957.6	4, 20, 38, 43, 48, 50, 54, 61, 71, 73

<b>код</b>	<b>LP</b>	<b>OPT</b>	<b>UB</b>	<b>Допустиме рішення (множина обраних підприємств)</b>
17	1874.7	2013.9	2028.5	8, 10, 35, 36, 37, 47, 60, 64, 69, 79, 85
18	1829.3	1974.2	1975.2	8, 12, 29, 33, 38, 41, 60, 75, 77, 93, 96
19	1793.1	1915.3	1915.3	16, 33, 34, 44, 47, 53, 70, 73, 76, 92, 100
20	1815.3	1939.2	1939.2	4, 7, 10, 15, 24, 27, 32, 39, 44, 57, 58, 84

## ДОВІДКА

### про використання результатів науково-дослідної роботи Є.К. Селютіна «ФРАГМЕНТАРНІ МОДЕЛІ В ЗАДАЧАХ ОПТИМАЛЬНОЇ КЛАСИФІКАЦІЇ»

Даною довідкою підтверджується, що на підприємстві використано результати науково-дослідної роботи, що виконувалася в Запорізькому національному університеті, зокрема на базі запропоновано комп'ютерного забезпечення:

– проводиться автоматизоване розміщення рекламних блоків на сторінках друкованих видань (газета «Теленеделя», «Телескоп», «Комсомольская правда в Украине» тощо) відповідно стандартних прототипів розміщення з урахування умов симетрії;

– проводиться автоматизоване формування звітів з аналізу діяльності підприємства, що дає можливість побудувати прогноз діяльності на наступні звітні періоди;

– автоматично формуються замовлення на розміщення реклами (до 10 тис. замовлень на рік);

– дружній інтерфейс системи допомагає менеджерам оперативно вирішувати питання, пов'язані з прийомом та обробкою рекламних замовлень.

Також виконується автоматизована генерація раціону для клієнта на день, тиждень, місяць з урахуванням всіх обмежень та переваг.

Використання результатів дослідження дозволило підвищити якість планування, зменшити ризики економічної діяльності, прискорити обробку заявки клієнта у 6 разів, а також підвищити ефективність діяльності підприємства в цілому.

27.04.2021

Директор підприємства \_\_\_\_\_

