НАЦІОНАЛЬНА АКАДЕМІЯ НАУК УКРАЇНИ
ІНСТИТУТ КІБЕРНЕТИКИ ІМЕНІ В.М. ГЛУШКОВА

Кваліфікаційна наукова
праця на правах рукопису

Супрун Антон Андрійович

УДК 519.8

ДИСЕРТАЦІЯ

R-АЛГОРИТМИ ТА КВАЗІНЬЮТОНІВСЬКІ МЕТОДИ В ПРИКЛАДНИХ ЗАДАЧАХ НЕГЛАДКОЇ ОПТИМІЗАЦІЇ

113 – «Прикладна математика»

Галузь знань 11 – «Математика та статистика»

Подається на здобуття наукового ступеня доктора філософії.

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

А.А. Супрун

Науковий керівник: Стецюк Петро Іванович
доктор фізико-математичних наук,
старший науковий співробітник

Київ – 2023

АНОТАЦІЯ

Супрун А.А. r-Алгоритми та квазіньютонівські методи в прикладних задачах негладкої оптимізації. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття ступеня доктора філософії за спеціальністю 113 Прикладна математика. – Інститут кібернетики імені В.М. Глушкова Національної академії наук України, Київ. – 2023.

Зміст дисертації. У **вступі** обґрунтовано актуальність теми, сформульовано мету та задачі досліджень, розкрито наукову новизну та практичну цінність роботи, представлено її загальну характеристику.

У **розділі 1** розглянуто основні етапи розвитку градієнтних, ньютонівських та квазіньютонівських методів, а також методів негладкої оптимізації. Проаналізовано роботи вітчизняних та закордонних вчених: Л. Брегмана, В. Гершовича, Ю. Єрмольєва, І. Єрьоміна, М. Журбенка, А. Неміровського, Б. Поляка, С. Стеценка, П. Стецюка, Л. Хачияна, Н. Шора, Д. Юдіна, S. Agmon, J.F. Bonnans, S. Boyd, C. Broyden, P. Camerini, A. Conn, W. Davidon, J. Dennis, R. Fletcher, L. Fratta, J.C. Gilbert, D. Goldfarb, I.M. Gould, M. Grötschel, R. Haelterman, F. Kappel, J. Kelley, H. König, A. Kuntsevich, C. Lemarèchal, L. Lovasz, F. Maffioli, J. Morè, T. Motzkin, J. Nocedal, D. Pallaschke, E. Polak, M. J.D. Powell, C.M. Reeves, G. Ribière, I. Schoenberg, A. Schriyver, D. Shanno, P. Toint, S. Wright. У роботах вищенаведених авторів викладено загальновідомі градієнтні, ньютонівські та квазіньютонівські, а також субградієнтні методи й алгоритми, аспекти чисельного та експериментального аналізу алгоритмів та їхнє застосування до широкого спектру прикладних задач, а також новітні алгоритми та модифікації, розроблені впродовж останніх років.

Викладено теоретичні основи низки відомих методів негладкої оптимізації, описано їхній розвиток та покращення різними авторами. На основі проведеного огляду зроблено висновки та поставлено задачі для дослідження.

У **розділі 2** розглядаються H- та B-форми методів квазіньютонівського типу та методу Давидона – Флетчера – Пауелла (ДФП-методу). B-форма методу квазіньютонівського типу дозволяє легко інтерпретувати ці методи, як градієнтні в перетвореному відповідним чином просторі аргументів. Проведено порівняння цих методів з r-алгоритмами, виділено їх переваги та недоліки. Для мінімізації гладких опуклих функцій побудовано градієнтний метод з перетворенням простору, що поєднує властивості як квазіньютонівських методів, так і r-алгоритмів – $DFPR(\alpha)$-алгоритм. Проведено порівняння цього методу з r-алгоритмами на прикладі мінімізації квадратичних функцій, в тому числі сильно яружних. Обговорюються можливі схеми методів такого типу для мінімізації негладких опуклих функцій.

У **розділі 3** запропоновано модифікацію $r(\alpha)$-алгоритму з адаптивним регулюванням кроку за напрямком спуску, яка використовує прискорену реалізацію розтягу простору. Наведено результати обчислювальних експериментів для задач мінімізації опуклих кусково-лінійних та квадратичних яружних функцій з використанням цієї модифікації та класичних варіантів r-алгоритму, проведено порівняння отриманих результатів. Розглянуто модель квантильної регресії, сформульованої як задача безумовної мінімізації опуклої негладкої функції, а також застосування $r(\alpha)$-алгоритму з адаптивним регулюванням кроку для її розв’язання. Розглянуто використання методу BFSG та L-BFGS-B для задачі побудови S-подібної кривої. Запропоновано оптимізаційну модель для задачі побудови S-подібної кривої, яка формулюється як задача мінімізації гладкої функції суми нев’язок з простими двосторонніми обмеженнями на змінні. Наведено результати обчислювальних експериментів для цієї задачі з використанням методів BFGS та L-BFGS-B.

У **розділі 4** розглядаються математичні моделі двох класів задач знаходження пропускних спроможностей дуг відмовостійких мереж, які представлені задачами лінійного, змішаного булевого лінійного та нелінійного програмування з блочною структурою матриці обмежень. У першому класі

задач (задача А) для передачі потоків можуть використовуватись всі можливі шляхи в мережі. У другому класі задач (задача Р) для передачі потоків задіяні тільки шляхи із наперед заданої множини шляхів. Наведено результати обчислювальних експериментів розв'язання булевих задач за допомогою відомої програми Gurobi. Розроблено декомпозиційні методи на основі *r*-алгоритму Шора та показано їх конкурентоспроможність з програмою IPOPT при розв'язанні задач нелінійного програмування.

У **розділі 5** побудовано формулювання двох оптимізаційних задач, призначених для знаходження максимального k-плекса у неорієнтованому графі. Перша задача є квадратичною оптимізаційною задачею, для неї описано застосування техніки лагранжевих двоїстих оцінок. Показано, що її можна отримати із відомої лінійної моделі у формі задачі лінійного булевого програмування, домножуючи обмеження на невід'ємні змінні для кожної із вершин графа. Друга задача є задачею булевого лінійного програмування, для неї описано алгоритм знаходження усіх розв'язків. Розроблено алгоритм пошуку всіх максимальних k-плексів для неорієнтованого графа. В основі алгоритму лежить послідовне додавання до задачі лінійного булевого програмування додаткового обмеження, яке відсікає вже знайдені максимальні k-плекси. Алгоритм реалізовано мовою Octave за допомогою GLPK (GNU Linear Programming Kit).

У **розділі 6** розглядаються три математичні моделі двоетапної транспортної задачі: класичної двоетапної транспортної задачі та задачі з обмеженнями на кількість проміжних пунктів (їм відповідає задача лінійного програмування), а також двоетапної транспортної задачі квадратичного програмування. Наведено AMPL-код для розв'язання двоетапної транспортної задачі лінійного програмування за допомогою сучасного програмного забезпечення для задач лінійного програмування. Наведено та проаналізовано результати розрахунку за допомогою програми Gurobi для двоетапної транспортної задачі, яка має багато розв'язків. Сформульовано двоетапну

транспортну задачу квадратичного програмування та досліджено умови, при яких вона має єдиний розв’язок.

Ключові слова: квазіньютонівські алгоритми, методи з перетворенням простору, ДФП-метод, r-алгоритм, відмовостійка мережа, максимальний k-плекс, двоетапна транспортна задача, Gurobi, AMPL.

ABSTRACT

Suprun A.A. r-Algorithms and quasi-Newton methods in applied non-smooth optimization problems. – Qualifying scientific work as a manuscript.

Dissertation for a Doctor of Philosophy Degree by specialty 113 Applied mathematics. – V.M. Glushkov Institute of Cybernetics of the National Academy of Science of Ukraine. – Kyiv, 2023.

The contents of the dissertation. In introduction the relevance of research topic is substantiated, research purpose and tasks are formulated, research scientific novelty and practical value are explained, and its general description is presented.

Chapter 1 discusses main stages of development of gradient, Newton and quasi-Newton methods, as well as non-smooth optimization methods. The works of domestic and foreign scientists were analyzed: L. Bregman, V. Gershovych, Yu. Yermoliev, I. Yeriomin, M. Zhurbenko, A. Nemirovskyi, B. Polyak, S. Stetsenko, P. Stetsyuk, L. Khachian, N. Shor, D. Yudin, S. Agmon, J.F. Bonnans, S. Boyd, C. Broyden, P. Camerini, A. Conn, W. Davidon, J. Dennis, R. Fletcher, L. Fratta, J.C. Gilbert, D. Goldfarb, I.M. Gould, M. Grötschel, R. Haelterman, F. Kappel, J. Kelley, H. König, A. Kuntsevich, C. Lemarèchal, L. Lovasz, F. Maffioli, J. Morè, T. Motzkin, J. Nocedal, D. Pallaschke, E. Polak, M. J.D. Powell, C.M. Reeves, G. Ribière, I. Schoenberg, A. Schriyver, D. Shanno, P. Toint, S. Wright. Publications of the authors mentioned above describe well-known gradient, Newton and quasi-Newton, as well as subgradient methods and algorithms, aspects of numerical and experimental analysis of algorithms and their application to a wide range of applied problems, as well as the latest algorithms and modifications developed in recent years.

The theoretical foundations of several well-known non-smooth optimization methods are outlined, their development and improvements by various authors is described. Based on the conducted review, conclusions were drawn and tasks for research were set.

In **Chapter 2** H- and B-forms of quasi-Newton methods and the Davidon – Fletcher – Powell method (DFP method) are considered. B-form of quasi-Newton method permits to interpret these methods as gradient methods in the appropriately transformed space of arguments. These methods are compared with r-algorithms, their advantages and disadvantages are highlighted. For minimization of smooth convex functions, a gradient method with a space transformation is constructed, which combines properties of both quasi-Newton methods and r-algorithms – $DFPR(\alpha)$-algorithm. This method is compared with r-algorithms on the example of quadratic functions minimization, including strongly ravine ones. Possible schemes of methods of this type for non-smooth convex functions minimization are discussed.

In **Chapter 3** a modification of the r-algorithm with adaptive step adjustment in the direction of descent is proposed, which uses an accelerated implementation of space dilation. The results of computational experiments for convex piecewise linear and quadratic ravine functions minimization with use of this modification and classical variants of r-algorithm are given, the obtained results were compared. Quantile regression model, formulated as unconditional minimization problem of a convex non-smooth function, and application of r-algorithm with adaptive step adjustment for its solving, are considered. The use of the BFSG and L-BFGS-B methods for S-shaped curve constructing problem is considered. An optimization model is proposed for the problem, which is formulated as a problem of minimizing a smooth function of residuals sum with simple two-side variable constraints. The results of computational experiments for this problem using the BFGS and L-BFGS-B methods are given.

In **Chapter 4** mathematical models of two classes of problems for finding arc bandwidths of fault-tolerant networks are considered, which are formulated as linear, mixed Boolean linear and nonlinear programming problems with block structure of constraint matrix. In the first class of problems (problem A), all possible paths in the network can be used to transfer flows. In the second class of problems (problem P), only paths from a predetermined set of paths are used for flows transfer. The results

of computational experiments for solving Boolean problems using the well-known Gurobi program are given. Decomposition methods have been developed based on Shor's r-algorithm and their competitiveness with the IPOPT program in solving nonlinear programming problems is shown.

In **Chapter 5** two optimization problems designed to find the maximum k-plex in an undirected graph are built. The first problem is a quadratic optimization problem, for which the application of Lagrangian dual estimation technique is described. It is shown that it can be obtained from a known linear model in form of a linear Boolean programming problem by multiplying the constraints by non-negative variables for each vertice of a graph. The second problem is a Boolean linear programming problem, for which an algorithm for finding all the solutions is described. An algorithm for finding all maximal k-plexes for an undirected graph is developed. The algorithm is based on successive addition of an additional constraint to the linear Boolean programming problem, which cuts off already found maximal k-plexes. The algorithm is implemented using the Octave language via GLPK (GNU Linear Programming Kit).

In **Chapter 6** three mathematical models of two-stage transportation problem are considered: classical two-stage transportation problem and the problem with constraints on the number of intermediate points (corresponding to the linear programming problem), as well as two-stage transportation problem of quadratic programming. AMPL code for solving the linear programming two-stage transportation problem using modern linear programming software is given. Calculation results using the Gurobi program for the two-stage transportation problem with many solutions are given and analyzed. The two-stage transportation problem of quadratic programming is formulated, and conditions under which it has a unique solution are investigated.

Key words: quasi-Newton algorithms, methods with space transformation, DFP method, r-algorithm, fault-tolerant network, maximal k-plex, two-stage transportation problem, Gurobi, CPLEX.

Список публікацій здобувача

Публікації, в яких опубліковано основні наукові результати дисертації

1. Стецюк, Петро, Володимир Ляшко, та Антон Супрун. 2020. «Метод BFGS для задачі побудови S-подібної кривої». *Наукові записки НаУКМА*, *Комп'ютерні науки* 3:102–106.
2. Стецюк, Петро, Олексій Лиховид, та Антон Супрун. 2020. «Про лінійну та квадратичну двоетапні транспортні задачі». *Кібернетика та комп'ютерні технології* 4:5–14.
3. Стецюк, Петр, Виктор Стовба, и Антон Супрун. 2021. «В-форма метода Давидона–Флетчера–Пауэлла». *Журнал обчислювальної та прикладної математики* 2(136):93–110.
4. Stetsyuk, Petro, Olha Khomiak, Yehor Blokhin, and Anton Suprun. 2022. "Optimization Problems for the Maximum k-Plex". *Cybernetics and Systems Analysis* 58(4):46–58.
5. Стецюк, Петро, Олексій Лиховид, Володимир Жидков, та Антон Супрун. 2023. Оптимізаційні задачі модернізації пропускних спроможностей дуг відмовостійких мереж. У *Методи негладкої оптимізації в прикладних задачах*, 11–34. Київ: ЛАЗУРИТ ПОЛІГРАФ.
6. Стецюк, Петро, Ольга Хом'як, Олександр Жмуд, та Антон Супрун. 2023. Двоетапна транспортна задача з обмеженням на кількість проміжних пунктів. У *Методи негладкої оптимізації в прикладних задачах*, 68–81. Київ: ЛАЗУРИТ ПОЛІГРАФ.
7. Стецюк, Петро, Ольга Хом'як, та Антон Супрун. 2023. Оптимізаційні задачі для максимального k-плекса. У *Методи негладкої оптимізації в прикладних задачах*, 206–229. Київ: ЛАЗУРИТ ПОЛІГРАФ.

8. Супрун, Антон. 2023. «Обчислювальні експерименти для $r(\alpha)$-алгоритму з прискореною реалізацією розтягу простору». *Кібернетика та комп'ютерні технології* 2:46–54.

Публікації, що засвідчують апробацію матеріалів дисертації

1. Suprun, Anton. 2020. "Computational aspects of quantile regression". Тези доповідей XVIII міжнародної науково-практичної конференції «Математичне та програмне забезпечення інтелектуальних систем (MPZIS-2020)», Дніпро, Листопад 18–20.
2. Стецюк, Петро, та Антон Супрун. 2021. "Прискорення GUROBI та CPLEX для задачі комівояжера". Міжнародний сателітний симпозіум "Інтелектуальні рішення - 2021-С", Київ, Вересень 29.
3. Супрун, Антон, та Андрій Івлічев. 2021. "Інтерактивна програма для задачі побудови та аналізу плоскої кривої з квадратичною кривиною". Міжнародний сателітний симпозіум "Інтелектуальні рішення - 2021-С", Київ, Вересень 29.
4. Супрун, Антон. 2021. "Модифікація r-алгоритму для задачі квантильної регресії". Тези доповідей XIX міжнародної науково-практичної конференції «Математичне та програмне забезпечення інтелектуальних систем (MPZIS-2020)», Дніпро, Листопад 18–20.

ЗМІСТ

ВСТУП

Математична оптимізація є важливою частиною прикладної математики, оскільки за допомогою її методів та моделей можна розв’язувати багато прикладних задач. Дана дисципліна ґрунтується на двох аспектах – розробкою та обґрунтуванням відповідних математичних моделей, що описують досліджувану задачу, та числових методів їх розв’язку. В даній дисертаційній роботі приділяється увага обом цим аспектам – дослідженню двох важливих класів методів оптимізації – квазіньютонівських алгоритмів та субградієнтних алгоритмів з розтягом простору, та дослідженню деяких класів математичних моделей, які можуть застосовуватись для розв’язку широкого класу прикладних задач.

Актуальність теми. Квазіньютонівські алгоритми та субградієнтні алгоритми з розтягом простору є двома класами методів оптимізації, які часто використовуються на практиці. Так, варіант Бройдена – Флетчера – Гольдфарба – Шенно є одним із найбільш застосовуваних для мінімізації нелінійних функцій, а варіант r-алгоритму з адаптивним регулюванням крокового множника вважається одним із найбільш ефективних методів мінімізації негладких функцій.

Актуальність дослідження зв’язку між цими двома класами алгоритмів полягає в тому, що це може допомогти краще зрозуміти і обґрунтувати алгоритми з перетворенням простору і, на їх основі, розробити нові сімейства алгоритмів з розтягом простору, які за своєю ефективністю будуть близькими до r-алгоритму. Розглянуті в дисертації різні оптимізаційні моделі є актуальними, оскільки за допомогою них можна описати і розв’язувати багато прикладних задач. Так, наприклад, за допомогою моделей в (Стецюк, Лиховид, та Супрун 2020), (Стецюк та ін. 2021), (Стецюк та ін. 2022) можна описати задачі побудови або модернізації різних систем мережевої структури, наприклад, системи логістики або енергетичні системи.

Мета й завдання дослідження. Метою роботи є дослідження квазіньютонівських алгоритмів та субградієнтних алгоритмів з розтягом простору, розробка нових градієнтних та субградієнтних алгоритмів з перетворенням простору, розробка нових математичних моделей для розв'язання прикладних задач.

Для досягнення мети дослідження поставлено наступні завдання:

- розробити та обґрунтувати нові форми квазіньютонівських алгоритмів як алгоритмів з відповідною процедурою перетворенням простору змінних;
- розробити та обґрунтувати нові варіанти алгоритмів з класу алгоритмів з перетворенням простору для мінімізації яружних опуклих функцій;
- розробити програмні реалізації нових алгоритмів та провести обчислювальні експерименти для оцінки їх ефективності для задач мінімізації гладких та негладких яружних опуклих функцій;
- розробити нові моделі прикладних задач оптимізації;
- дослідити застосування розроблених методів, алгоритмів та моделей для розв'язання прикладних задач.

Об'єкт дослідження – квазіньютонівські та субградієнтні алгоритми з перетворенням простору, оптимізаційні моделі в прикладних задачах.

Предмет дослідження – алгоритм Давидона – Флетчера – Паувела, r-алгоритм, квантильна регресія, s-подібна крива, мережева та двоетапна транспортна задачі, k-плекс.

Методи дослідження. В дослідженні використовуються методи лінійної алгебри, математичного, негладкого та опуклого аналізу, математичного моделювання, комп'ютерного програмування.

Наукова новизна одержаних результатів. Наукову новизну в цій роботі мають такі теоретичні і практичні результати:

- запропоновано і описано B-форму алгоритму Давидона –Флетчера – Пауела, на її основі запропоновано нове сімейство алгоритмів з розтягом простору;

- запропоновано нове сімейство субградієнтних алгоритмів з розтягом простору у напрямку модифікованої різниці двох субградієнтів у перетвореному просторі, частковим випадком якого є *r*-алгоритм;
- запропоновано оптимізаційну модель для задачі знаходження оцінок параметрів квантильної регресії, модель формулюється як задача безумовної мінімізації кусково-лінійної функції;
- запропоновано оптимізаційну модель для задачі побудови s-подібної кривої. модель формулюється як задача мінімізації гладкої функції суми нев'язок з простими двосторонніми обмеженнями на змінні;
- запропоновано два класи оптимізаційних моделей для задачі побудови відмовостійкої мережі, моделі описуються задачами лінійного, нелінійного та булевого лінійного програмування.

Практичне значення отриманих результатів. Запропоновані в дисертаційній роботі сімейства алгоритмів можуть бути застосовані до задач мінімізації негладких або погано обумовлених опуклих функцій. На основі запропонованих схем алгоритмів можна також розробляти їх нові варіанти та модифікації. Запропоновані в дисертаційній роботі оптимізаційні моделі можуть бути використані для розв'язання відповідних прикладних задач – задачі оцінки параметрів квантильної регресії, задачі побудови контурів профілю різних частин моделей в машинобудуванні з відповідними аеродинамічними властивостями, задачі побудови надійних систем мережевої структури (наприклад, енергетичної системи, логістичної системи, тощо), задачі пошуку максимального k-плекса тощо.

Особистий внесок здобувача. Автором самостійно отримано основні результати дисертаційного дослідження. В опублікованих в співавторстві наукових працях здобувачем здійснено:

- у публікації (Стецюк, Ляшко, та Супрун 2020) – опис оптимізаційної задачі з простими двосторонніми обмеженнями на змінні, застосування

відповідних квазіньютонівських алгоритмів для задачі мінімізації відповідної задачі, аналіз обчислювальних результатів;

- у статті (Стецюк, Лиховид, та Супрун 2020) – формулювання та доведення теореми про єдиність розв'язку двоетапної транспортної задачі при використанні квадратичної регуляризації;
- у статті (Стецюк та ін. 2021) – опис двох класів оптимізаційних задач відмовостійких мереж, проведення обчислювальних експериментів;
- у статті (Стецюк, Стовба, и Супрун 2021) – опис та дослідження властивостей алгоритма Давидона – Флетчера – Пауела в B-формі, опис та дослідження властивостей алгоритма $DFPR(\alpha)$, їх програмні реалізації та проведення обчислювальних експериментів;
- у статті **(**Стецюк та ін. 2022) – опис оптимізаційних моделей задачі пошуку максимального k-плекса;
- у статті (Супрун 2023) – опис нового сімейства субградієнтних алгоритмів з розтягом простору у напрямку модифікованої різниці двох послідовних субградієнтів, програмна реалізація алгоритму, проведення обчислювальних експериментів.

Апробація результатів дисертації. Результати дисертації доповідались та обговорювались на:

- XVIII міжнародній науково-практичній конференції «Математичне та програмне забезпечення інтелектуальних систем (MPZIS-2020)», 18–20 листопада, 2020, м. Дніпро, Україна;
- Міжнародному сателітному симпозіумі «Інтелектуальні рішення – 2021-С», 29 вересня, 2021, м. Київ, Україна;
- фаховому семінарі відділу методів негладкої оптимізації Інституту кібернетики імені В.М. Глушкова НАН України (24 жовтня 2023 року).

Публікації. Основні наукові результати дисертаційної роботи у повній мірі викладено в 13 публікаціях, з яких: 5 наукові статті опубліковано в фахових виданнях України; 1 статтю опубліковано в іноземному виданні,

проіндексованому в наукометричній базі SCOPUS; 3 роботи опубліковано як колективні розділи колективної монографії; 4 тези доповідей опубліковано в збірниках доповідей міжнародних наукових та науково-практичних конференцій і семінарів.

Структура та обсяг дисертації. Дисертаційна робота складається зі вступу, шести розділів, загальних висновків, списку використаних літературних джерел, який містить 100 найменувань. Загальний обсяг дисертаційних досліджень викладено на 178 сторінках друкованого тексту, де обсяг основного тексту – 123 сторінки. Дисертація включає 13 рисунків, 19 таблиць та 2 додатки на 25 сторінках.

РОЗДІЛ 1. ОГЛЯД ЛІТЕРАТУРИ ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Розглянемо задачу мінімізації опуклої функції $f(x)$, $x \in R^n$, x^* – точка мінімуму. Для задачі пошуку точки мінімуму зазвичай використовують ітеративні процедури, які в загальному виді мають таку форму:

$$x_{k+1} = x_k + h_k d_k,\ x_0 \text{ – початкова точка, } k \geq 0. \quad (1.1)$$

Залежно від правил вибору напряму руху d_k, кроку $h_k \geq 0$, та критерію зупинки ітеративного процесу в (1.1), можна задавати відповідні класи алгоритмів мінімізації.

1.1 Градієнтні та квазіньютонівські алгоритми

Якщо функція $f(x)$ неперервно-диференційовна, то в околі точки x_k її можна апроксимувати лінійною функцією:

$$\varphi_k(x) = (g_k, x - x_k) + f(x_k), \quad (1.2)$$

де g_k – градієнт функції f в точці x_k. Мінімізуючи $\varphi_k(x)$ в деякому заданому околі точки x_k, отримуємо градієнтний алгоритм:

$$x_{k+1} = x_k - h_k g_k, \quad (1.3)$$

де h_k – кроковий множник, $h_k \geq 0$. Цей множник може бути обраний сталим $(h_k = h)$ або таким, що задовольняє умову найшвидшого спуску:

$$h_k \approx h_k^* = \arg\min_{h \geq 0} f(x_k - h g_k). \quad (1.4)$$

При виконанні відповідних умов для $f(x)$ такі алгоритми збігаються до точки мінімуму з лінійною швидкістю (Поляк 1983). Для обчислення наближеного мінімуму за напрямком в (1.4) зазвичай застосовують процедури лінійного пошуку, що використовують умови Вольфе (Wolfe 1969), або умови Армійо (Armijo 1966).

Перевагою алгоритмів (1.3) – (1.4) є їх простота та невелика кількість арифметичних операцій на ітерацію. Головним недоліком градієнтних алгоритмів є їх повільна збіжність для погано обумовлених функцій (так звані яружні функції).

Агрегатні градієнтні алгоритми. Для прискорення градієнтного алгоритму були запропоновані різні класи алгоритмів виду (1.1), які використовують додаткову інформацію про функцію з попередніх ітерацій алгоритму.

Так, одним із важливих класів таких алгоритмів є метод спряжених градієнтів, який на кожній ітерації алгоритму використовує два вектори – поточний градієнт g_k, та напрям руху на попередній ітерації d_{k-1} :

$$d_k := -g_k + \gamma_k d_{k-1},\ d_0 = -g_0,\ k \geq 1.$$

Залежно від вибору γ_k задається відповідних варіант алгоритму. Наприклад, в алгоритмі Флетчера – Рівза (Fletcher and Reeves 1964) використовується

$$\gamma_{k-1} = \frac{\|g_k\|^2}{\|g_{k-1}\|^2},$$

а в алгоритмі Полака – Ріб'єра (Polak and Ribière 1969)

$$\gamma_{k-1} = \frac{g_k\left(g_k - g_{k-1}\right)}{\|g_{k-1}\|^2}.$$

Ньютонівські алгоритми. Ще одним способом прискорення алгоритму (1.3) є використання інших класів функцій, що апроксимують $f(x)$. Наприклад, якщо нелінійна функція двічі неперервно диференційовна, то цілком природно спробувати апроксимувати її квадратичною функцією:

$$\varphi_k(x) = \frac{1}{2}\left(G_k\left(x - x_k\right), x - x_k\right) + \left(g_k, x - x_k\right) + f\left(x_k\right). \qquad (1.5)$$

Тут G_k – матриця других похідних (матриця Гессе) в точці x_k.

Мінімізуючи $\varphi_k(x)$, отримаємо ньютонівський алгоритм:

$$x_{k+1} = x_k - h_k G_k^{-1} g_k, \qquad (1.6)$$

Формулу (1.6) також можна отримати, якщо застосувати метод Ньютона – Рафсона для знаходження розв'язку системи рівнянь, що задає умови стаціонарності точки для функції $f(x)$:

$$g_k^i = 0,\ g_k = \left(g_k^1, ..., g_k^n\right).$$

При виконанні відповідних умов для $f(x)$ (Поляк 1983) алгоритм (1.6) має квадратичну швидкість збіжності в околі точки мінімуму.

Очевидно, що головним недоліком ньютонівського алгоритму є необхідність на кожній ітерації обчислювати і зберігати обернену матрицю других похідних.

Квазіньютонівські алгоритми. На противагу недолікам ньютонівського алгоритму в 1960-х роках виник новий клас алгоритмів, ідея якого полягає в тому, щоб замість обчислення оберненої матриці Гессе G_k^{-1} використовувати симетричну додатно визначену матрицю H_k, що її апроксимує: $H_k \approx G_k^{-1}$, $H_k \underset{k\to\infty}{\to} G_k^{-1}$. Такий клас алгоритмів отримав назву квазіньютонівських, загальна схема яких така:

$$x_{k+1} = x_k - h_k H_k g_k, \tag{1.7}$$

$$h_k \approx h_k^* = \arg\min_{h\geq 0}\left\{ f\left(x_k - hH_k g_k\right)\right\}, \tag{1.8}$$

а перерахунок матриці H_k на кожній ітерації здійснюється таким чином:

$$H_{k+1} = H_k + \Delta H_k, \tag{1.9}$$

де ΔH_k – матриця відносно невеликого рангу (зазвичай рангу 1 або 2), побудована на основі поведінки функції на попередніх ітераціях.

Хоча, взагалі кажучи, квазіньютонівським алгоритмом вважається будь який ітеративний процес виду (1.7) – (1.9), зазвичай в оптимізації присутнє додаткове обмеження – так звана квазіньютонівська умова:

$$H_{k+1} y_k = \Delta x_k, \tag{1.10}$$

де $y_k = g_{k+1} - g_k$, $\Delta x_k = x_{k+1} - x_k = -h_k H_k g_k$. Залежно від вибору матриці H_k в (1.9) можна задавати конкретні алгоритми.

Напевне, найбільш відомими квазіньютонівськими алгоритмами є алгоритм Давидона – Флетчера – Пауела (DFP) та алгоритм Бройдена – Флетчера –Гольдфарба – Шенно (BFGS). Обидва алгоритми використовують дворангову матрицю корекції $\Delta H_k = \alpha u_k u_k^T + \beta v_k v_k^T$. Формула перерахунку матриці H_k у DFP-методі задається таким чином:

$$H_{k+1} = H_k - \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k} + h_k \frac{H_k g_k g_k^T H_k}{y_k^T H_k g_k}. \qquad (1.11)$$

Її можна отримати як розв'язок такої задачі:

$$\min_{H^{-1}} \left\| H^{-1} - H_k^{-1} \right\| \text{ при } H^{-1} = \left(H^{-1} \right)^T,\ y_k = H^{-1} \Delta x_k. \qquad (1.12)$$

Історично, алгоритм, який використовував цю формулу, вважається першим квазіньютонівським алгоритмом. Він був запропонований В. Давидоном (Davidon 1959) в 1959 році як альтернатива повільним існуючим на той час алгоритмам. Згодом він був досліджений і популяризований Флетчером та Пауелом (Fletcher and Powell 1963) в 1963 році. Саме поява цього алгоритму дала поштовх до розвитку нового класу алгоритмів нелінійного програмування, який і дістав назву квазіньютонівських алгоритмів.

Формула перерахунку методу BFGS (Broyden 1970), (Fletcher 1970), (Goldfarb 1970), (Shanno 1970) задається так:

$$H_{k+1} = H_k - \frac{H_k g_k y_k^T H_k + H_k y_k g_k^T H_k}{y_k^T H_k g_k} - \left(h_k - \frac{y_k^T H_k y_k}{y_k^T H_k g_k} \right) \frac{H_k g_k g_k^T H_k}{y_k^T H_k g_k}. \qquad (1.13)$$

Її можна отримати як розв'язок такої задачі:

$$\min_{H} \left\| H - H_k \right\| \text{ при } H = H^T,\ H y_k = \Delta x_k. \qquad (1.14)$$

На сьогоднішній день, даний алгоритм вважається найбільш ефективним та найбільш використовуваним квазіньютонівським алгоритмом (Nocedal and Wright 2006).

Ще однією достатньо поширеною формулою перерахунку H є так звана симетрична однорангова корекція (SR1) (Broyden 1967), (Conn, Gould, and Toint 1991):

$$H_{k+1} = H_k - \frac{\left(\Delta x_k - H_k y_k\right)\left(\Delta x_k - H_k y_k\right)^T}{\left(\Delta x_k - H_k y_k\right)^T y_k}. \tag{1.15}$$

Ця формула отримана як розв'язок системи, що складається з обмеження (1.11) та умови $H_{k+1} = H_{k+1}^T$. Основним її недоліком є те, що залежно від знаку знаменника матриця H_{k+1} може бути від'ємно визначеною. Але цю проблему можна обійти, використовуючи замість стандартних процедур пошуку мінімуму за напрямком (1.8) методи довірчих областей (Nocedal and Wright 2006), (Khalfan, Byrd, and Schnabel 1993). Більш детальний огляд квазіньютонівських алгоритмів подано в (Haelterman 2009), (Dennis and Morè 1977). При виконанні стандартних умов для функції $f\left(x\right)$ та використанні умов Вольфе в процедурі (1.8), розглянуті алгоритми збігаються в околі невиродженої точки мінімуму з надлінійною швидкістю (Поляк 1983), (Nocedal and Wright 2006).

1.2 Методи негладкої оптимізації

Субградієнтні алгоритми.

$$x_{k+1} = x_k - h_k g_k, \tag{1.16}$$

де $g_k \in \partial f\left(x_k\right)$ – довільний субградієнт функції f в точці x_k з субдиференціала $\partial f\left(x_k\right)$. Цей метод запропонував Н.З. Шор у своїй кандидатській дисертації 1964 року, застосувавши його до чисельного розв'язку задачі, двоїстої до транспортної задачі лінійного програмування. Пізніше цей підхід був розширений та обґрунтований Б.Т. Поляком в (1967) та (1969), Ю.М. Єрмольєвим (1966), а також узагальнений Н.З. Шором у монографії (1979) (перекладена англійською (Shor 1985)).

Формула субградієнтного алгоритму (1.16) співпадає із відповідною формулою звичайного градієнтного алгоритму (1.3) з заміною градієнта на довільний субградієнт функції в точці. Якщо функція є диференційовною, то субградієнтний метод використовує той же напрямок руху, що і градієнтний метод.

Якщо вибирати крок сталим, тобто $h_k = h$, як в звичайному градієнтному алгоритмі, це може призвести до того, що метод не буде збігатись, оскільки норма субградієнта g_k може не збігатись до нуля в околі точки мінімуму. Так, напрямок $-g_k$ не обов'язково буде напрямком зменшення значення функції (Поляк 1983), (Bonnans et al. 2006). Якщо спробувати обчислити напрямок найшвидшого спуску $-\frac{g_k}{\|g_k\|}$, де $g_k = \underset{\partial f(x_k)}{\text{Proj}}\{0\}$ – найменший за нормою субградієнт в $\partial f(x_k)$, то такий підхід також не спрацьовує. Так, для обчислення g_k необхідно мати повну інформацію про субдиференціал $\partial f(x_k)$, що зазвичай неможливо на практиці. Інша проблема полягає в тому, що такі алгоритми знову ж таки можуть збігатися до нестаціонарних точок (Поляк 1983), (Bonnans et al. 2006). Проблема полягає в тому, що відображення $x \to \partial f(x_k)$ не є неперервним (Шор 1979), (Демьянов и Васильев 1981).

Головна ідея цього методу полягає в тому, що при відповідному виборі крокових множників h_k монотонно спадає не значення функції чи норма градієнта, а відстань до точки мінімуму. Так виникли класичні стратегії вибору крокових множників у субградієнтному методі – обирати таку послідовність кроків, що збігається до нуля, але «не дуже швидко» (Шор 1979), (Shor 1985), (Boyd and Mutapcic 2007):

$$h_k \geq 0,\ h_k \to 0,\ \sum h_k = \infty,\ k \to \infty, \tag{1.17}$$

$$h_k \geq 0,\ \sum h_k^2 < \infty,\ \sum h_k = \infty,\ k \to \infty, \tag{1.18}$$

$$h_k = \frac{\gamma_k}{\|g_k\|},\ \gamma_k \geq 0,\ \sum \gamma_k^2 < \infty,\ \sum \gamma_k = \infty,\ k \to \infty, \tag{1.19}$$

Якщо використовувати сталі крокові множники:

$$h_k = \frac{h}{\|g_k\|},\ k \geq 0, \tag{1.20}$$

$$h_k = h,\ k \geq 0, \tag{1.21}$$

то відповідні алгоритми збігаються до точки, яка лежить в деякому околі точки мінімуму. Зазначимо, що субградієнтні алгоритми, які використовують крокові множники з (1.17) – (1.21) збігаються дуже повільно.

Також використовуються інші стратегії щодо вибору крокового множника:

$$h_k = h_0 2^{-[k/N]}, \tag{1.22}$$

де N – параметр, який залежить від ступеня яружності функції. Субградієнтні алгоритми з таким вибором кроку збігаються з лінійною швидкістю до точки мінімуму (Шор 1979), (Shor 1985). На практиці, алгоритми з таким вибором кроку є ефективними, якщо функція не сильно яружна.

Іншим способом регулювання кроку в субградієнтному методі (1.16) є використання апріорної інформації про мінімальне значення функції f^* (Поляк 1983), (Поляк 1969):

$$h_k = \frac{\left(f(x_k) - f^*\right)}{\|g_k\|^2}. \tag{1.23}$$

Якщо величина f^* невідома, то можна використовувати її оцінку. Він називається класичним феєрівським кроком (або кроком Аґмона – Моцкіна – Шонберга чи кроком Поляка), а відповідний йому метод – класичним феєрівським методом. В математичній оптимізації цей метод також відомий як субградієнтний алгоритм з кроком Поляка. Уперше такий вибір крок в 1954 році незалежно використали Аґмон (Agmon 1954) та Моцкін і Шонберг (Motzkin and Schoenberg 1954) у релаксаційному методі для знаходження хоча б одного розв’язку сумісної системи лінійних нерівностей. Пізніше було запропоновано узагальнення цього методу для системи опуклих нерівностей (Еремин 1965) та знаходження спільної точки опуклих множин (Брэгман 1967).

Такий метод збігається зі лінійною швидкістю (Поляк 1983). Але, незважаючи його на теоретичну прозорість та простоту, він також повільно працює для яружних опуклих функцій (Стецюк 1997).

Для прискорення субградієнтних алгоритмів можна використати додаткову інформацію про функцію на попередніх ітераціях. Так, для прискорення класичного феєрівського методу функцію в околі точки мінімуму можна наблизити кусочно-лінійним функціоналом, використовуючи інформацію попередніх ітерацій – такий метод був запропонований Поляком в (1969):

$$x_{k+1} = \underset{Q_k}{\mathrm{Proj}}\left(x_k\right),\ Q_k = \left\{x:\ f\left(x_i\right) + \left(g_i, x - x_i\right) \le f^*, i \in I_k\right\} \qquad (1.24)$$

де I_k – деяка множина індексів, яка обов'язково містить k. При $I_k = k$ такий алгоритм співпадає зі звичайним субградієнтним алгоритмом з кроком Поляка (1.23), а варіант алгоритму при $I_k = \{0,1,...,k\}$ є ідейно близьким до методу Келлі (Kelley 1960), причому для кусково-лінійної функції він збігається за скінчену кількість кроків до мінімуму функції.

Ще один спосіб прискорення субградієнтних алгоритмів полягає у використанні вектору руху на попередніх ітераціях. Так, варто відзначити один із перших субградієнтних алгоритмів такого типу, який для побудови чергового напрямку руху, використовує лінійну комбінацію двох векторів – субградієнту функції в точці x_k та вектору руху на попередньому кроці (Camerini, Fratta and Maffioli 1975):

$$x_{k+1} := x_k - h_k d_k,\ d_k := g_k + \beta_k d_{k-1}\ , \qquad (1.25)$$

$$\beta_k = \max\left(0, -\gamma_k \frac{d_{k-1}^T g_k}{\left\|d_{k-1}\right\|^2}\right),\ \gamma_k \in [0,2),\ d_0 = g_0,\ k \ge 1. \qquad (1.26)$$

Такий вибір напрямку забезпечує більш «плавний» рух вздовж яру. Але, нажаль, даний алгоритм є також неефективним для функцій з багатовимірним яром (Стецюк 1997).

Загалом, варто зазначити, що попри свою простоту, субградієнтні алгоритми є важливим класом алгоритмів оптимізації. Саме поява

субградієнтних алгоритмів призвела до початку систематичного дослідження методів оптимізації негладких функцій. Такі алгоритми не потребують значної інформації і тому можуть застосовуватись до задач дуже великої розмірності. Із недоліків цього класу алгоритмів можна відмітити відсутність явного критерію зупинки для деяких його варіантів та їх повільну збіжність для яружних погано обумовлених функцій.

Субградієнтні алгоритми з перетворенням простору. Повільна збіжність субградієнтних алгоритмів пов'язана з тим, що напрям спуску з поточної точки та напрям на точку мінімуму функції зазвичай утворюють кут, близький до прямого. Для гладких функцій, для прискорення ітеративного процесу використовують квадратичну апроксимацію. В негладкому випадку такий підхід неможливий, оскільки матриця Гессе може не існувати (наприклад, для кусково-лінійної функції).

Для подолання цієї проблеми, наприкінці 1960-х років, Н. Шор запропонував використовувати неортогональні лінійні оператори перетворення простору. За їх допомогою можна зменшити кут між напрямком спуску та напрямком до точки мінімуму у перетвореному просторі, і таким чином, пришвидшити роботу алгоритму. Для цього був запропонований спеціальний лінійний оператор розтягу простору за напрямком η:

$$R_\alpha(\eta) = I + (\alpha - 1)\eta\eta^T, \tag{1.27}$$

де η – напрям розтягу простору, $\|\eta\| = 1$. На основі цієї ідеї був представлений клас субградієнтних алгоритмів з розтягом простору:

$$x_{k+1} := x_k - h_k B_k B_k^T g_k,\ B_{k+1} = B_k R_{\beta_k}(\eta_k), \tag{1.28}$$

де $R_{\beta_k}(\eta_k)$ – оператор «стиснення» простору в напрямку нормованого вектора η_k з коефіцієнтом «стиснення» простору $\beta_k = 1/\alpha_k < 1$.

Схема (1.28) є так званою B-формою алгоритмів з розтягом простору. Її також можна описати і в еквівалентній до (1.28) H-формі, використовуючи симетричну додатно визначену матрицю $H_k = B_k B_k^T$:

$$x_{k+1} := x_k - h_k \frac{H_k g_k}{\sqrt{g_k^T H_k g_k}}, \quad H_{k+1} = H_k + \left(\beta_k^2 - 1\right)\frac{H_k g_k g_k^T H_k}{\eta_k^T H_k \eta_k}, \quad (1.29)$$

Відзначимо схожість формул в (1.29) з відповідними схемами квазіньютонівських методів в (1.11), (1.13) та (1.15). Залежності від вибору напрямку розтягу простору η_k можна задавати різні сімейства алгоритмів з розтягом простору.

Одним із перших сімейств (1.28) – (1.29) були алгоритми з розтягом простору у напрямку субградієнта: $\eta_k = g_k$. Вважливим алгоритмом цього сімейства є відомий метод еліпсоїдів (Шор 1977). В роботі (1976) Д.Б. Юдін та А.С. Неміровський описали метод еліпсоїдів як варіант методу послідовних відсікань, назвавши його модифікованим методом центрованих відсікань (ММЦВ). А Н. Шор, у свою чергу, представив метод еліпсоїдів як частковий випадок субградієнтних методів з розтягом простору в напрямку субградієнта, вказавши відповідний коефіцієнт розтягу простору та параметри регулювання кроку в напрямку нормованого антисубградієнту такими при яких ітеративний процес збігається з геометричною швидкістю зменшення об’єму еліпсоїда, в якому локалізована точка мінімуму опуклої функції, чим і довів збіжність методу еліпсоїдів.

Цей алгоритм має важливе теоретичне значення. Так, в 1979 році Л.Г. Хачіян, використовуючи метод еліпсоїдів, побудував перший поліноміальний алгоритм розв’язку задачі лінійного програмування з раціональними коефіцієнтами (1979), чим довів існування поліноміальних алгоритмів для задач такого типу. Цей результат став початком інтенсивних досліджень нових практичних алгоритмів для розв’язання задач лінійного програмування. Ще одним важливим наслідком появи методу еліпсоїдів став результат (Grötschel, Lovasz, and Schriyver 1981), в якому автори показали, що більшість комбінаторних задач, для яких були відомі поліноміальні алгоритми, можна розв’язати за поліноміальний час, використовуючи метод еліпсоїдів через перехід до відповідної задачі відокремлення. Прикладами таких задач можуть

бути задачі на мінімальний розріз, оптимізація лінійної функції на перетині двох матроїдів, максимальний зважений незалежний набір ребер графа, задача про китайського листоноші, мінімізація субмодулярної функції на структурі підмножин тощо.

Незважаючи на неспроможність методу еліпсоїдів розв'язувати складні обчислювальні задачі на практиці, його ідея не вичерпала всіх своїх можливостей. Внаслідок цього почали з'являтись численні модифікації цього алгоритма.

В роботах (Шор и Гершович 1979), (Шор и Гершович 1982) та (König and Pallaschke 1981) були зроблені спроби побудувати модифікації методу еліпсоїдів з більшими коефіцієнтами зменшення об'єму на кожному кроці. Для цього використовувались оптимальні еліпсоїди, описані навколо складних опуклих тіл (сегмент, «кульовий» шар та s-піраміда). Ітерація таких методів полягала в переході в центр мінімального за об'ємом еліпсоїда, описаного навколо одного з цих опуклих тіл. Продовженням цих робіт можна вважати статтю (Стецюк и Буханцов 2002), в якій прискорення досягається через використання еліпсоїда мінімального об'єму, що містить кульовий шар. Ідея такої модифікації полягає в тому, що на поточній ітерації використовується відтинаюча гіперплощина з попереднього кроку. Якщо вона дає можливість зменшити об'єм апроксимуючого еліпсоїда – зменшення реалізується, інакше використовується звичайний крок методу еліпсоїдів. При мінімізації суттєво-яружних функцій така модифікація демонструє в декілька разів вищу швидкість збіжності, ніж метод еліпсоїдів Юдіна – Неміровського – Шора.

Другим важливим сімейством алгоритмів з розтягом простору є r-алгоритм, в якому напрям розтягу простору здійснюється у напрямку різниці двох послідовних субградієнтів (Шор и Журбенко 1971), (Шор 1979):

$$R_{\beta_k}(\eta_k) = I + (\beta_k - 1)\eta_k \eta_k^T, \quad \eta_k := \frac{B_k^T r_k}{\left\| B_k^T r_k \right\|}, \quad r_k := g_{k+1} - g_k. \qquad (1.30)$$

Залежно від вибору коефіцієнту розтягу простору α_k в (1.28), процедури пошуку (наближеного) мінімуму функції за напрямком в (1.29) та умов зупинки алгоритма можна отримати різні його варіанти. На практиці, зазвичай, застосовують процедури наближеного пошуку мінімуму за напрямком:

$$h_k \approx h_k^* = \arg\min_{h \geq 0} f\left(x_k - hB_k B_k^T g_k\right). \qquad (1.31)$$

Теоретичні обґрунтування збіжності цього сімейства алгоритмів на даний момент є не повними. Найбільш загальним результатом про їхню збіжність є теорема, доведена в роботі (Шор 1975) для «ідеалізованого» варіанту r-алгоритму – $r_\mu(\alpha)$-алгоритму. Вона стверджує, що для класу майже диференційовних кусково-гладких функцій (Шор 1972) отримані достатні умови, за яких $r_\mu(\alpha)$-алгоритм збігається до локального мінімуму. Однак вони є занадто сильними і, взагалі кажучи, не виконуються, наприклад, для кусково-лінійних функцій. Найбільш типовий випадок, коли це відбувається, пов'язаний з порушенням лінійної незалежності множини $G_f(x)$ майже-градієнтів функції $f(x)$ в точці x. Приклад такої функції наведено в роботі (Стецюк 1995).

Так, одним із ефективних і, напевне, найбільш застосованим на сьогодні варіантом r-алгоритму є $r(\alpha)$-алгоритм з адаптивним регулюванням кроку. В ньому для розтягу простору використовується постійне значення коефіцієнту розтягу простору $\alpha_k = \alpha$, а величина кроку h_k обирається з умови наближеного пошуку мінімуму за напрямком і так, щоб виконувалась умова $h_k > h_k^*$. Такий спосіб регулювання кроку дає змогу зменшити кількість обчислень значення функції $f(x)$ та субградієнта $g(x)$ так, щоб на кожну ітерацію алгоритму припадало не більше 2-3 таких обчислень. Детальний опис такого способу регулювання кроку наведено в роботах (Шор и Стеценко 1989), (Супрун 2023). На його основі, а також на основі його модифікацій, було розроблено низку програмних реалізацій r-алгоритму (Kuntsevich, and Kappel 1997), (Шор та ін. 2003), (Стецюк 2011).

1.3 Постановка завдання дослідження

Метою роботи є дослідження квазіньютонівських алгоритмів та субградієнтних алгоритмів з розтягом простору, розробка нових градієнтних та субградієнтних алгоритмів з перетворенням простору, розробка нових математичних моделей для розв’язання прикладних задач.

Для проведення дослідження необхідно виконати наступні завдання:

1. розробити та обґрунтувати модифікації квазіньютонівських алгоритмів у B-формі;
2. розробити та обґрунтувати варіанти алгоритмів з перетворенням простору, які використовують нові оператори перетворення простору;
3. розробити програмні реалізації запропонованих алгоритмів та провести обчислювальні експерименти для мінімізації гладких та негладких яружних опуклих функцій;
4. розробити моделі прикладних задач оптимізації, які можуть характеризуватися недиференційовністю або яружністю цільових функцій, великою розмірністю, великою кількістю обмежень;
5. застосувати алгоритми з розтягом простору для розв’язання прикладних задач на основі запропонованих математичних моделей.

РОЗДІЛ 2. КВАЗІНЬЮТОНІВСЬКІ МЕТОДИ ТА R-АЛГОРИТМ

Обговорюється спеціальна форма (B-форма) методів квазіньютонівського типу, яка дозволяє легко інтерпретувати ці методи, як градієнтні в перетвореному відповідним чином просторі аргументів. Наведено B-форму методу Давидона – Флетчера – Пауела та на її основі проведено порівняння цього методу з r-алгоритмами. Для мінімізації гладких опуклих функцій побудовано градієнтний метод з перетворенням простору, що поєднує властивості як квазіньютонівських методів, так і r-алгоритмів. Обговорюються можливі схеми методів такого типу для мінімізації негладких опуклих функцій.

2.1 H та B - форми квазіньютонівських методів

Нехай є задача

$$\min f(x), \tag{2.1}$$

де $f(x)$ – опукла двічі диференційовна функція векторного аргументу $x \in X$, $X = \mathbb{R}^n$. $\mathbb{R}^n$ – евклідів простір розмірності n зі скалярним добутком (x, y); x^* – екстремальна точка задачі (2.1).

Нехай початкову точку x_0 обрано в досить малому околі точки мінімуму x^*, тобто $f(x)$ добре апроксимується квадратичною функцією

$$\varphi(x) = \frac{1}{2}\left(\nabla^2 f\left(x^*\right)\left(x - x^*\right), x - x^*\right) + f\left(x^*\right).$$

Тут $\nabla^2 f\left(x^*\right)$ – матриця Гессе в точці мінімуму. Нехай H_0 – симетрична додатно визначена матриця розмірності $n \times n$. Зазвичай $H_0 = I$, де I – одинична матриця.

Методи квазіньютонівського типу в H-формі генерують послідовність точок $\left\{x_k,\, k=\overline{0,n}\right\}$ за таким правилом:

$$x_{k+1}=x_k-h_kH_k\nabla f\left(x_k\right), \tag{2.2}$$

де $\nabla f\left(x_k\right)$ – градієнт функції $f\left(x\right)$ в точці x_k; H_k – симетрична матриця розмірності $n\times n$; h_k – крок, що відповідає мінімуму $f\left(x\right)$ в напрямку $-H_k\nabla f\left(x_k\right)$, тобто

$$h_k=\arg\min_{h\geq 0}\left\{f\left(x_k-hH_k\nabla f\left(x_k\right)\right)\right\}. \tag{2.3}$$

Перерахунок матриці H від кроку до кроку здійснюється таким чином:

$$H_{k+1}=H_k+\Delta H_k, \tag{2.4}$$

де ΔH_k – матриця невеликого рангу, побудована на основі поведінки градієнтів $\nabla f\left(x_k\right)$ та $\nabla f\left(x_{k+1}\right)$ так, щоб $H_n\approx\left[\nabla^2 f\left(x^*\right)\right]^{-1}$. Можливість такого перерахунку забезпечує так звана квазіньютонівська умова

$$H_{k+1}\left(\nabla f\left(x_{k+1}\right)-\nabla f\left(x_k\right)\right)=-h_kH_k\nabla f\left(x_k\right). \tag{2.5}$$

Для квазіньютонівських методів, у яких умову закінчення процесу за n кроків при мінімізації квадратичних функцій відкинуто, умова (2.5) забезпечує виконання умови, що полягає в тому, що власні числа матриці H_k прямують до власних чисел матриці $\left[\nabla^2 f\left(x^*\right)\right]^{-1}$. Фактична різниця між методами квазіньютонівського типу в H-формі полягає у різних формулах перерахунку матриці H_{k+1}, які задовольняють квазіньютонівську умову (2.5) та один з вищенаведених способів наближення H_k до $\left[\nabla^2 f\left(x^*\right)\right]^{-1}$.

Додатно визначену симетричну матрицю H_k завжди можна представити у вигляді $H_k=B_kB_k^T$, де B_k – невироджена матриця розмірності $n\times n$. Співвідношення (2.2), яке реалізує перехід до наступної точки для H-форми квазіньютонівських методів, можна записати так:

$$x_{k+1} = x_k - h_k B_k B_k^T \nabla f\left(x_k\right), \tag{2.6}$$

або

$$y_{k+1} = y_k - h_k B_k^T \nabla f\left(x_k\right) = y_k - h_k \nabla \varphi_k\left(y_k\right), \tag{2.7}$$

де $y_{k+1} = B_k^{-1} x_{k+1}$ та $y_k = B_k^{-1} x_k$ – образи точок x_{k+1} та x_k з X в перетвореному за допомогою лінійного оператора $A_k = B_k^{-1}$ просторі аргументів. $\nabla \varphi_k\left(y_k\right) = B_k^T \nabla f\left(x_k\right)$ – градієнт функції $\varphi_k\left(y\right) = f\left(B_k y\right)$, визначеної в просторі аргументів $Y_k = A_k X$, в точці y_k.

Процес (2.7) можна інтерпретувати як градієнтний метод найшвидшого спуску в перетвореному просторі аргументів $Y_k = A_k X$ для мінімізації опуклої гладкої функції $\varphi_k\left(y\right) = f\left(B_k y\right)$. Співвідношення (2.3), що визначає вибір крокового множника в квазіньютонівських методах, рівносильне такому:

$$h_k = \arg\min_{h \geq 0} \left\{ \varphi_k\left(y_k - h B_k^T \nabla f\left(x_k\right)\right) \right\} = \arg\min_{h \geq 0} \left\{ \varphi_k\left(y_k - h \nabla \varphi_k\left(y_k\right)\right) \right\}, \tag{2.8}$$

де $\varphi_k\left(y\right) = f\left(B_k y\right)$ – функція, визначена в Y_k.

Співвідношення (2.6) – (2.8) у поєднанні з процедурою малорангової корекції матриці B_k дозволяють описувати квазіньютонівські методи у B-формі. При цьому такі методи мають достатньо просту градієнтну природу у перетвореному просторі аргументів. Зауважимо, що, зважаючи на неоднозначність розкладу $H_k = B_k B_k^T$, для конкретного методу в H-формі випливає існування різних методів у B-формі. Але це не так вже й погано, тому що при побудові B-методу легко врахувати й ту умову, яка забезпечує чисельну стійкість методу. Наприклад, при одноранговій корекції матриці $B_{k+1} = B_k T_k$ матрицю T_k вибираємо так, щоб відношення $\lambda_{\max}\left(T_k\right) / \lambda_{\min}\left(T_k\right)$ було якомога меншим. Тут $\lambda_{\max}\left(T_k\right)\left(\lambda_{\min}\left(T_k\right)\right)$ – максимальне (мінімальне) власне число матриці T_k.

Однак чисельна стійкість B-форми методів поступається двом перевагам H-форми: більш економному зберіганню матриці та можливості обійтися

меншою кількістю арифметичних операцій на ітерації. Тому тут потрібен певний компроміс між дослідженням та аналізом методів та їхньою реалізацією на ЕОМ. Таким компромісом може слугувати, наприклад, розробка чисельно стійкого методу квазіньютонівського типу у B-формі, та розгляд його H-форми як наслідку цього методу з метою економії пам'яті та обчислень. Тим більше, що розробці такого методу нічого не заважає. Дійсно, квазіньютонівську умову можна записати так:

$$B_{k+1}B_{k+1}^T\left(\nabla f\left(x_{k+1}\right)-\nabla f\left(x_k\right)\right)=-h_kB_kB_k^T\nabla f\left(x_k\right).$$

Позначимо $\xi=B_k^T\nabla f\left(x_{k+1}\right)-B_k^T\nabla f\left(x_k\right)$ та $\eta=B_k^T\nabla f\left(x_k\right)$ – вектори різниці послідовних градієнтів та поточного градієнта у перетвореному просторі аргументів відповідно. Нехай для перерахунку B_{k+1} використовується однорангова корекція досить загального виду:

$$B_{k+1}=B_k\left(I+t_1\left(\xi+t_2\eta\right)\left(\xi+t_3\eta\right)^T\right),$$

де t_1, t_2, t_3 – невідомі скалярні параметри. Тоді, для виконання квазіньютонівської умови, параметри t_1, t_2, t_3 мають задовольняти таке співвідношення:

$$\left(I+t_1\left(\xi+t_2\eta\right)\left(\xi+t_3\eta\right)^T\right)\left(I+t_1\left(\xi+t_3\eta\right)\left(\xi+t_2\eta\right)^T\right)\xi=-h_k\eta, \qquad (2.9)$$

яке пов’язує вектори ξ та η в перетвореному просторі аргументів.

Зі співвідношення (2.9) випливають два рівняння для трьох невідомих параметрів t_1, t_2 і t_3. Вибір певних із них дає вже відомі методи квазіньютонівського типу. Наприклад, поклавши $t_3=0$ та однозначно визначивши t_1 і t_2, отримаємо ДФП-метод. Але вибір цих параметрів породжує і низку нових методів квазіньютонівського типу, які використовують однорангову корекцію матриці B_k, і цілком можливо, що серед цих методів можна відшукати й такі, що за своєю ефективністю не будуть уступати методу Бройдена – Флетчера – Гольдфарба – Шанно.

2.2 H та B - форми ДФП-методу

Обидві форми ДФП-методу для задачі (2.1) опишемо, припускаючи, що x_0 – початкове наближення з досить малого околу точки мінімуму x^*. H-форму ДФП-методу наведемо згідно з (Полак 1974) з точністю до незначних перепозначень.

H -форма ДФП-методу

Крок 0. Вибрати $x_0 \in \mathbb{R}^n$. Якщо $\nabla f(x_0) = 0$ – зупинитись і покласти $x^* = x_0$. Інакше покласти $H_0 = I$, де I – одинична матриця розмірності $n \times n$, $g_0 = \nabla f(x_0)$, $k = 0$ та перейти до кроку 1.

Крок 1. Покласти

$$\xi_k = H_k g_k . \tag{2.10}$$

Крок 2. Обчислити

$$h_k = \arg\min_{h \geq 0} \left\{ f(x_k - h\xi_k) \right\}. \tag{2.11}$$

Крок 3. Покласти

$$x_{k+1} = x_k - h_k \xi_k . \tag{2.12}$$

Крок 4. Якщо $\nabla f(x_{k+1}) = 0$ – зупинитись і покласти $x^* = x_{k+1}$. Інакше покласти

$$g_{k+1} = \nabla f(x_{k+1}),\ \Delta g_k = g_{k+1} - g_k,\ \Delta x_k = x_{k+1} - x_k , \tag{2.13}$$

$$H_{k+1} = H_k - \frac{H_k \Delta g_k (\Delta g_k)^T H_k}{(\Delta g_k, H_k \Delta g_k)} + \frac{\Delta x_k (\Delta x_k)^T}{(\Delta g_k, \Delta x_k)}. \tag{2.14}$$

Крок 5. Покласти $k = k + 1$ та перейти до кроку 1.

Для опису B-форми ДФП-методу нам знадобляться певні допоміжні результати. Зокрема, використовуючи (2.12) та (2.13), перерахунок матриці H_{k+1}, згідно з (2.14), можна записати так:

$$H_{k+1} = H_k - \frac{H_k \Delta g_k (\Delta g_k)^T H_k}{(\Delta g_k, H_k \Delta g_k)} + h_k \frac{H_k g_k g_k^T H_k}{(\Delta g_k, H_k g_k)} =$$

$$= B_k B_k^T - \frac{B_k B_k^T \Delta g_k \left(\Delta g_k\right)^T B_k B_k^T}{\left(\Delta g_k, B_k B_k^T \Delta g_k\right)} + h_k \frac{B_k B_k^T g_k g_k^T B_k B_k^T}{\left(\Delta g_k, B_k B_k^T g_k\right)} = \qquad (2.15)$$

$$= B_k \left(I - \xi_k \xi_k^T + t_k^2 \eta_k \eta_k^T\right) B_k^T = B_{k+1} B_{k+1}^T,$$

де $\xi_k = \frac{B_k^T g_{k+1} - B_k^T g_k}{\left\| B_k^T g_{k+1} - B_k^T g_k \right\|}$, $\eta_k = \frac{B_k^T g_k}{\left\| B_k^T g_k \right\|}$ та $t_k^2 = h_k \frac{\left\| B_k^T g_k \right\|^2}{\left(B_k^T g_k, B_k^T g_k - B_k^T g_{k+1}\right)}$.

Додатність параметра t_k^2 забезпечується вибором h_k з умови найшвидшого спуску для гладкої функції в перетвореному просторі аргументів. При точній реалізації найшвидшого спуску $t_k^2 = h_k$, оскільки $\left(B_k^T g_k, B_k^T g_{k+1}\right) = 0$. Для забезпечення перерахунку матриці H_{k+1} як у (2.15), достатньо коректувати $B_{k+1} = B_k T_k$, де $T_k = \left(I - \left(\xi_k + t_k \eta_k\right) \xi_k^T\right)$. Насправді при такій корекції B_{k+1} для $T_k T_k^T$ маємо:

$$T_k T_k^T = \left(I - \left(\xi_k + t_k \eta_k\right) \xi_k^T\right)\left(I - \xi_k \left(\xi_k + t_k \eta_k\right)^T\right) =$$

$$= I - \left(\xi_k + t_k \eta_k\right) \xi_k^T - \xi_k \left(\xi_k + t_k \eta_k\right)^T + \left(\xi_k + t_k \eta_k\right)\left(\xi_k + t_k \eta_k\right)^T =$$

$$= I - \xi_k \xi_k^T + t_k^2 \eta_k \eta_k^2,$$

що забезпечує середній множник у правій частині (2.15).

Отже, для однорангової корекції матриці B_{k+1} підходить така формула:

$$B_{k+1} = B_k \left(I - \left(\frac{g_{k+1} - g_k}{\left\| g_{k+1} - g_k \right\|} + \sqrt{h_k \frac{\left\| g_k \right\|^2}{\left(g_k, g_k - g_{k+1}\right)}} \frac{g_k}{\left\| g_k \right\|} \right) \left(\frac{g_{k+1} - g_k}{\left\| g_{k+1} - g_k \right\|} \right)^T \right), \quad (2.16)$$

де g_{k+1} та g_k – субградієнти функції $\varphi_k(y) = f\left(B_k y\right)$ в точках $y_{k+1} = B_k^{-1} x_{k+1}$ та $y_k = B_k^{-1} x_k$ відповідно. Враховуючи (2.16), B-форма ДФП-методу набуває такого вигляду.

B-форма ДФП-методу

Крок 0. Вибрати $x_0 \in \mathbb{R}^n$. Якщо $\nabla f(x_0) = 0$ – зупинитись і покласти $x^* = x_0$. Інакше покласти $B_0 = I$, де I – одинична матриця розмірності $n \times n$, $g_0 = \nabla f(x_0)$, $k = 0$ та перейти до кроку 1.

Крок 1. Покласти

$$g_k = B_k^T g_k . \tag{2.17}$$

Крок 2. Покласти

$$\xi_k = B_k g_k . \tag{2.18}$$

Крок 3. Обчислити

$$h_k = \arg\min_{h \geq 0} \left\{ f(x_k - h\xi_k) \right\}. \tag{2.19}$$

Крок 4. Покласти

$$x_{k+1} = x_k - h_k \eta_k . \tag{2.20}$$

Крок 5. Якщо $\nabla f(x_{k+1}) = 0$ – зупинитись і покласти $x^* = x_{k+1}$. Інакше покласти $g_{k+1} = \nabla f(x_{k+1})$, $g_{k+1} = B_k^T g_{k+1}$,

$$\xi_k = \frac{g_{k+1} - g_k}{\left\| g_{k+1} - g_k \right\|} + \sqrt{h_k \frac{\left\| g_k \right\|^2}{(g_k, g_k - g_{k+1})}} \frac{g_k}{\left\| g_k \right\|}, \quad \eta_k = \frac{g_{k+1} - g_k}{\left\| g_{k+1} - g_k \right\|}, \tag{2.21}$$

$$B_{k+1} = B_k \left(I - \xi_k \eta_k^T \right). \tag{2.22}$$

Крок 6. Покласти $k = k + 1$ та перейти до кроку 1.

Наведений ДФП-метод у B-формі (2.17) – (2.22) не є оптимальним за використанням арифметичних операцій. Його можна покращити, прибравши операцію множення матриці на вектор в (2.17) за рахунок того, що $B_{k+1}^T g_{k+1} = \left(I - \eta_k \xi_k^T \right) B_k^T g_{k+1} = g_{k+1} - \eta_k (\xi_k, g_{k+1})$. Але навіть в такому випадку він буде поступатись ДФП-методу в H-формі (2.10) – (2.14) за кількістю арифметичних операцій, хоча цей розрив буде незначним ($4n^2$ множень як порівняти з $3n^2$ множеннями).

Тим не менш B-форма дозволяє прояснити низку моментів для ДФП-методу. По-перше, додатна визначеність матриці $H_{k+1} = B_{k+1}B_{k+1}^T$ є наслідком невиродженості матриці B_{k+1}, якщо матриця B_k – невироджена. Насправді, визначник матриці B_{k+1} буде таким:

$$\det\left(B_{k+1}\right) = \det\left(B_k\left(I - \xi_k\eta_k^T\right)\right) = \det\left(B_k\right)\left(1 - \left(\xi_k, \eta_k\right)\right) =$$

$$= \det\left(B_k\right)\left(1 - \left(\frac{g_{k+1} - g_k}{\left\|g_{k+1} - g_k\right\|} + \sqrt{h_k \frac{\left\|g_k\right\|^2}{\left(g_k, g_k - g_{k+1}\right)}}\frac{g_k}{\left\|g_k\right\|}, \frac{g_{k+1} - g_k}{\left\|g_{k+1} - g_k\right\|}\right)\right) =$$

$$= \det\left(B_k\right)\sqrt{h_k \frac{\left\|g_k\right\|^2}{\left(g_k, g_k - g_{k+1}\right)}}\frac{\left(g_k, g_k - g_{k+1}\right)}{\left\|g_{k+1} - g_k\right\| \cdot \left\|g_k\right\|} = \det\left(B_k\right)\sqrt{h_k \frac{\left(g_k, g_k - g_{k+1}\right)}{\left\|g_{k+1} - g_k\right\|^2}} \neq 0.$$

При точному найшвидшому спуску

$$\det\left(B_{k+1}\right) = \det\left(B_k\right)\sqrt{h_k \frac{\left\|g_k\right\|^2}{\left\|g_k\right\|^2 + \left\|g_{k+1}\right\|^2}}$$

та він рівний нулю тоді, коли або $h_k = 0$, або $\left\|g_k\right\| = 0$, що для гладких опуклих функцій означає виконання достатньої умови оптимальності в перетвореному просторі аргументів.

По-друге, параметр $\Delta_k = \frac{\left(g_k, g_{k+1}\right)}{\left\|g_k\right\|^2} = \frac{\left(g_k, H_k g_{k+1}\right)}{\left(g_k, H_k g_k\right)}$, який випливає з (2.21) при $\frac{g_k}{\left\|g_k\right\|}$, задає точність виконання умови найшвидшого спуску в перетвореному просторі аргументів. Цю або близьку до неї умову доцільно використовувати в квазіньютонівських методах з наближеним обчисленням мінімуму функції за напрямком як критерій зупинки при одновимірному спуску за напрямком.

По-третє, Δ_k в комбінації з кроком найшвидшого спуску h_k розумно використовувати для процедури повторного старту методів квазіньютонівського типу, враховуючи, що в процесі перерахунку матриці B_{k+1} можливе накопичення помилок. Саме ці параметри та в більшій мірі крок найшвидшого спуску h_k впливають на точність перерахунку матриці B_{k+1}.

2.3 ДФП-метод та *r*-алгоритми

Для r-алгоритмів (Шор 1979), (Shor 1998), (Стецюк 2017) підкреслено їхню «близькість» до ДФП-методу в плані перерахунку матриці H_{k+1}. Використовуючи B-форму ДФП-методу, можна інтерпретувати цю «близькість» більш змістовно шляхом аналізу того, що відбувається в перетвореному просторі аргументів. Виконаємо це для ДФП-методу (2.17) – (2.22) і тих варіантів r-алгоритмів, які використовують точний пошук мінімуму функції за напрямком – $r_\mu(\alpha)$-алгоритм ($\alpha \in [2,4]$) та граничний варіант r-алгоритму ($\alpha = \infty$) (Шор 1979). Як характеристику для порівняння виберемо зміну кута між двома послідовними субградієнтами, яка характерна для однієї ітерації цих методів при переході з Y_k в Y_{k+1}. Очевидно, такі порівняння справедливі лише для гладких функцій.

Нехай g_k та g_{k+1} – субградієнти функції $\varphi_k(y) = f(B_k y)$, визначеної в перетвореному просторі аргументів $Y_k = B_k^{-1} X$, отримані згідно з точним кроком найшвидшого спуску h_k^* в просторі аргументів Y_k. Тоді їх образи g_k та g_{k+1} в перетвореному просторі $Y_{k+1} = B_{k+1}^{-1} X$, враховуючи, що $(g_k, g_{k+1}) = 0$, будуть

$$g_k = \left(I - \frac{g_{k+1} - g_k}{\|g_{k+1} - g_k\|} \left(\frac{g_{k+1} - g_k}{\|g_{k+1} - g_k\|} + \sqrt{h_k^*} \frac{g_k}{\|g_k\|} \right)^T \right) g_k =$$

$$= g_k + \left(\frac{\|g_k\|^2}{\|g_{k+1}\|^2 + \|g_k\|^2} - \sqrt{h_k^* \frac{\|g_k\|^2}{\|g_{k+1}\|^2 + \|g_k\|^2}} \right)(g_{k+1} - g_k) =$$

$$= \left(\frac{\|g_{k+1}\|^2}{\|g_{k+1}\|^2 + \|g_k\|^2} + \sqrt{h_k^* \frac{\|g_k\|^2}{\|g_{k+1}\|^2 + \|g_k\|^2}} \right) g_k +$$

$$+ \left(\frac{\|g_k\|^2}{\|g_{k+1}\|^2 + \|g_k\|^2} - \sqrt{h_k^* \frac{\|g_k\|^2}{\|g_{k+1}\|^2 + \|g_k\|^2}} \right) g_{k+1}, \qquad (2.23)$$

$$g_{k+1} = \left(I - \frac{g_{k+1} - g_k}{\|g_{k+1} - g_k\|} \left(\frac{g_{k+1} - g_k}{\|g_{k+1} - g_k\|} + \sqrt{h_k^*} \frac{g_k}{\|g_k\|} \right)^T \right) g_{k+1} =$$

$$= g_{k+1} - \frac{\|g_{k+1}\|^2}{\|g_{k+1}\|^2 + \|g_k\|^2}(g_{k+1} - g_k) = \frac{\|g_{k+1}\|^2}{\|g_{k+1}\|^2 + \|g_k\|^2} g_k + \frac{\|g_k\|^2}{\|g_{k+1}\|^2 + \|g_k\|^2} g_{k+1}. \qquad (2.24)$$

З (2.23), (2.24) випливає, що косинус кута ψ_k між векторами g_k та g_{k+1} буде

$$\cos\psi_k = \left(\frac{g_k}{\|g_k\|}, \frac{g_{k+1}}{\|g_{k+1}\|} \right) = 1 \Big/ \sqrt{1 + h_k \frac{\|g_k\|^2 + \|g_{k+1}\|^2}{\|g_k\|^2}}\,. \qquad (2.25)$$

Отже, один крок ДФП-методу з Y_k до Y_{k+1} призводить до зменшення кута між послідовними субградієнтами від $\pi/2$ до гострого ψ_k, який визначається з умови (2.25). Ця властивість ДФП-методу характерна і для r-алгоритмів, побудованих на ідеї «розширити» конус можливих напрямків спадання функції шляхом операції розтягу простору аргументів в напрямку різниці двох послідовних субградієнтів.

З (2.25) випливає, що коли $h_k \ll 1$, тоді ітерація ДФП-методу близька до ітерації граничного варіанту r-алгоритмів, який перетворює субградієнти g_k та g_{k+1} так, щоб у перетвореному просторі $\cos\psi_k = 1$. Цей же факт випливає з (2.16), оскільки $h_k \ll 1$, то операція перетворення простору в ДФП-методів буде близькою до розтягу простору з дуже великим коефіцієнтом в напрямку різниці двох послідовних субградієнтів. Така ситуація буде мати місце для сильно

яружних функцій, коли ітераційний процес знаходиться в точці, близькій до «дна» яру.

Для $r_\mu(\alpha)$-алгоритму, який використовує постійний коефіцієнт розтягу простору в напрямку $g_{k+1}-g_k$, характерне перетворення прямого кута в гострий ψ_k, косинус якого рівний

$$\cos\psi_k=\left(\alpha^2-1\right)\Big/\sqrt{\alpha^4+\alpha^2\left(\frac{\|g_k\|^2}{\|g_{k+1}\|^2}+\frac{\|g_{k+1}\|^2}{\|g_k\|^2}\right)+1}\le 1-\frac{2}{\alpha^2+1}.$$

Тут перетворення кута між послідовними градієнтами прямо не пов'язано з h_k^*, хоча непрямий зв'язок забезпечує співвідношення між нормами векторів g_k та g_{k+1}.

Тому для гладких функцій перетворення, що використовується в ДФП-методі, виглядає більш логічним в тому сенсі, що коли крок найшвидшого спуску великий, то слабше «розтягується» конус підходящих напрямків спадання функції $f(x)$, а коли крок найшвидшого спуску малий, то він тягне за собою більш сильний «розтяг» конусу. Але для негладких функцій, де h_k може бути рівним нулю, таке перетворення загалом не застосовне.

Тим не менш, цю обставину можна використовувати для модифікації $r_\mu(\alpha)$-алгоритму, щоб розширити область його застосування для класу майже диференційовних функцій. Насправді, «пастки» для мінімізуючої послідовності $r_\mu(\alpha)$-алгоритму (Стецюк 2014, с. 110–115) базуються на використанні ним постійного коефіцієнту розтягу простору. Замінивши на кроці $r_\mu(\alpha)$-алгоритму постійний коефіцієнт розтягу на змінний, який залежить від параметрів h_k^*, $\|g_k\|$ та $\|g_{k+1}\|$, для такого варіанту r-алгоритмів цілком можливо забезпечити вихід з «поганих» кутових точок для майже диференційовних функцій, при цьому зберігаючи для нього максимальну «близькість» до $r_\mu(\alpha)$-алгоритму.

2.4 DFPR(α)-алгоритм

Для задачі (2.1), за умови, що ми вміємо точно реалізовувати процедуру найшвидшого спуску, побудуємо метод «змінної метрики» – проміжний між ДФП-методом та $r_\mu(\alpha)$-алгоритмом. При цьому збережемо переваги як першого (вибір наступного напрямку руху), так і другого (метод можна перенести на негладкий випадок).

Нехай g_k та g_{k+1} – градієнти $\varphi_k(y) = f(B_k y)$ в точках y_k та y_{k+1}. Тут y_{k+1} отримана згідно з кроком найшвидшого спуску в напрямку $-g_k$ з точки y_k в просторі аргументів $Y_k = B_k^{-1} X$. Тоді $(g_k, g_{k+1}) = 0$. Перетворення простору з Y_k до Y_{k+1}, що лежить в основі ДФП-методу, має одну «чудову» властивість – незалежно від параметра t_k, рух у просторі Y_k, що відповідає напрямку образу градієнта g_{k+1} в Y_{k+1}, виконується по найкоротшому вектору опуклої комбінації g_k та g_{k+1}. Насправді, напрямок руху в Y_k, який відповідає руху по g_{k+1} в Y_{k+1} визначається:

$$p_k = T_k(g_k, g_{k+1}) T_k^T(g_k, g_{k+1}) g_{k+1} =$$

$$= \left(I - \left(\frac{g_{k+1} - g_k}{\|g_{k+1} - g_k\|} + t_k \frac{g_k}{\|g_k\|} \right) \left(\frac{g_{k+1} - g_k}{\|g_{k+1} - g_k\|} \right)^T \right) \left(I - \left(\frac{g_{k+1} - g_k}{\|g_{k+1} - g_k\|} \right) \left(\frac{g_{k+1} - g_k}{\|g_{k+1} - g_k\|} + t_k \frac{g_k}{\|g_k\|} \right)^T \right) g_{k+1} =$$

$$= \left(I - \left(\frac{g_{k+1} - g_k}{\|g_{k+1} - g_k\|} \right) \left(\frac{g_{k+1} - g_k}{\|g_{k+1} - g_k\|} \right)^T + t_k^2 \frac{g_k}{\|g_k\|} \frac{g_k}{\|g_k\|}^T \right) g_{k+1} =$$

$$= g_{k+1} - (g_{k+1} - g_k) \frac{\|g_{k+1}\|^2}{\|g_{k+1}\|^2 + \|g_k\|^2} = \frac{\|g_{k+1}\|^2}{\|g_{k+1}\|^2 + \|g_k\|^2} g_k + \frac{\|g_k\|^2}{\|g_{k+1}\|^2 + \|g_k\|^2} g_{k+1}, \quad (2.26)$$

оскільки $(g_k, g_{k+1}) = 0$. Вектор p_k, який визначається згідно з (2.26), є розв'язком такої задачі:

$$\min \frac{1}{2} \|\lambda_1 g_k + \lambda_2 g_{k+1}\|^2, \quad (2.27)$$

$$(g_k, g_{k+1}) = 0; \qquad \lambda_1 + \lambda_2 = 1; \qquad \lambda_1 \geq 1; \qquad \lambda_1 \geq 1. \tag{2.28}$$

Саме тому в основу методу доцільно покласти перетворення, на якому базується ДФП-метод, оскільки воно забезпечує досить успішний напрямок руху на кроці в Y_k. Так, для квадратичних функцій $f(x)$ з розмірністю простору $n = 2$ такий напрямок руху з точки найшвидшого спуску задаватиме точний напрямок на мінімум незалежно від початкової точки. Тому для таких функцій при довільному виборі параметра t_k метод потребуватиме не більше ніж два кроки найшвидшого спуску.

Однак, збіжність методу за n кроків для квадратичних функцій при $n \geq 3$ (аналогічно до ДФП-методу) забезпечуватиметься далеко не кожен вибір параметра t_k. Тому відкинемо умову завершення процесу мінімізації за n кроків для квадратичних функцій та на вибір t_k накладемо умову, що метод має наближатись до $r_\mu(\alpha)$-алгоритму в тому сенсі, що перетворення простору було «стискаючим» простір субградієнтів так, як це відбувається в $r_\mu(\alpha)$-алгоритмі, тобто

$$\det(B_{k+1}) = \frac{1}{\alpha}\det(B_k), \qquad \alpha > 1, \qquad \alpha \in [2,4].$$

Для цього достатньо покласти $t_k = \frac{1}{\alpha}\frac{\|g_{k+1} - g_k\|}{\|g_k\|}$ і для корекції матриці $B_{k+1} = B_k T_k(g_k, g_{k+1})$ обрати

$$T_k(g_k, g_{k+1}) = I - \left(\frac{g_{k+1} - g_k}{\|g_{k+1} - g_k\|} + \frac{1}{\alpha}\frac{\|g_{k+1} - g_k\|}{\|g_k\|}\frac{g_k}{\|g_k\|}\right)\left(\frac{g_{k+1} - g_k}{\|g_{k+1} - g_k\|}\right)^T, \tag{2.29}$$

оскільки

$$\det(T_k(g_k, g_{k+1})) = 1 - \left(\frac{g_{k+1} - g_k}{\|g_{k+1} - g_k\|} + \frac{1}{\alpha}\frac{\|g_{k+1} - g_k\|}{\|g_k\|}\frac{g_k}{\|g_k\|}, \frac{g_{k+1} - g_k}{\|g_{k+1} - g_k\|}\right) =$$

$$= 1 - 1 - \frac{1}{\alpha}\frac{\|g_{k+1} - g_k\|}{\|g_k\|}\left(\frac{g_k}{\|g_k\|}, \frac{g_{k+1} - g_k}{\|g_{k+1} - g_k\|}\right) = \frac{1}{\alpha}\frac{\|g_{k+1} - g_k\|}{\|g_k\|}\frac{\|g_k\|}{\|g_{k+1} - g_k\|} = \frac{1}{\alpha}.$$

З вищенаведених міркувань отримуємо такий метод «змінної метрики» (будемо називати його DFPR(α)-алгоритмом) для розв'язання задачі (2.1).

DFPR(α)-алгоритм

Крок 0. Перед початком обчислень маємо $\alpha > 1$, $\varepsilon_g > 0$, $x_0 \in R^n$, $B_0 = I_n$ – одинична матриця розмірності $n \times n$, $g_0 = \nabla f(x_0)$. Тут ε_g – досить мале число, яке задає критерій зупинки за нормою градієнта.

Якщо $\|g_0\| \le \varepsilon_g$, то x_0 – шукана точка та зупинка. Інакше переходимо до наступного кроку.

Нехай на k-му кроці отримано $x_k \in R^n$, $g_k \in R^n$, B_k – матриця $n \times n$. Тут $g_k = \nabla f(x_k)$ – градієнт функції $f(x)$ в точці x_k. Тоді $(k+1)$-й крок характеризується такою послідовністю операцій.

Крок 1. Покласти $g_k = B_k^T g_k$.

Крок 2. Покласти $\xi_k = B_k g_k$.

Крок 3. Обчислити $h_k = \arg\min\limits_{h \ge 0} \{ f(x_k - h\xi_k) \}$.

Крок 4. Обчислити чергове наближення $x_{k+1} = x_k - h_k \eta_k$.

Крок 5. Покласти $g_{k+1} = \nabla f(x_{k+1})$. Якщо $\|g_{k+1}\| \le \varepsilon_g$, то зупинка і x_{k+1} – шукана точка. Інакше обчислимо

$$g_{k+1} = B_k^T g_{k+1}, \qquad t_k = \frac{1}{\alpha}\sqrt{1 + \frac{\|g_{k+1}\|^2}{\|g_k\|^2}},$$

$$\eta_1 = \frac{g_{k+1} - g_k}{\|g_{k+1} - g_k\|} + t_k \frac{g_k}{\|g_k\|}, \qquad \eta_2 = \frac{g_{k+1} - g_k}{\|g_{k+1} - g_k\|},$$

$$B_{k+1} = B_k \left(I - \eta_1 \eta_2^T \right).$$

Крок 6. Переходимо до наступного кроку з x_{k+1}, B_{k+1}, g_{k+1}.

Зрозуміло, що збіжність DFPR(α)-алгоритму до розв'язку x^* для задачі (2.1) забезпечуватиме процедура найшвидшого спуску в напрямку антиградієнта. Крім того, він, як і ДФП-метод та r-алгоритми, використовує перетворення простору для того, щоб розширити конус можливих напрямків спадання функції на наступному кроці методу. Насправді, кут між послідовними градієнтами в Y_{k+1} визначається таким співвідношенням:

$$\cos\psi_k = 1\Big/\sqrt{1+\left(\frac{1}{\alpha}\left(\frac{\|g_k\|}{\|g_{k+1}\|}+\frac{\|g_{k+1}\|}{\|g_k\|}\right)\right)^2}, \qquad (2.30)$$

і чим більше α, тим більше він буде зменшуватись.

Якщо $\alpha=\infty$, то DFPR(α)-алгоритм є рівносильним граничному варіанту r-алгоритмів і для квадратичних функцій забезпечуватиме збіжність до x^* за n кроків. З (2.30) маємо, що якщо в DFPR(α)-алгоритмі постійний коефіцієнт α замінити на змінний $\alpha_k=\frac{\|g_k\|}{\|g_{k+1}\|}+\frac{\|g_{k+1}\|}{\|g_k\|}\geq 2$, то кут між послідовними градієнтами зменшуватиметься рівно вдвічі, тобто з $\pi/2$ до $\pi/4$.

Ще одну цікаву модифікацію DFPR(α)-алгоритму забезпечує такий вибір α_k:

$$\alpha_k=\frac{\|g_{k+1}\|}{\|g_k\|}\left(\frac{\|g_k\|}{\|g_{k+1}\|}+\frac{\|g_{k+1}\|}{\|g_k\|}\right)=1+\frac{\|g_{k+1}\|^2}{\|g_k\|^2}>1,$$

при якому кут між послідовними градієнтами зменшуватиметься згідно зі співвідношенням

$$\cos\psi_k = 1\Big/\sqrt{1+\frac{\|g_{k+1}\|^2}{\|g_k\|^2}}.$$

При такому виборі α_k дуже простого виду набувають як однорангова матриця оберненого перетворення $T_k\left(g_k, g_{k+1}\right)$

$$T_k\left(g_k, g_{k+1}\right) = I - \frac{1}{\left\|g_{k+1} - g_k\right\|} g_{k+1}\left(g_{k+1} - g_k\right)^T,$$

так і матриця перетворення простору аргументів

$$T_k^{-1}\left(g_k, g_{k+1}\right) = I - \frac{1}{\left\|g_k\right\|^2} g_{k+1}\left(g_{k+1} - g_k\right)^T.$$

Враховуючи, що перетворення простору спрямовано на зменшення кута між двома послідовними градієнтами, DFPR(α)-алгоритм має бути ефективним для мінімізації гладких яружних функцій в широкому діапазоні значень параметра α, в тому числі при $\alpha \in [2,4]$. Підтвердженням цього можуть слугувати наведені в таблицях 1 та 2 результати чисельних експериментів для мінімізації квадратичних функцій

$$f(x) = \frac{1}{2} x^T A x = \frac{1}{2} \sum_i^n q^{i-1} x_i^2 = Quad(q,n),$$

в тому числі й сильно яружних. Для порівняння тут наведено також результати роботи $r_\mu(\alpha)$-алгоритму за тих же значень α та параметрі $\mu = 0$ ($r_0(\alpha)$-алгоритм). Обидва методи працювали в однакових умовах: початкова стартова точка $x_0 = (1,1,...,1)$; критерій зупинки – $\varepsilon_g = 10^{-10}$; крок найшвидшого спуску обчислювався аналітично – $h_k = \frac{(g_k, \xi_k)}{(A\xi_k, \xi_k)}$ і при цьому точність виконання найшвидшого спуску в Y_k була досить високою, тобто $\left|\left(\frac{g_k}{\|g_k\|}, \frac{g_{k+1}}{\|g_{k+1}\|}\right)\right| < 10^{-14}$.

Отже, можна вважати, що результати, наведені в таблицях 2.1 та 2.2, показують, наскільки перетворення типу ДФП-методу краще, ніж розтяг простору в напрямку різниці двох послідовних градієнтів, другий з яких отримано згідно з кроком найшвидшого спуску в перетвореному просторі аргументів. Для несильно яружних квадратичних функцій $\left(Quad(1.1, 70), Quad(1.2, 50)\right)$ робота dpfr(α)-алгоритму при невеликих α

близька до роботи ДФП-методу в тому сенсі, що забезпечує збіжність за кількість кроків, дуже близьку до n.

Таблиця 2.1

Числові експерименти при малих значеннях α (Стецюк 1996)

$Quad(q,n)$	DFPR(α)-алгоритм			$r_0(\alpha)$-алгоритм		
	$\alpha=2$	$\alpha=3$	$\alpha=4$	$\alpha=2$	$\alpha=3$	$\alpha=4$
$Quad(1.1,200)$	732	581	508	1168	885	775
$Quad(1.1,130)$	288	241	218	496	419	398
$Quad(1.1,70)$	88	79	74	178	176	183
$Quad(1.2,100)$	337	273	239	627	494	457
$Quad(1.2,50)$	80	69	66	191	176	173
$Quad(2.0,30)$	103	83	76	273	218	206

Звісно, розрив у кількості ітерацій між dpfr(α)-алгоритмом та $r_0(\alpha)$-алгоритмом має зменшуватись при збільшенні α, але це має місце при дуже великих значеннях α. Так, при $\alpha=1000$ (таблиця 2.2), яке не можна вважати малим, цей розрив досі є досить великим. Якщо dpfr(α)-алгоритм гарантує збіжність до x^* практично за n кроків, то для $r_0(\alpha)$-алгоритмом це не так. Однією з причин такої поведінки методів є занадто велика точність розв'язання задач $\varepsilon_g=10^{-10}$, що рівносильно $f(x_k)-f^*<10^{-20}$. У випадку менш жорсткого критерію зупинки розрив буде меншим, але й кількість ітерацій для DFPR(α)-алгоритму буде меншою. Але така точність для функції, що мінімізується, свідчить про стійкість DFPR(α)-алгоритму до точності розв'язку навіть для сильно яружних задач.

Таблиця 2.2

Числові експерименти при великих значеннях α (Стецюк 1996)

$Quad(q,n)$	DFPR(α)-алгоритм			$r_0(\alpha)$-алгоритм		
	$\alpha=2$	$\alpha=3$	$\alpha=4$	$\alpha=2$	$\alpha=3$	$\alpha=4$
$Quad(1.1,200)$	379	271	221	702	692	550

$Quad(1.1,130)$	177	131	130	391	384	290
$Quad(1.1,70)$	70	70	70	212	177	140
$Quad(1.2,100)$	181	133	107	422	365	276
$Quad(1.2,50)$	54	50	50	185	143	106
$Quad(2.0,30)$	58	42	36	178	120	87

Отже, для погано обумовлених задач DFPR(α)-алгоритм можна вважати більш стійким, ніж $r_0(\alpha)$-алгоритм, точніше – перетворення простору типу ДФП-методу варто признати більш раціональним, ніж розтяг простору в напрямку різниці двох послідовних градієнтів.

2.5 Про субградієнтні методи типу ДФП-методу

DFPR(α)-алгоритм має «ідеалізований» характер у тому значенні, що він передбачає точну процедуру найшвидшого спуску, яка нереалізовна навіть для гладких функцій. Крім того, для негладких функцій точна процедура найшвидшого спуску гарантує сильнішу умову, ніж існування антисубградієнта, ортогонального до напрямку спуску. Зокрема, кут може бути тупим, тобто $(g_k, g_{k+1}) < 0$. Тому перенесення основних принципів DFPR(α)-алгоритму на загальний випадок мінімізації опуклих функцій вимагає вирішення двох головних питань. Перше – заміна перетворення (2.29) перетворенням, яке б забезпечувало вибір напрямку руху, як в DFPR(α)-алгоритмі, не лише при ортогональних послідовних субградієнтах. Друге – заміна точного найшвидшого спуску в напрямку субградієнта іншим регулюванням крокового множника, яке б потребувало невеликої кількості обчислень $f(x)$ та $\partial f(x)$.

Перше питання не є проблемою, оскільки незалежно від кута між g_k та g_{k+1} в $Y_k = B_k^{-1}X$, вибір напрямку руху як в DFPR(α)-алгоритмі можна зберегти, якщо для корекції матриці $B_{k+1} = B_k T_k(g_k, g_{k+1})$ обрати

$$T_k(g_k, g_{k+1}) = I - \left(\frac{g_{k+1} - g_k}{\|g_{k+1} - g_k\|} + t_k \left(g_k - \frac{(g_{k+1}, g_k)}{\|g_{k+1}\|^2} g_{k+1} \right) \right) \left(\frac{g_{k+1} - g_k}{\|g_{k+1} - g_k\|} \right)^T . \quad (2.31)$$

Тут t_k – певний скалярний параметр, від вибору якого залежить збіжність методів.

Друге питання пов'язане з регулюванням крокового множника в напрямку субградієнту та є більш складним і призводить до різного роду модифікацій. Так, класичний феєрівський крок в напрямку субградієнта або певний його аналог, якщо f^* відоме, призводить до немонотонним по функціоналу субградієнтним методам «змінної метрики» на основі перетворення (2.31). При цьому очевидно, що перетворення простору потрібне на тих кроках методу, коли для найкоротшої опуклої комбінації векторів g_k та g_{k+1} виконуються $\lambda_1 > 0$ та $\lambda_2 > 0$, як в (2.27) – (2.28).

Перетворення простору (2.31) сильніше розширює конус можливих напрямків спадання функції, ніж «ортогоналізація» в подібних методах (Стецюк 2014, с. 239–244). Обґрунтування збіжності таких методів можна побудувати аналогічно до того, як це виконано в (Стецюк 2014), прибравши навіть вимогу на обмеженість евклідової норми матриці B_k. Такий спосіб регулювання крокового множника потребує не більше одного обчислення $f(x)$ та $\partial f(x)$ на кожному кроці методу і дозволяє за скінчену кількість кроків K або знайти $f(x_K)$ зі значенням функції, яке потребується, або отримати достатні умови розбіжності феєрівського процесу, коли замість f^* використовується занижене значення.

Адаптивний спосіб регулювання кроку (Шор и Стеценко 1989), який використовується в $r(\alpha)$-алгоритмі, дозволяє отримати майже монотонні по

функціоналу методи. Формально він призводить до заміни точного пошуку мінімуму за напрямком на наближений, тобто $h_k > h_k^*$, але так, щоб h_k було близьким до h_k^*. Тут h_k^* – точний крок найшвидшого спуску в напрямку антисубградієнта. Враховуючи, що гарантоване зменшення норми субградієнта в перетвореному просторі дозволяє просто уточнити крок найшвидшого спуску в перетвореному просторі, за такого регулювання крокового множника достатньо на кожному кроці методу використовувати в середньому 2-3 обчислення $f(x)$ та $\partial f(x)$. При адаптивному регулюванні крокового множника можна переходити і в точки з меншим кроком, ніж h_k^*. Таким чином, одночасне виконання умов $\|g_k\|^2 > (g_k, g_{k+1})$ та $\|g_{k+1}\|^2 > (g_k, g_{k+1})$ дозволяє застосовувати перетворення (2.31) і коли $(g_k, g_{k+1}) > 0$.

Отже, для мінімізації опуклих функцій як перший, так і другий способи регулювання крокового множника дозволяють отримати практично реалізовні субградієнтні методи «змінної метрики» на основі перетворення типу ДФП-методу в умовах мінімального використання інформації (тільки два послідовних субградієнти).

Відмітимо, що обговорені вище субградієнтні методи «змінної метрики» більше відповідають назві роботи (Lemaréchal 1975), ніж методи, які запропоновано в цій роботі, які стали основою для створення ε-субградієнтних методів в негладкій оптимізації. Насправді, в ε-субградієнтних методах з ДФП-методу був запозичений лише принцип руху за напрямком, протилежним до найкоротшого вектора до опуклої оболонки двох послідовних субградієнтів, і узагальнений на випадок більшої кількості векторів. Але зміст ДФП-методу полягає не скільки у виборі такого напрямку руху, скільки в лінійному перетворенні простору, який застосовується, і яке дозволяє покращувати структуру поверхонь рівня функції, що мінімізується.

2.6 Висновки до другого розділу

1. Розглянуто H- та B-форми квазіньютонівських методів та методу Давидона – Флетчера – Пауелла (ДФП-методу). Проведено порівняння цих методів з r-алгоритмами, виділено їх переваги та недоліки.
2. Для мінімізації гладких опуклих функцій побудовано DFPR(α)-алгоритм – градієнтний метод з перетворенням простору, який поєднує властивості як квазіньютонівських методів, так і r-алгоритмів. Проведено порівняння цього методу з r-алгоритмами на прикладі мінімізації квадратичних функцій, в тому числі й сильно яружних. Обговорюються можливі схеми такого типу методів для мінімізації негладких опуклих функцій.

РОЗДІЛ 3. ЕКОНОМНИЙ ВАРІАНТ R-АЛГОРИТМУ ТА L-BFGS-B

У третьому розділі розглядається $r(\alpha)$-алгоритм з прискореною реалізацією розтягу простору. Наведено результати обчислювальних експериментів та проведено порівняння з класичними варіантами *r*-алгоритмів. У підрозділі 3.3 розглядається задача квантильної регресії та застосування $r(\alpha)$-алгоритму з адаптивним регулюванням кроку для її розв'язання. В підрозділі 3.4 розглядається використання методу BFSG та його варіантів для задачі побудови S-подібної кривої.

3.1 $r(\alpha)$-алгоритм з прискореною реалізацією розтягу простору

r-Алгоритми – це сімейство субградієнтних методів з перетворенням простору змінних, в якому використовується спуск по антисубградієнту, а перетворення простору на кожній ітерації реалізується за допомогою оператора розтягу простору в напрямку різниці двох послідовних субградієнтів. Загальну схему r -алгоритмів можна описати наступним чином.

Нехай $f(x)$ – опукла функція, $x \in R^n$, x_0 – стартова точка, B_0 – деяка задана $n \times n$-матриця, $\alpha_k \ (\alpha_k > 1)$ – набір коефіцієнтів розтягу простору, $k = 0,1,2,\ldots$. Тоді *r*-алгоритм – це ітеративний процес пошуку мінімуму функції $f(x)$, що генерує наступну послідовність:

$$x_{k+1} = x_k - h_k B_k \xi_k, \ B_{k+1} = B_k R_{\beta_k}(\eta_k), \ k = 0,1,2,\ldots, \tag{3.1}$$

де

$$\xi_k := \frac{B_k^T g(x_k)}{\left\| B_k^T g(x_k) \right\|}, \ h_k \approx h_k^* := \arg\min_{h \geq 0} f(x_k - h B_k \xi_k), \tag{3.2}$$

$$R_\beta(\eta_k) = I + (\beta - 1)\eta_k \eta_k^T,\ \eta_k := \frac{B_k^T r_k}{\left\| B_k^T r_k \right\|},\ r_k := g_{k+1} - g_k. \qquad (3.3)$$

Тут h_k – величина кроку з точки x_k до точки (наближеного) мінімуму у напрямку $-h_k B_k \xi_k$, $R_\beta(\eta) = I + (\beta - 1)\eta\eta^T$ – оператор стиснення простору у напрямку нормованого вектора η з коефіцієнтом стиснення $\beta = \frac{1}{\alpha} < 1$, g_{k+1} та g_k – субградієнти функції f в точках x_{k+1} та x_k відповідно.

Залежно від вибору коефіцієнту розтягу простору α_k, процедури пошуку (наближеного) мінімуму функції за напрямком в (3.2) та умов зупинки алгоритма можна отримати різні варіанти *r*-алгоритма.

Одним із поширених варіантів *r*-алгоритмів є $r(\alpha)$-алгоритм з адаптивним спуском за напрямком. В ньому, як видно з назви, для розтягу простору використовується постійне значення коефіцієнту розтягу простору $\alpha_k = \alpha$.

В якості умов зупинки використовуються стандартні критерії:

- $\left\| x_{k+1} - x_k \right\| < epsx\ (epsx > 0)$ – умова зупинки по аргументу,
- $\left\| g_{k+1} \right\| < epsg\ (epsg > 0)$ – умова зупинки по нормі градієнта (для гладких функцій),
- $k > maxitn\ (maxitn\ N)$ – умова зупинки за максимальної кількості ітерацій.

Також використовується умова зупинки по максимальній кількості ітерацій в процедурі пошуку наближеного мінімуму за напрямком. За замовчуванням, цей параметр дорівнює 500.

В якості процедури пошуку наближеного мінімуму за напрямком в $r(\alpha)$-алгоритмі використовується спеціальна процедура адаптивного спуску в напрямку антисубградієнта в перетвореному просторі змінних. Адаптивний спуск за напрямком – це ітеративна процедура одновимірного спуску за напрямком, яка завершується, як тільки виконується задана умова. Умовою

завершення процедури спуску для $r(\alpha)$-алгоритма є умова, що субградієнт функції у знайденій точці має утворювати із вектором спуску гострий кут. На кожному кроці ітерації процедури відбувається рух вздовж заданого напрямку на величину hs, яка налаштовується адаптивно за допомогою чотирьох параметрів: h_0 – початкова величина крокового множника (використовується на першій ітерації *r*-алгоритму, а на кожній наступній – уточнюється), $q_1\ (q_1 \le 1)$ – коефіцієнт зменшення крокового множника (якщо умова завершення спуску за напрямком виконується за перший крок), $q_2\ (q_2 \le 1)$ – коефіцієнт збільшення крокового множника. Через кожні $nh\ (nh > 1)$ кроків одновимірного спуску величина крокового множника hs збільшується в q_2 раз.

Алгоритм адаптивного спуску за напрямком можна описати так.

Вхідні дані:

p – напрямок спуску;

y_0 – початкова точка.

Параметри алгоритму:

h_0 – початкова величина крокового множника;

q_1 – коефіцієнт зменшення крокового множника (якщо умова завершення спуску за напрямком виконується за перший крок);

q_2 – коефіцієнт збільшення крокового множника;

nh – кількість кроків після яких збільшується кроковий множник.

Вихідні дані:

y_{ls} – точка, в якій виконується умова зупинки процедури спуску.

Початок алгоритму:

$hs = h_0$ – ініціалізація початкового значення *hs*;

$ls = 0$ – ініціалізація початкового значення лічильника циклу.

Початок циклу:

$ls = ls + 1$ – номер ітерації;

$y_{ls} = y_{ls-1} + hs * p$ – перерахунок нової точки.

Якщо y_{ls} задовольняє умову зупинки – зупинка та вихід із циклу.

Якщо ls кратне nh, тоді $hs = hs * q_2$ – оновити значення hs.

Кінець циклу.

Якщо $ls = 1$, тоді $h_0 = hs * q_1$ – оновити початкове значення кроку hs для наступного застосування процедури.

Кінець алгоритму.

Опис прискореного варіанту $r(\alpha)$-алгоритму. На кожній ітерації $r(\alpha)$-алгоритму матриця B_k перераховується за такою формулою:

$$B_{k+1} = B_k R_\beta(\eta_k) = B_k\left(I + (\beta - 1)\eta_k \eta_k^T\right), \tag{3.4}$$

де $\eta_k = \dfrac{\eta}{\|\eta\|}$, $\eta = B_k^T r_k$ – вектор різниці двох послідовних субградієнтів у перетвореному просторі, а матриця $R_\beta(\eta_k)$ задає оператор розтягу простору в напрямку η_k.

В (Стецюк та Жмуд 2020) було відмічено, що кількість операцій множення в (3.4) можна зменшити, якщо у векторі η залишити лише $m\,(m < n)$ найбільших за модулем елементів. Наприклад, вектор η можна модифікувати за допомогою спеціального параметра контролю t таким чином:

$$\eta = \left\{\eta^i\right\}_{i=1}^n, \text{ де } \eta^i = \begin{cases} 0, & |\eta^i| \le t \max\limits_{j=1,n} |\eta^j| \\ \eta^i, & |\eta^i| \ge t \max\limits_{j=1,n} |\eta^j| \end{cases}, \quad 0 \le t < 1, \quad \eta = \left\{\eta^i\right\}_{i=1}^n. \tag{3.5}$$

Отже, якщо в (3.4) використовувати модифікований вектор розтягу простору $\eta_k = \dfrac{\eta}{\|\eta\|}$, де η обчислюється за формулою (3.5), то має місце таке твердження.

Лема 3.1. Якщо в (3.4) використовувати модифікований напрямок розтягу простору η_k, то кількість операцій множення, необхідних для перерахунку B_{k+1}, зменшується з $2n^2 + 3n$ до $2nm + m$, де m – кількість ненульових елементів у η_k.

Доведення. Запишемо (3.4) таким чином: $B_{k+1} = B_k\left(I + (\beta - 1)\eta_k\eta_k^T\right) =$ $= B_k + (\beta - 1)(B_k\eta_k)\eta_k^T$. Тоді легко бачити, що для перерахунку B_{k+1} в (3.4) виконується $2n^2 + n$ операцій множення: n^2 операцій для обчислення вектора $\zeta = B_k\eta_k$; n операцій для обчислення вектора $\xi = (\beta - 1)\zeta$, та n^2 операцій для обчислення матриці $\xi\eta_k^T$. Аналогічно, для η_k маємо $B_{k+1} = B_k + (\beta - 1)\left(B_k\eta_k\right)\eta_k^T$, звідки легко бачити, що для перерахунку B_{k+1} виконується $2nm + m$ операцій множення: mn операцій для обчислення вектора $\zeta = B_k\eta_k$, m операцій для обчислення вектора $\xi = (\beta - 1)\zeta$, та mn операцій для обчислення матриці $\xi\eta_k^T$. Лему доведено.

Зазначимо, що при $m < n$, оператор розтягу простору $R_\beta(\eta_k)$ діє не на всьому просторі змінних, як у загальній схемі r-алгоритму, а тільки на деякому m-вимірному підпросторі.

На основі вищенаведеного, опишемо схему $r(\alpha)$-алгоритму з адаптивною процедурою спуску за напрямком та прискореним варіантом розтягу простору у напрямку наближеної різниці двох послідовних субградієнтів:

Вхідні параметри:

x_0 – стартова точка,

$calcfg(x)$ – функція, шо обчислює значення та субградієнт цільової функції $f(x)$ в точці x.

Параметри для операції розтягу простору:

α – коефіцієнт розтягу простору;

t – коефіцієнт контролю напрямку розтягу простору.

Параметри процедури адапонивого спуску за напрямком:

h_0 – початкове значення крокового множника;

q_1 – коефіцієнт зменшення крокового множника;

q_2 – коефіцієнт збільшення крокового множника;

nh – період збільшення крокового множника.

Параметри критеріїв зупинки алгоритму:

$maxitn$ – максимальна кількість ітерацій основного циклу;

$epsg$ – параметр критерію зупинки за нормою градієнта (працює для гладких функцій);

$epsx$ – параметр критерію зупинки за аргументами;

Вихідні параметри:

f_r – рекордне значення функції;

x_r – рекордна точка.

Початок алгоритму:

Ініціалізуємо локальні змінні алгоритму:

$k = 0$ – лічильник ітерацій основного циклу;

$B_0 = I$ – матрицю B_0 ініціалізуємо як одиничну діагональну;

$hs = h_0$ – кроковий множника для адаптивної процедури спуску за напрямком;

$w = 1/\alpha - 1$ – множник оператора стиснення простору.

Обчислюємо значення функції f та її субградієнта g_1 в стартовій точці x_0:

$$f, g_1 = calcfg(x_0).$$

Ініціалізуємо дані про рекордну точку:

$$f_r = f,\ x_r = x.$$

Перевіряємо критерій зупинки за $epsg$:

Якщо $\|g_1\| < epsg$ – завершити алгоритм.

Початок головного циклу:

Перевіряємо критерій зупинки по кількості ітерацій у циклі:

Якщо $k > maxitn$ – завершити алгоритм.

Оновлюємо лічильник k:

$$k = k + 1.$$

Обчислюємо субградієнт g_1 у перетвореному просторі змінних:

$$g = B_k^T g_1.$$

Нормуємо вектор g:

$$g = \frac{g}{\|g\|}.$$

Обчислюємо відповідний вектор у початковому просторі змінних:

$$dx = B_k^T g\ .$$

Ініціалізуємо лічильник ls та початкову точку x_{k+1}:

$$ls = 0,\ x_{k+1} = x_k\ .$$

Початок циклу процедури спуску за напрямком $-dx$:

Перераховуємо x_{k+1} та ls:

$$x_{k+1} = x_{k+1} - hs * dx;$$

$$ls = ls + 1.$$

Обчислюємо значення функції f та її субградієнта g в точці x_{k+1}:

$$f, g_2 = calcfg\left(x_{k+1}\right).$$

Оновлюємо дані про рекордну точку:

Якщо $f < f_r$, тоді $f_r = f$, $x_r = x_{k+1}$

Перевіряємо умову зупинки за нормою градієнта (для гладких функцій):

Якщо $\|g_2\| < epsg$ – завершити алгоритм.

Перевіряємо умову зупинки за необмеженістю f:

Якщо $ls > 500$ – завершити алгоритм.

Оновлюємо кроковий множник hs:

Якщо ls кратне nh, тоді $hs = hs * q_2$.

Обчислюємо скалярний добуток напрямку спуску dx та субградієнта g_2:

$$d = g_2^T dx$$

Перевіряємо умову завершення спуску за напрямком:

Якщо $d < 0$ – завершити цикл.

Переходимо до наступної ітерації циклу.

Кінець циклу процедури спуску за напрямком $-dx$.

Оновлюємо кроковий множник hs для процедури спуску на наступній ітерації:

Якщо $ls = 1$, тоді $hs = hs * q_1$.

Перевіряємо умову зупинки за аргументом:

Якщо $\|x_{k+1} - x_k\| < epsx$, тоді завершити алгоритм.

Обчислюємо вектор різниці субградієнтів:

$$r_k = g_2 - g_1.$$

Обчислюємо r_k у перетвореному просторі змінних:

$$\eta = B_k^T r_k.$$

Модифікуємо вектор r згідно з (3.5):

$$\eta = \left\{\eta^i\right\}_{i=1}^n, \text{ де } \eta^i = \begin{cases} 0, & |\eta^i| \le t \max\limits_{j=1,n} |\eta^j| \\ \eta^i, & |\eta^i| \ge t \max\limits_{j=1,n} |\eta^j| \end{cases}, \quad \eta = \left\{\eta^i\right\}_{i=1}^n, \quad 0 \le t < 1.$$

Нормуємо вектор η:

$$\eta_k = \frac{\eta}{\|\eta\|}.$$

Обчислюємо матрицю B_{k+1}:

$$B_{k+1} = B_k R_\beta\left(\eta_k\right) = B_k\left(I + w\eta_k \eta_k^T\right)$$

Зберігаємо поточний субградієнт:

$$g_1 = g_2.$$

Переходимо до наступної ітерації циклу.

Кінець головного циклу

Кінець алгоритму

3.2 Обчислювальні експерименти

В цьому підрозділі наведемо результати обчислювальних експериментів задач мінімізації двох сімейств опуклих яружних функцій – кусково-лінійних негладких функцій $SABS(q,n)=\sum_{i=1}^{n} q^{i-1} abs(x_i-1)$ та квадратичних строго опуклих функцій $SQUAD(q,n)=\sum_{i=1}^{n} q^{2(i-1)}\left(x_i^{'}-1\right)^2$ з параметром $q=1.1$ при розмірностях $n=100$ та $n=200$ та точкою мінімуму $x_0=(1,1,...,1)$. Для їхньої мінімізації використовувались C++ реалізація $r(\alpha)$-алгоритму з адаптивною процедурою спуску за напрямком ralgb5a та реалізація модифікованого $r(\alpha)$-алгоритму з адаптивною процедурою спуску за напрямком та прискореним розтягом простору ralgb5at. Абревіатура "b5" означає, що коректується $n\times n$-матриця B_k, а обчислювальна складність кожної ітерації k визначається $5n^2$ арифметичними операціями множення (з них, як зазначалось, $2n^2$ припадає на перерахунок матриці B_k в (3.4)). “a” означає, що пошук наближеного мінімуму за напрямком реалізується адаптивною процедурою пошуку за напрямком. “t” означає, що реалізується операція прискореного розтяг простору, напрям якого контролюється параметром t за формулою (3.5). Обчислення проводились на комп’ютері з процесором Intel(R) Core(TM) i7-8565U CPU 1.80GHz 1.99GHz та оперативною пам’яттю 8 Гб з використанням компілятора g++ (GCC) 10.1.0. При компіляції вихідного коду використовувалась команда оптимізації -03.

Параметри алгоритмів вибирались згідно з рекомендаціями в (Стецюк 2017) – $\alpha=2$, $h_0=10$ для $n=100$ та $h_0=15$ для $n=200$, $q_1=1$ для негладких функцій $SABS$ та $q_1=0.85$ для гладких функцій $SQUAD$, $q_2=1.1$, $nh=3$, $epsx=10^{-6}$, $epsg=10^{-12}$, $maxitn=15000$. Стартова точка вибиралась $x_0=(0,0,...,0)$.

В таблицях 3.1 – 3.4 наведено результати тестів мінімізації чотирьох вищенаведених функцій для шести варіантів $r(\alpha)$-алгоритму при різних значеннях параметру контроля напрямку розтягу простору $t = \{0, 0.5, 0.2, 0.1, 0.02, 0.01\}$. При $t = 0$ використовується класичний ralgb5a, при інших значеннях t – ralgb5at. Результати описуються наступними показниками: t – значення параметру контроля напрямку розтягу простору, *itn* – кількість виконаних алгоритмом ітерацій, *ncalls* – кількість викликів функції *calcfg*, що обчислює значення мінімізуючої фунції та її субградієнта в точці, *nzeros* – сумарна кількість нульових компонент вектора розтягу простору за всі ітерації, *totalcomp* – сумарна кількість операцій множення з ненульовими елементами за всі ітерації для перерахунку B_k в (3.4), *economcomp* – відсоткове відношення значення *totalcomp* для даного алгоритма відносно відповідного значення для ralgb5a, *time* – час роботи алгоритма в секундах.

Таблиця 3.1

Розрахунки для $SABS(1.1, 100) = \sum_{i=1}^{100} 1.1^{i-1} abs(x_i' - 1)$

T	*itn*	*ncalls*	*nzeros*	*totalcomp*	*economcomp*	*time*
0(ralgb5a)	2778	2785	0	56 393 400	100	0.37
0.5	2826	2827	201 293	16 686 314	29.58	0.378
0.2	2772	2778	106 444	34 749 612	61.61	0.372
0.1	2774	2782	61 540	43 860 820	77.77	0.365
0.02	2784	2791	18 456	52 766 788	93.59	0.366
0.01	2750	2755	11 595	53 462 510	94.8	0.355

Таблиця 3.2

Розрахунки для $SABS(1.1, 200) = \sum_{i=1}^{200} 1.1^{i-1} abs(x_i' - 1)$

t	*itn*	*ncalls*	*nzeros*	*totalcomp*	*economcomp*	*time*
0(ralgb5a)	6 953	6967	0	560 411 800	100	1.113

0.5	7040	7042	1 083 135	132 003 930	24	1.034
0.2	6 947	6952	625 929	308 304 942	55	1.04
0.1	6 124	6923	398577	397 204 246	70.87	1.02
0.02	6932	6944	172 708	489 290 784	87.3	1.024
0.01	6 938	6929	133 835	504 677 730	90.05	1.016

Таблиця 3.3

Розрахунки для $SQUAD(1.1,100)=\sum_{i=1}^{100}1.1^{2(i-1)}\left(x_i^{'}-1\right)^2$

t	*itn*	*ncalls*	*nzeros*	*totalcomp*	*economcomp*	*time*
0(ralgb5a)	528	1 032	0	10 718 400	100	0.027
0.5	310	563	28 833	468 834	4.37	0.014
0.2	313	560	26 586	983 628	9.17	0.014
0.1	335	613	26 401	1 467 598	13.69	0.014
0.02	494	921	26 041	4 768 018	44.48	0.021
0.01	539	1 006	20 998	6 700 204	62.51	0.023

Таблиця 3.4

Розрахунки для $SQUAD(1.1,200)=\sum_{i=1}^{200}1.1^{2(i-1)}\left(x_i^{'}-1\right)^2$

t	*itn*	*ncalls*	*nzeros*	*totalcomp*	*economcomp*	*time*
0(ralgb5a)	2 286	4 792	0	184 251600	100	0.384
0.5	695	1326	133 730	2 257 740	1.222 324	0.115
0.2	722	1386	131 864	5 184 072	2,806619	0.119
0.1	950	1 925	161 331	11715 138	6,34249	0.156
0.02	1 644	3 407	205 935	49 720 730	26,91844	0.259
0.01	2 233	4 615	186 602	104 965 996	56,82783	0.354

Зазначимо, що всі ітерації алгоритмів зійшлись до точки мінімуму функцій і завершилися за умовою зупинки по аргументу. З таблиць видно, що для алгоритмів з модифікованим варіантом розтягу простору ralgb5at (рядки при $t > 0$) сумарна кількість операцій множення (стовпчик *totalcomp*) значно менша у порівнянні з класичним ralgb5a (рядок при $t = 0$). Слід зазначити, що для негладких функцій *SABS* загальна кількість ітерацій (стовпчик *itn*) і кількість обчислень значень функції та субградієнта (стовпчик *ncalls*) приблизно однакова, а для квадратичних функцій *SQUAD* загальна кількість ітерацій, кількість обчислень значень функції та градієнта, і, як наслідок, час виконання (стовпчик *time*) варіантів ralgb5at значно менший у порівнянні з класичним ralgb5a. Це може свідчити про те, що для таких класів функцій прискорена реалізація розтягу простору краще враховує характер яружності функцій.

3.3 *r* -Алгоритм для задачі оцінки квантильної регресії

Квантильна регресія – це статистична модель, яка для заданого значення змінної-регресора оцінює квантиль змінної-відгуку:

$$Q_{Y|X}(\tau) = \inf\left\{y : F_{Y|X}(y) \geq \tau\right\}. \tag{3.6}$$

Зазвичай, для (3.6) розглядається лінійна модель:

$$Q_{\tau}(y) = \sum_{i=1}^{p} \beta_i(\tau)\, x_i, \tag{3.7}$$

де $x = (x_1, ..., x_d)$ – вектор змінної-регресора, d – розмірність простору змінної-регресора, y – змінна-відгук. На практиці, задачу оцінки параметрів регресійної моделі за заданою вибіркою На практиці, задачу оцінки параметрів регресійної моделі за заданою вибіркою $\{x_i, y_i\}_{i=1}^{n}$, $x = (x_{i1}, ..., x_{id})$, n – розмір вибірки, зводять до задачі мінімізації відповідної штрафної функції. Так, в (Koenker and Bassett 1978), для моделі (3.7) пропонується використовувати таку функцію штрафу:

$$L_\tau(\beta) = \sum_{\{i:\, y_i \geq x_i'\beta\}} \tau \left| y_i - x_i'\beta \right| + \sum_{\{i:\, y_i < x_i'\beta\}} (1-\tau) \left| y_i - x_i'\beta \right|. \quad (3.8)$$

Стандартним способом мінімізації (3.8) є перехід до відповідної задачі лінійного програмування. Тоді пряма задача лінійного програмування для мінімізації (3.8) формулюється так: знайти

$$\tau \sum_{i=1}^{n} u_i + (1-\tau) \sum_{i=1}^{n} v_i \to \min \quad (3.9)$$

за обмежень

$$\sum_{j=1}^{d} x_{ij}\beta_j + u_i - v_i = y_i\,,\ 1 \leq i \leq n,\ u_i \geq 0,\ v_i \geq 0. \quad (3.10)$$

Двоїста до (3.9) – (3.10) задача формулюється так: знайти

$$\tau \sum_{i=1}^{n} y_i a_i \to \max \quad (3.11)$$

за обмежень

$$\sum_{i=1}^{n} x_{ij} a_i = (1-\tau) \sum_{j=1}^{n} x_{ij}\,,\ 1 \leq j \leq d\,,\ 0 \leq a_i \leq 1,\ 1 \leq i \leq n. \quad (3.12)$$

Для розв'язку задач (3.9) – (3.10) та (3.11) – (3.12) зазвичай застосовують симплекс метод та методи внутрішніх точок (Koenker 2017).

Але, водночас, можна помітити, що штрафну функцію в (3.8) можна записати в еквівалентній їй формі таким чином (Suprun 2020):

$$\sum_{i=1}^{n} \max\left(\tau\left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j \right), -(1-\tau)\left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j \right) \right). \quad (3.13)$$

Отже, маємо задачу безумовної мінімізації опуклої негладкої функції. Для мінімізації (3.13) можуть бути використані алгоритми негладкої оптимізації, наприклад, субградієнтні алгоритми, або субградієнтні алгоритми з розтягом простору (Shor 1985). Відзначимо, що при $\tau = 0.5$ задача мінімізації функції (3.13) співпадає з методом найменших модулів.

На основі цього були проведені чисельні експерименти для моделей (3.9) – (3.10), (3.11) – (3.12) та (3.13). Обчислення проводились на комп'ютері з процесором Intel(R) Core(TM) i7-8565U CPU 1.80GHz 1.99GHz та оперативною пам'яттю 8 Гб в середовищі GNU Octave. Дані для вибірки використовувалися з пакета Quantreg для мови R, або генерувалися випадковим чином з рівномірним розподілом похибок. Для мінімізації (3.13) використовувалась програмна реалізація $r(\alpha)$-алгоритму з адаптивним регулюванням кроку ralgb5a, написана на мові програмування Octave (Стецюк, Бєлих, та Криворучко 2019). Для моделей (3.9) – (3.10) та (3.11) – (3.12) застосовувався варіант алгоритму внутрішніх точок, реалізований Р. Конкером мовою програмування MATLAB. Результати чисельних експериментів показують, що алгоритми у всіх випадках збігаються до околу одних і тих самих точок. Також, можна зазначити, що для наборів даних із великим розміром вибірки n та відносно невеликою розмірністю параметрів d, варіант r-алгоритму може швидше збігатися до розв'язку, ніж метод внутрішніх точок. Так, наприклад, у таблиці 5 наведено час обчислень (в секундах) для набору даних з 100 000 вибірками та 100 параметрами з параметром зупинки для обох алгоритмів $\varepsilon = 10^{-5}$.

Таблиця 3.5

Порівняння часу роботи r-алгоритму та IPDA

τ	0.1	0.2	0.4	0.5	0.8	0.9
r-алгоритм	61.9203	20.9172	21.3754	15.98	19.4265	87.4515
IPDA	70.9699	87.8515	71.7939	61.9695	90.3999	85.1519

Можна підкреслити, що розглянута в (3.13) модель є достатньо простим і лаконічним варіантом стандартної моделі квантильної регресії. Для її мінімізації можна застосовувати та розробляти нові варіанти субградієнтних алгоритмів з розтягом простору. Також, варто додати, що в науковій літературі можна знайти не так багато джерел, які стосуються застосувань негладких методів оптимізації для квантильної регресії, або інших подібних робастних

моделей. Тому розвиток нових моделей та методів негладкої оптимізації для статистичних моделей типу (3.6) – (3.7) є цілком доцільним і актуальним завданням.

3.4 Метод BFGS для задачі побудови S-подібної кривої

Метод **BFGS** (Broyden, Fletcher, Goldfarb, Shanno) відомий як один з ефективних методів квазиньютонівського типу (Пшеничный и Данилин 1975), (Gill, Murray, and Wright 1981), в яких на відміну від ньютонівських методів гессіан функції обчислюється наближено за допомогою градієнтів в точках ітераційного процесу. Для строго опуклої квадратичної функції від n змінних методи квазіньютонівського типу гарантують збіжність до точки мінімуму не більше ніж за n ітерацій, де кожна ітерація вимагає малорангових корекцій симетричної $n \times n$-матриці H та реалізується за O(n^2) арифметичних операцій. На останній ітерації методу матриця H з точністю до постійного множника співпадає з матрицею, оберненою до гессіана квадратичної функції.

Програмні реалізації методу BFGS та його проективного варіанту L-BFGS-B (Byrd et al. 1995) активно використовуються для мінімізації гладких функцій. Вони є ефективними, якщо стартова точка вибирається в тому околі точки мінімуму, де функція, що мінімізується, достатньо точно апроксимується опуклою квадратичною функцією.

В підрозділі розглянемо застосування методів BFGS та L-BFGS-B для мінімізації нелінійної гладкої функції, яка відповідає знаходженню розвязків системи п’яти нелінійних рівнянь (Стецюк, Ткаченко, та Грицай 2020), де три рівняння є інтегральними (залежать від невідомих параметрів підінтегральних функцій та невідомих верхніх границь для визначеного інтегралу). Ця система відповідає задачі побудови S-подібної кривої у натуральній параметризації (Борисенко, Устенко, и Устенко 2018), яка проходить через дві задані точки із

заданими кутами нахилу дотичних у них та забезпечує заданий кут нахилу дотичної в проміжній точці із заданою абсцисою.

Система нелінійних рівнянь та її властивості. Задача побудови плоскої кривої (в натуральній параметризації з квадратичним графіком кривини), яка проходить через дві точки (x_1, y_1) та (x_2, y_2) із заданими φ_1, φ_2 – кутами нахилу дотичних у них та забезпечує φ_p – заданий кут нахилу дотичної в точці (x_p, y_p) із заданою абсцисою x_p описується за допомогою системи із п'яти нелінійних рівнянь (Стецюк, Ткаченко, та Грицай 2020):

$$x_2 = x_1 + \int_0^S \cos\left(\varphi_1 + \frac{as^3}{3} + \frac{bs^2}{2} + cs\right) ds, \tag{3.14}$$

$$y_2 = y_1 + \int_0^S \sin\left(\varphi_1 + \frac{as^3}{3} + \frac{bs^2}{2} + cs\right) ds, \tag{3.15}$$

$$\varphi_2 = \varphi_1 + \frac{aS^3}{3} + \frac{bS^2}{2} + cS, \tag{3.16}$$

$$x_p = x_1 + \int_0^{s_p} \cos\left(\varphi_1 + \frac{as^3}{3} + \frac{bs^2}{2} + cs\right) ds, \tag{3.17}$$

$$\varphi_p = \varphi_1 + \frac{as_p^3}{3} + \frac{bs_p^2}{2} + cs_p. \tag{3.18}$$

Система (3.14) – (3.18) має п'ять невідомих: a, b, c – три коефіцієнти квадратичної функції $as^2 + bs + c$, S – довжина кривої від точки (x_1, y_1) до точки (x_2, y_2), s_p – довжина ділянки кривої від точки (x_1, y_1) до точки (x_p, y_p). Вона включає п'ять нелінійних рівнянь, серед яких рівняння (1), (2), (4) є інтегральними та залежать від невідомих параметрів підінтегральних функцій та невідомих верхніх границь для визначеного інтегралу.

Рівняння (3.14) – (3.18) визначаються формулами для кривої $x(s)$, $y(s)$, $0 \le s \le S$ в натуральній параметризації (Мищенко и Фоменко 2004), де кут нахилу дотичної $\varphi(s)$ в точці $x(s)$, $y(s)$ визначається за формулою:

$$\varphi(s) = \varphi(0) + \int_0^s k(s)ds = \varphi(0) + \frac{as^3}{3} + \frac{bs^2}{2} + cs, \qquad (3.19)$$

а координати точок – за формулами:

$$x(s) = x(0) + \int_0^s \cos\varphi(s)ds, \quad y(s) = y(0) + \int_0^s \sin\varphi(s)ds \qquad (3.20)$$

де $k(s) = as^2 + bs + c$ – задана квадратична функція кривини. Так, інтегральні рівняння (3.14) та (3.15) зв’язують між собою точки (x_1, y_1) та (x_2, y_2) за формулами (3.20). Рівняння (3.16) за формулою (3.19) забезпечує потрібний кут φ_2 в точці (x_2, y_2), який визначається за заданим кутом φ_1 в точці (x_1, y_1). Рівняння (3.17) та (3.18) забезпечують кут рівний φ_p в точці з абсцисою x_p.

У загальному випадку система (3.14) – (3.18) має багато розв'язків. Так, наприклад, у таблиці 3.1 наведено два розв’язки для наступних вихідних даних:

$$x_1 = 0,\ y_1 = 2.46,\ \varphi_1 = 0;\ \ x_2 = 1,\ y_2 = 2.75,\ \varphi_2 = 0;\ x_p = 0.7, \varphi_p = 0.209440 = 12^\circ. \qquad (3.21)$$

Графіки кривих для цих розв'язків наведено на рисунку 3.1. Крива для першого розв’язку знаходиться ліворуч, крива для другого розв’язку – праворуч.

Таблиця 3.6

Два розв'язки системи (3.14) – (3.18) для вихідних даних (8)

	1	2
a^*	7.92822	-7.03716
b^*	-11.4065	16.4890
c^*	3.07557	-6.19900
S^*	1.05562	2.42487

	1	2
s_p^*	0.753992	1.15072

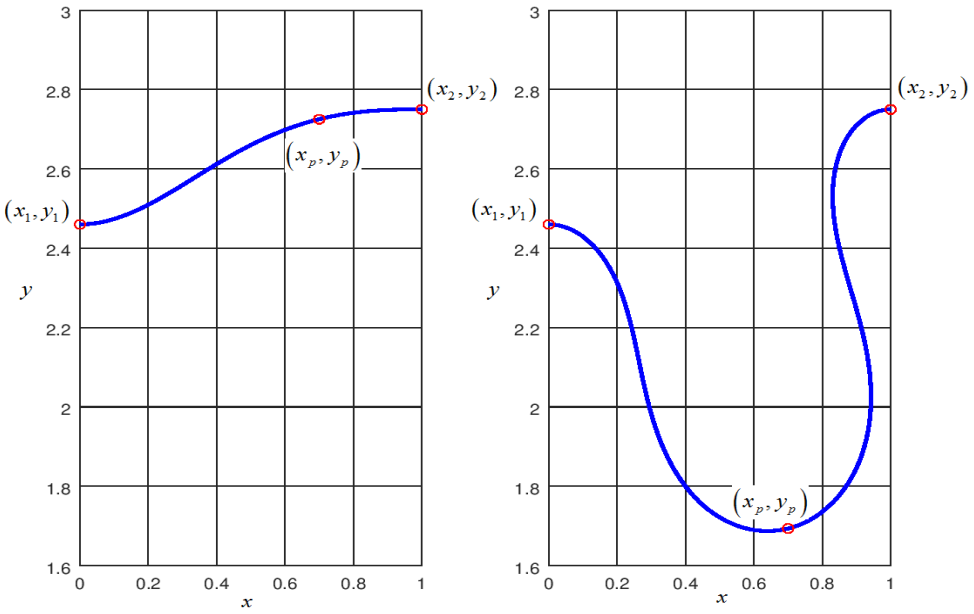


Рис. 3.1. Графіки кривих, які відповідають розв'язкам із таблиці 3.1.

З рисунку 3.1 видно, що для першого розв’язку крива є S-подібною. Для другого розв’язку це не так. Він включає дві ділянки, кожна із яких є S-подібною кривою, та характеризується довжиною $S^* = 2.42487$, яка є значно більшою за довжину $S^* = 1.05562$ для першого розв'язку. Відмітимо, що для другого розв'язку існує ділянка кривої $y(x)$, де ординати визначені неоднозначно.

Відсікти розв'язки, яким не відповідають S-подібні криві, можна за допомогою обмеження зверху на довжину кривої. Так, наприклад, якщо до системи рівнянь (3.14) – (3.18) додати нерівність $S \le 2$, то це буде відсікати другий розв'язок з таблиці 3.1.

Оптимізаційна задача для модифікації *r*-алгоритму. Розглянемо оптимізаційну задачу (Стецюк, Ткаченко, та Грицай 2020), яка має вигляд: знайти

$$f^* = f\left(a^*, b^*, c^*, S^*, s_p^*\right) =$$

$$= \min_{a,b,c,S,s_p} \left\{ f\left(a,b,c,S,s_p\right) = \sum_{i=1}^{3} \left| f_i\left(a,b,c,S\right)\right| + \sum_{i=4}^{5} \left| f_i\left(a,b,c,s_p\right)\right| \right\} \qquad (3.21)$$

при обмеженнях

$$S_{lo} \le S \le S_{up}, \qquad (3.22)$$

$$\left| x_p - x_1 \right| \le s_p \le S, \qquad (3.23)$$

де

$$f_1(a,b,c,S) = x_2 - x_1 - \int_0^S \cos\left(\varphi_1 + \frac{as^3}{3} + \frac{bs^2}{2} + cs \right) ds,$$

$$f_2(a,b,c,S) = y_2 - y_1 - \int_0^S \sin\left(\varphi_1 + \frac{as^3}{3} + \frac{bs^2}{2} + cs \right) ds,$$

$$f_3(a,b,c,S) = \varphi_2 - \varphi_1 - \frac{aS^3}{3} - \frac{bS^2}{2} - cS,$$

$$f_4(a,b,c,s_p) = x_p - x_1 - \int_0^{s_p} \cos\left(\varphi_1 + \frac{as^3}{3} + \frac{bs^2}{2} + cs \right) ds,$$

$$f_5(a,b,c,s_p) = \varphi_p - \varphi_1 - \frac{as_p^3}{3} - \frac{bs_p^2}{2} - cs_p.$$

Тут цільова функція (3.21) є негладкою та означає мінімізацію суми модулів нев'язок для рівнянь (3.14) – (3.18). Обмеження (3.22) та (3.23) гарантують додатні значення із допустимих діапазонів для довжин S та s_p, які є верхніми границями для визначених інтегралів в функціях f_1, f_2, f_4. Тут $S_{lo} > 0$, S_{up} – параметри для управління нижньою та верхнею межами на S – загальну довжину кривої. При цьому нижня межа S_{lo} не може бути меншою за $S_{\min} = \sqrt{\left(x_2 - x_1\right)^2 + \left(y_2 - y_1\right)^2}$, де $S_{\min}$ – мінімальна відстань між точками (x_1, y_1)

та (x_2, y_2). Чим ближчим до $S_{\min}$ є значення верхньої межі S_{up}, тим легше для системи (3.14) – (3.18) забезпечити розв'язок, який визначає S-подібну криву (перший розв'язок на рисунку 3.1).

Якщо в результаті пошуку локального мінімуму для задачі (3.21) – (3.22) отримуємо $f^* = 0$, то це означає, що знайдена точка глобального мінімуму $a^*, b^*, c^*, S^*, s_p^*$, яка є розв'язком системи (3.14) – (3.18). Якщо отримуємо $f^* > 0$, то знайдена точка $a^*, b^*, c^*, S^*, s_p^*$ не є розв'язком системи (3.14) – (3.18). Це може бути як у випадку відсутності розв'язку у системи (3.14) – (3.18) при обмеженнях (3.22), (3.23), так і у випадку, якщо алгоритм зупиниться в «неоптимальній» точці, враховуючи, що для великих значень параметра S_{up} задача (3.21) – (3.23) є багатоекстремальною.

Задача (3.21) – (3.23) є задачею мінімізації негладкої функції, яка визначена не при всіх значеннях S та s_p, а тільки при тих, які є додатними та дозволяють обчислювати визначені інтеграли для функцій $f_1(a,b,c,S)$, $f_2(a,b,c,S)$ та $f_4(a,b,c,s_p)$. Для знаходження точки глобального мінімуму у задачі (3.21) – (3.23) може бути використана модифікація *r*-алгоритму (Shor and Stetsyuk 1997), яка враховує вказану особливість задачі. У точці, де узагальнений градієнт цільової функції є невизначеним, модифікація *r*-алгоритму використовує узагальнений градієнт до максимально порушеного із обмежень (3.22) та (3.23).

Алгоритм розв'язання задачі (3.21) – (3.22) реалізований за допомогою методу мультистарту та octave-функції **ralgb5a** (Stetsyuk 2017b). Він використовує аналітичний спосіб обчислення узагальнених градієнтів цільової функції (3.21) та метод трапецій для обчислення визначених інтегралів. За його допомогою можна знаходити як розв'язки, яким відповідають S-подібні криві, так і інші розв'язки.

Алгоритм виявився стійким при пошуку S-подібного контуру (перший розв'язок у таблиці 1). Це пояснюється тим, що вихідні дані (3.20) є добре

масштабованими і відношення максимальної по модулю компоненти розв'язку до мінімальної є невеликим та рівним 11.4065/0.753992. Тому при параметрі $S_{up} = 1.2 * S_{\min}$ з усіх п'ятдесяти стартових точок, які вибирались випадковими або із інтервалу [0,1] або із інтервалу [0,10], алгоритм збігається до одного і того ж розв'язку системи (3.14) – (3.18). Однак, при виборі стартових точок з інтервалу [0,50] алгоритм в дев'ятнадцяти випадках знайшов розв'язок для S-подібного контуру, а в 31 випадку не зміг знайти такого розв'язку та зупинився в точках, де значення цільової функції (3.21) є більшим нуля.

Оптимізаційні задачі для для методів BFGS та L-BFGS-B. Для використання методів BFGS та L-BFGS-B необхідно оптимізаційну задачу сформулювати як задачу мінімізації гладкої функції при двосторонніх обмеженнях на змінні (для методу L-BFGS-B). Для цього будемо використовувати два формулювання оптимізаційних задач, які є еквівалентними задачі знаходження розв'язків системи (3.14) – (3.18).

Перша оптимізаційна задача має вигляд: знайти

$$F_1^* = F_1\left(a^*, b^*, c^*, S^*, \lambda^*\right) =$$

$$= \min_{a,b,c,S,\lambda} \left\{ F_1(a,b,c,S,\lambda) = \sum_{i=1}^{3} f_i^2(a,b,c,S) + \sum_{i=4}^{5} f_i^2(a,b,c,\lambda S) \right\} \qquad (3.24)$$

при обмеженнях

$$x_p / S_{\max} \le \lambda \le 1,\ S_{\min} \le S \le S_{\max}, \qquad (3.25)$$

де $S_{\min}$ та $S_{\max}$ − нижня та верхня межі на невідому довжину S. Тут цільова функція (3.24) є гладкою та означає мінімізацію суми квадратів нев'язок до рівнянь (3.14) − (3.18). Для зручності задання двосторонніх обмежень змінна s_p замінена на змінну λ, яка визначається із співвідношення $s_p = \lambda S$.

Друга оптимізаційна задача має вигляд: знайти

$$F_2^* = F_2\left(a^*, b^*, c^*, S^*, s_p^*\right) =$$

$$= \min_{a,b,c,S,s_p} \left\{ F_2\left(a,b,c,S,s_p\right) = \sum_{i=1}^{3} f_i^2\left(a,b,c,S\right) + \sum_{i=4}^{5} f_i^2\left(a,b,c,s_p\right) \right\} \qquad (3.26)$$

при обмеженнях

$$a_0 - \varepsilon_a \le a \le a_0 + \varepsilon_a,\ b_0 - \varepsilon_b \le b \le b_0 + \varepsilon_b,\ c_0 - \varepsilon_c \le c \le c_0 + \varepsilon_c, \qquad (3.27)$$

$$(s_p)_0 - \varepsilon_s \le s_p \le (s_p)_0 + \varepsilon_s,\ S_0 - \varepsilon_s \le S \le S_0 + \varepsilon_s, \qquad (3.28)$$

де a_0, b_0, c_0, S_0, $\left(s_p\right)_0$ – початкове наближення, в околі якого параметри $\varepsilon_a, \varepsilon_b, \varepsilon_c$ визначають нижні та верхні межі на невідомі коефіцієнти a,b,c, а параметр ε_s визначає нижні та верхні межі на невідомі довжини S та s_p. Відмітимо, що для тестових прикладів за початкові наближення можуть бути вибрані точки глобальних мінімумів, знайдені за допомогою модифікації *r*-алгоритму для задачі (3.21) – (3.23).

Нижче представлено результати збіжності методів L-BFGS-B та BFGS для обох оптимізаційних задач, яким відповідає оптимальний розв'язок 1 з таблиці 3.1 для S-подібної кривої. Для обчислювального експерименту використовувалися реалізації алгоритмів BFGS та L-BFGS-B на мові Python з бібліотеки SciPy. Критерієм розв'язання задач була вибрана умова, що значення функції, що мінімізується, менше ніж 10^{-9}, і норма градієнта менша ніж 10^{-10}.

У таблиці 3.7 наведено результати збіжності алгоритму L-BFGS-B для задачі (3.24) – (3.25) в залежності від вибору стартової точки, яка генерувалась за допомогою датчика випадкових чисел **np.random.seed(100)** у діапазонах (гіперкубах) $[0,1]^5$, $[0,10]^5$, $[0,20]^5$, $[0,30]^5$ та $[0,50]^5$. Нижня та верхня межі вибиралися за даними (3.8), де $S_{\min} = \sqrt{\left(x_2 - x_1\right)^2 + \left(y_2 - y_1\right)^2}$, а $S_{\max} = qS_{\min}$, де $q \in \{1.2, 1.5, 2.0\}$.

Таблиця 3.7

Збіжність методу L-BFGS-B для задачі (3.24) – (3.25)

гіперкуб	$q = 1.2$	$q = 1.5$	$q = 2.0$
$[0,1]^5$	50/50	50/50	50/50

гіперкуб	$q=1.2$	$q=1.5$	$q=2.0$
$[0,10]^5$	50/50	50/50	50/50
$[0,20]^5$	46/50	43/50	50/50
$[0,30]^5$	40/50	40/50	49/50
$[0,50]^5$	30/50	36/50	38/50

Із таблиці 3.7 видно, що із 50 запусків, алгоритм L-BFGS-B завжди збігається до точки глобального мінімуму задачі (3.24) – (3.25), якщо стартова точка локалізована у діапазонах $[0,1]^5$ та $[0,10]^5$. Для діапазонів $[0,20]^5$, $[0,30]^5$ та $[0,50]^5$ метод не завжди збігається до точки глобального мінімуму. Так, наприклад, для гіперкуба $[0,50]^5$ та $q=1.2$ алгоритм L-BFGS-B збігається до точки глобального мінімуму у тридцяти випадках із п’ятдесяти.

У таблиці 3.8 наведено результати порівняння збіжності методів BFGS та L-BFGS-B для задачі (3.26) – (3.28), де початкове наближення дорівнює першому розв’язку системи з таблиці 3.1, при різних значеннях параметрів $\varepsilon_a, \varepsilon_b, \varepsilon_c, \varepsilon_s$, які визначають нижні та верхні межі на змінні. Тут параметр $\varepsilon_s \le 0.15$ вибирається таким, щоб виконувалася нерівність $s_p \le S$.

Таблиця 3.8

Збіжність методів BFGS та L-BFGS-B для задачі (3.26) – (3.28)

ε_a	ε_b	ε_c	ε_s	**BFGS**	**L-BFGS-B**
20	20	20	0.15	79/100	99/100
20	20	20	0.01	82/100	97/100
10	10	10	0.15	98/100	92/100
10	10	10	0.1	97/100	96/100
10	10	10	0.01	100/100	100/100
5	5	5	0.1	100/100	100/100

Із таблиці 3.8 видно, що із зменшенням параметрів $\varepsilon_a, \varepsilon_b, \varepsilon_c, \varepsilon_s$ починає збільшуватися кількість успішних запусків, тобто методи збігаються до точки глобального мінімуму. Так, наприклад, для $\varepsilon_a = \varepsilon_b = \varepsilon_c = 10, \varepsilon_s = 0.01$ та $\varepsilon_a = \varepsilon_b = \varepsilon_c = 5, \varepsilon_s = 0.1$ обидва методи збігаються до точки глобального мінімуму у ста випадках із ста. Варто відмітити, що проективний варіант методу є більш стійким у випадку далеких стартових точок, яким відповідають параметри $\varepsilon_a, \varepsilon_b, \varepsilon_c \geq 20$.

Інтерактивна програма для побудови та аналізу плоскої кривої. Для розв'язання системи (3.14) – (3.18) та аналізу отриманого розв'язку розроблено інтерактивну програму мовою програмування Python. Користувач має можливість вводити початкові дані для системи (3.14) – (3.18) та налаштовувати параметри вибраного алгоритму розв'язання системи. В результаті роботи програма відображає на екран розв'язок системи (3.14) – (3.18) (якщо він існує), отриману криву та графіки її характеристик (кут нахилу дотичних, кривина та похідна від кривини). Для розв'язання системи (3.14) – (3.18) використовуються реалізації алгоритмів BFGS та L-BFGS-B бібліотеки SciPy. Для реалізації інтерфейсу використовуються бібліотеки tkinter та mathplotlib. Приклад діалогового вікна програми представлено на рисунку 3.2.

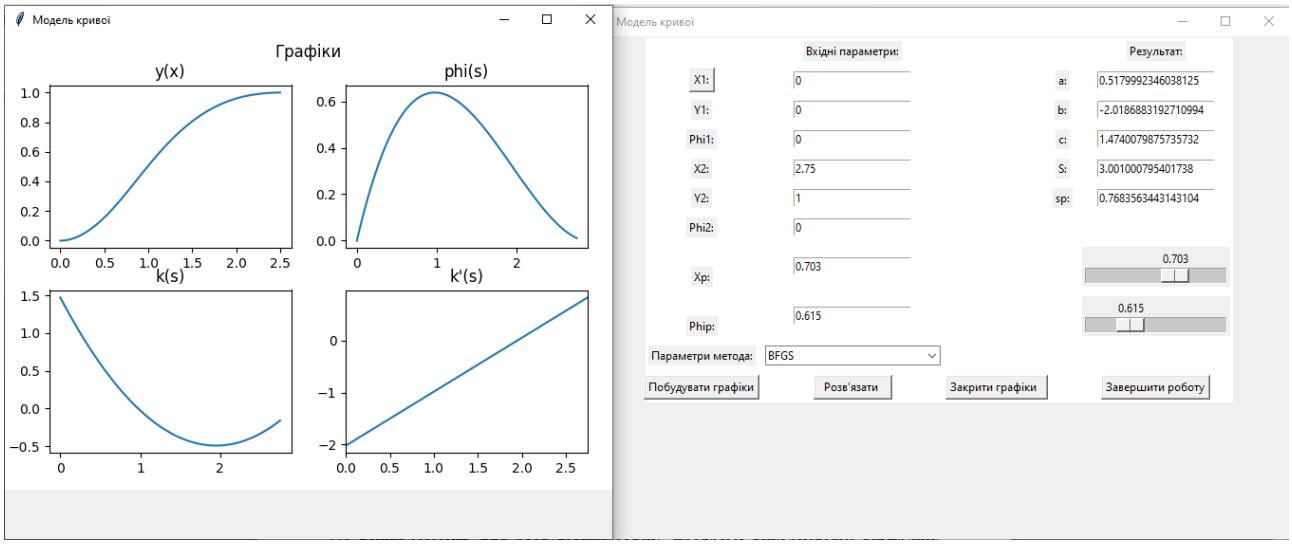


Рис. 3.2. Діалогове вікно інтерактивної програми для побудови та аналізу плоскої кривої

3.5 Висновки до третього розділу

1. Розглянуто модифікацію $r(\alpha)$-алгоритму з адаптивним регулюванням кроку за напрямком спуску, яка використовує прискорену реалізацію розтягу простору. Наведено схему модифікованого $r(\alpha)$-алгоритму та результати обчислювальних експериментів для задач мінімізації опуклих кусково-лінійних та квадратичних яружних функцій. На основі отриманих результатів зазначено, що порівняно зі звичайним $r(\alpha)$-алгоритмом, його модифікований варіант для мінімізації кусково-лінійної функції загалом потребує приблизно таку ж саму кількість ітерацій, але, при цьому, виконує значно меншу кількість операцій множення. Для мінімізації квадратичної строго опуклої функції кількість ітерацій і, як наслідок, час виконання програми, зменшуються.
2. Розглянуто модель квантильної регресії як задачу безумовної мінімізації опуклої негладкої функції. Наведено результати обчислювальних експериментів з використанням програмної реалізації $r(\alpha)$-алгоритму з адаптивним регулюванням кроку.
3. Розглянуто застосування методів BFGS та L-BFGS-B для задачі побудови S-подібної кривої. Ця задача відповідає мінімізації нелінійної гладкої функції, яка відповідає знаходженню розв’язків системи п’яти нелінійних рівнянь. Запропоновано оптимізаційну модель для задачі побудови s-подібної кривої, яка формулюється як задача мінімізації гладкої функції суми нев’язок з простими двосторонніми обмеженнями на змінні. Наведено результати обчислювальних експериментів з використанням методів BFGS та L-BFGS-B.

РОЗДІЛ 4. ЗАДАЧІ ЗНАХОДЖЕННЯ ПРОПУСКНИХ ЗДАТНОСТЕЙ ДУГ ВІДМОВОСТІЙКИХ МЕРЕЖ

У четвертому розділі розглядаються оптимізаційні задачі лінійного програмування, змішаного булевого лінійного програмування та нелінійного програмування для знаходження пропускних спроможностей дуг відмовостійких мереж. Наведено результати обчислювальних експериментів розв'язання булевих задач за допомогою відомої програми Gurobi. Розроблено декомпозиційні методи на основі *r*-алгоритму Шора та показано їх конкурентоспроможність з програмою IPOPT при розв'язанні задач нелінійного програмування.

4.1 Основні поняття для відмовостійкої мережі

Нехай $N=(V,A)$ – орієнтована мережа з множиною вершин V та множиною дуг A. Пропускну спроможність дуги $a=(i,j)\in A$, спрямованої з вершини $i\in V$ у вершину $j\in V$, позначимо y_a.

Означення. *Одиничною відмовою мережі назвемо такий її стан, коли пропускні спроможності дуг змінюються за правилом:*

$$y_a^{'}=\mu_a y_a,\ \forall a\in A,$$

якщо хоча б один з коефіцієнтів $\mu_a\in[0,1)$.

За допомогою цього поняття можна описати різноманітні аварійні ситуації, що мають місце у функціонуючих мережах будь-якого типу: автомобільних, залізничних, комунікаційних, енергетичних тощо. Множину одиничних відмов мережі будемо називати сценарієм відмов мережі та будемо її позначати буквою F (Fault) в комбінації з відповідним цьому сценарію набором цифр. Зауважимо, що для зручності одиничною відмовою мережі будемо вважати її

повноцінне функціонування, що буде мати місце, якщо для всіх дуг коефіцієнти $\mu_a = 1$.

Для мережі $Net(4,4)$ (рисунок 4.1) в таблиці 4.1 наведено сценарій відмов 0,5F, коли одна, але будь-яка дуга з чотирьох зменшує свою пропускну спроможність у два рази ($\mu_a = 0.5$). Якщо для сценарію 0,5F всі значення 0,5 замінити нульовими, то отримаємо новий сценарій відмов 1F, рівносильний повній відмові тільки однієї будь-якої дуги в мережі. На рисунку 4.2 наведено одиничні відмови (окремі пошкодження) у мережі $Net(4,4)$ для сценарю 1F.

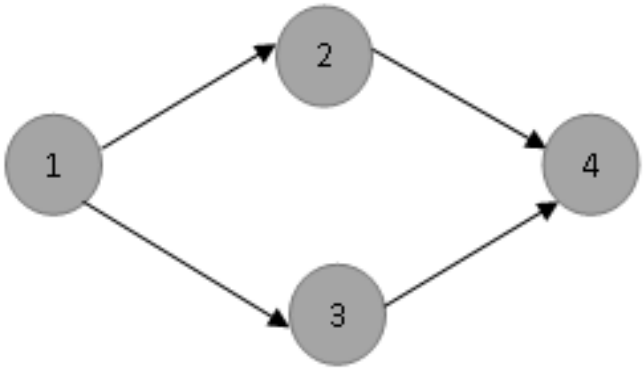


Рис. 4.1. Мережа $Net(4,4)$, 4 вершини і 4 дуги.

Таблиця 4.1

Сценарії 0F; 0,5F; 1F для мережі $Net(4,4)$

№	Дуги	0F	0,5F				1F			
	(i, j)	μ_0	μ_1	μ_2	μ_3	μ_4	μ_5	μ_6	μ_7	μ_8
1	$(1,2)$	1,0	0,5	1,0	1,0	1,0	0,0	1,0	1,0	1,0
2	$(1,3)$	1,0	1,0	0,5	1,0	1,0	1,0	0,0	1,0	1,0
3	$(2,4)$	1,0	1,0	1,0	0,5	1,0	1,0	1,0	0,0	1,0
4	$(3,4)$	1,0	1,0	1,0	1,0	0,5	1,0	1,0	1,0	0,0

Зміст задач знаходження пропускних спроможностей дуг та вершин для відмовостійкої мережі полягає в наступному.

Задані:

1. Мережа $N=(V,A)$; y_a^0 – значення існуючих пропускних спроможностей дуг, $a\in A$; y_a^{low}, y_a^{up} – нижня та верхня межі на пропускні спроможності дуг, $a\in A$.

2. Мережевий трафік K – обсяги потоку в мережі; d_k, $k\in K$ – обсяг потоку від вершини $s(k)\in V$ до вершини $r(k)\in V$, які будемо називати джерелом (sender) та стоком (receiver), відповідно.

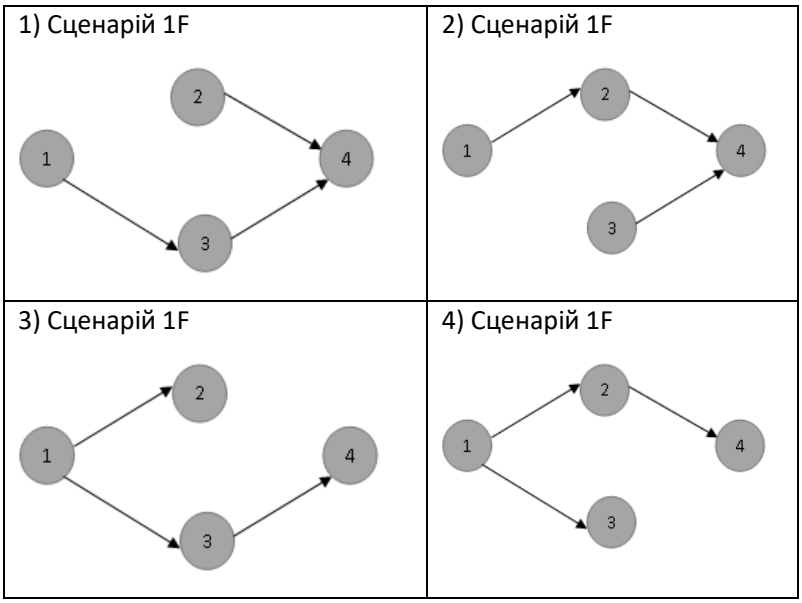


Рис. 4.2. Одиничні відмови мережі $Net(4,4)$ для сценарію 1F.

3. Сценарії відмов T для мережі $N=(V,A)$:

μ_{at}, $\mu_{at}\in[0,1]$, $\forall a\in A$, $\forall t\in T$. Якщо $\mu=0$, то це рівнозначно відмові дуги a. Якщо $\mu=1$, то це означає що дуга a не вімовляє.

Потрібно знайти: оптимальні за критерієм (розглянемо його пізніше) модернізації пропускні спроможності дуг y_a^*, $a\in A$ (що додаються до вже

існуючих y_a^0), при яких забезпечується заданий обсяг трафіку K в мережі $N=(V,A)$, якщо станеться одна, але довільна, одинична відмова зі сценарію відмов T.

Для мережі $N=(V,A)$ будемо розглядати два типи задач знаходження пропускних спроможностей дуг.

Задача А: для передачі потоків залучаються всі можливі шляхи в мережі.

Задача Р: для передачі потоків залучаються лише шляхи із заданої множини шляхів P. Тут $P=\bigcup_{k\in K} P_k$, де P_k – множина шляхів для потоку k.

Іншими словами, пересилання потоків в задачі А визначається фізичною (визначена вершинами та дугами орієнтовної мережі) структурою мережі, а в задачі Р визначається логічною (визначена набором можливих шляхів) структурою мережі.

Далі опишемо базові задачі лінійного програмування (ЛП-задачі А та Р), які будуть використані при побудові для задач А та Р відповідних моделей булевого лінійного програмування та нелінійного програмування. Розв'язки задач модернізації пропускних спроможностей дуг буде проілюстровано на прикладі орієнтованої мережі $Net(6,16)$ з шістьма вершинами та 16 дугами (рисунок 4.3), яка розглядалася при описі математичного забезпечення для задач проектування надійних мереж (Шор та ін. 2005).

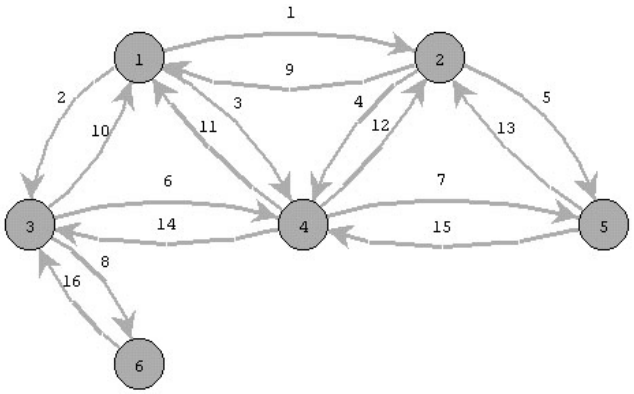


Рис. 4.3. Структура мережі $Net(6,16)$

Вартості створення одиниці пропускної спроможності для дуг (1,2) та (2,1) будуть рівними 1.5, а для всіх інших дуг – рівними 1. Будемо вважати, що між усіма парами вершин потрібно переслати по мережі один і той же обсяг потоку, що дорівнює десяти одиницям. Які із сценаріїв відмов у мережі $Net(6,16)$ із книги (Шор та ін., с. 114) будуть використані, будемо уточнювати для кожного окремого розрахунку тієї чи іншої задачі.

4.2 Базові ЛП-задачі A та P

Нехай c_v, c_a – вартість створення одиниці пропускних спроможностей для вершини v та дуги a, A_i^+ та A_i^- – множини дуг, що входять і виходять з вершини v, $v \in V$. Тоді задача A має такий вигляд: знайти

$$c_a^* = \min_{x,y} \sum_{a \in A} c_a y_a \tag{4.1}$$

за обмежень

$$\sum_{k \in K} x_{akt} \le \mu_{at} \left(y_a^0 + y_a \right), \quad \forall a \in A, \ \forall t \in T, \tag{4.2}$$

$$\sum_{a \in A_i^+} x_{akt} - \sum_{a \in A_i^-} x_{akt} = \begin{cases} d_k, & \text{якщо } i = s(k); \quad \forall i \in V, \\ -d_k, & \text{якщо } i = r(k); \quad \forall k \in K, \\ 0 \text{ у протилежному випадку}; & \forall t \in T, \end{cases} \tag{4.3}$$

$$x_{akt} \ge 0, \quad \forall a \in A, \ \forall k \in K, \ \forall t \in T, \tag{4.4}$$

$$y_a^{low} \le y_a \le y_a^{up}, \quad \forall a \in A. \tag{4.5}$$

Тут змінні x_{akt} позначають невідомий потік продукту k по дузі a при пошкодженні t, $s(k) = sender(k)$, $r(k) = receiver(k)$. Змінні y_a позначають невідомі значення пропускних спроможностей дуг $a \in A$, що додаються до вже існуючих пропускних спроможностей y_a^0.

Формулювання задачі P має такий вигляд: знайти

$$c_P^* = \min_{z,y} \sum_{a \in A} c_a y_a \tag{4.6}$$

за обмежень

$$\sum_{k\in K}\sum_{p\in P_k}\delta_{kpa}z_{kpt}\le\mu_{at}\left(y_a^0+y_a\right),\quad \forall a\in A,\ \forall t\in T, \tag{4.7}$$

$$\sum_{p\in P_k}z_{kpt}=d_k\quad \forall k\in K,\forall t\in T, \tag{4.8}$$

$$z_{kpt}\ge 0,\quad \forall k\in K,\ \forall p\in P,\ \forall t\in T, \tag{4.9}$$

$$y_a^{low}\le y_a\le y_a^{up},\quad \forall a\in A, \tag{4.10}$$

де

$$\delta_{kpa}=\begin{cases}1, & \textit{коли шлях } p\in P_k \textit{ включає дугу } a;\\ 0 & \textit{у протилежному випадку.}\end{cases}$$

Тут змінні z_{kpt} позначають потік продукту $k\in K$ по шляху $p\in P_k$ при пошкодженні $t\in T$. Змінні y_a такі ж як і для задачі (4.1) – (4.5). Зміст ЛП-задач (4.1) – (4.5) та (4.6) – (4.10) досить прозорий. Цільові функції (4.1) та (4.6), які мінімізуються, задають сумарні витрати на вартість тих пропускних спроможностей (що доповнюють вже існуючі) дуг, які потрібно знайти з метою забезпечення безвідмовної роботи мережі $N(V,A)$. Зміст обмежень у ЛП-задачах А та Р такий: обмеження (4.2) й (4.7) означають, що потоки по дугах не повинні перевищувати пропускних спроможностей дуг при довільній одній одиничній відмові зі сценарію відмов; обмеження (4.3) й (4.8) гарантують виконання умов по мережевому трафіку; обмеження (4.4) й (4.9) відповідають за невід'ємність потоків, а нерівності (4.5) й (4.10) накладають двосторонні обмеження на вибір пропускних спроможностей дуг, що додаються до уже існуючих.

Для ЛП-задачі (4.1) – (4.5) кількість змінних N_A та кількість обмежень M_A (без урахування найпростіших обмежень (4.4) – обмежень на невід’ємність змінних x_{akt}) визначаються за формулами

$$N_A=|A|*|K|*|T|+|A|,\quad M_A=|A|*|T|+|V|*|K|*|T|+2*|A|, \tag{4.11}$$

де для N_A перший доданок задає кількість змінних x_{akt}, другий – кількість змінних y_a, а для M_A перший доданок задає кількість обмежень (4.2), другий доданок – кількість обмежень (4.3), третій доданок – кількість обмежень (4.5).

Для ЛП-задачі (4.6) – (4.10) кількість змінних N_P та кількість обмежень M_P (без урахування найпростіших обмежень (4.9) – обмежень на невід'ємність змінних z_{kpt}) визначається за формулами

$$N_P = |K| * |P| * |T| + |A|, \quad M_P = |A| * |T| + |K| * |T| + 2 * |A|, \qquad (4.12)$$

де для N_P перший доданок задає кількість змінних z_{kpt}, другий – кількість змінних y_a, а для M_P перший доданок задає кількість обмежень (4.7), другий доданок – кількість обмежень (4.8), третій доданок – кількість обмежень (4.10).

Якщо мережа $N(V, A)$ має відносно невеликі розміри (20 вершин, 100 дуг), то ЛП задачі А та Р мають дуже велику вимірність. Так, наприклад, якщо $|A| \approx 100$, $|K| \approx 400$, $|V| \approx 20$, $|T| \approx 1000$, то у ЛП-задачі А кількість змінних N_A та кількість обмежень M_A мають порядок 10^7. Якщо при цьому ще й $|P| \approx 10000$, то у ЛП-задачі Р кількість змінних N_P має порядок 10^8 та кількість обмежень M_P мають порядок 10^5.

У цьому випадку ЛП-задачі (4.1) – (4.5) та (4.6) – (4.10) неможливо розв'язати навіть за допомогою найкращих сучасних програм для задач лінійного програмування, наприклад CPLEX та Gurobi, тому що це вимагає дуже значних обчислювальних ресурсів. Однак в обох ЛП-задачах матриця обмежень має вкладену блочну структуру (рисунок 4.4, нульові блоки помічено білим кольором), що дає можливість розробляти декомпозиційні алгоритми розв'язування ЛП-задач А та Р і для більших, ніж наведені вище, розмірів мережі, наприклад, мережа може містити 100 вершин та 1000 дуг.

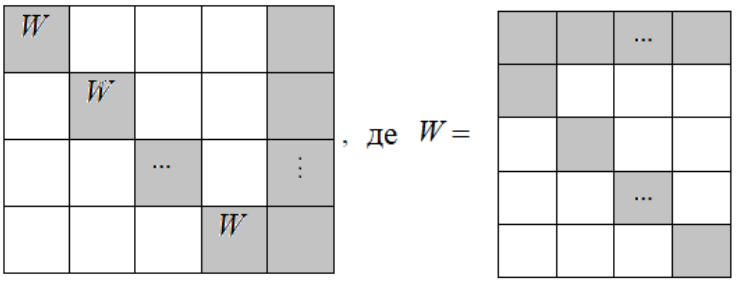


Рис. 4.4. Вкладена блочна структура матриці обмежень
ЛП-задач А та Р.

Такі декомпозиційні алгоритми будуть описані далі як для лінійної цільової функції, так і для опуклих гладкої та негладкої цільових функцій.

Нижче для задач А та Р описуються моделі змішаного булевого лінійного програмування, які враховують витрати на створення нових дуг та визначають ті дуги, пропускні спроможності яких потрібно модернізувати у відмовостійкій мережі.

4.3 Булеві задачі А та Р

Математичні моделі відповідних задач змішаного булевого лінійного програмування (булевих задач А та Р) легко отримати, якщо у задачах (4.1) – (4.5) та (4.6) – (4.10) цільові функції (4.1) і (4.6) замінити на цільові функції

$$F_A^* = \min_{x,y,u} \sum_{a\in A} C_a u_a + \sum_{a\in A} c_a y_a \,, \tag{4.13}$$

$$F_P^* = \min_{z,y,u} \sum_{a\in A} C_a u_a + \sum_{a\in A} c_a y_a \,, \tag{4.14}$$

а обмеження (4.5) та (4.10) замінити на обмеження

$$y_a^{low} u_a \le y_a \le y_a^{up} u_a, \quad \forall a \in A, \tag{4.15}$$

де C_a – витрати на створення доданих пропускних спроможностей дуг, а булеві змінні $u_a = 0 \vee 1$ дорівнюють одиниці, якщо дуга $a \in A$ додається до мережі, і

нулю в протилежному випадку. Тоді формулювання булевої задачі А має такий вигляд: знайти

$$F_A^* = \min_{x,y,u} \sum_{a \in A} C_a u_a + \sum_{a \in A} c_a y_a \tag{4.16}$$

за обмежень

$$\sum_{k \in K} x_{akt} \le \mu_{at} \left(y_a^0 + y_a \right), \quad \forall a \in A, \ \forall t \in T, \tag{4.17}$$

$$\sum_{a \in A_i^+} x_{akt} - \sum_{a \in A_i^-} x_{akt} = \begin{cases} d_k, & \text{якщо } i = s(k); \quad \forall i \in V, \\ -d_k, & \text{якщо } i = r(k); \quad \forall k \in K, \\ 0 \text{ у протилежному випадку}; & \forall t \in T, \end{cases} \tag{4.18}$$

$$x_{akt} \ge 0, \quad \forall a \in A, \ \forall k \in K, \ \forall t \in T, \tag{4.19}$$

$$y_a^{low} u_a \le y_a \le y_a^{up} u_a, \quad \forall a \in A. \tag{4.20}$$

Формулювання булевої задачі Р має такий вигляд: знайти

$$F_P^* = \min_{z,y,u} \sum_{a \in A} C_a u_a + \sum_{a \in A} c_a y_a \tag{4.21}$$

за обмежень

$$\sum_{k \in K} \sum_{p \in P_k} \delta_{kpa} z_{kpt} \le \mu_{at} \left(y_a^0 + y_a \right), \quad \forall a \in A, \ \forall t \in T, \tag{4.22}$$

$$\sum_{p \in P_k} z_{kpt} = d_k \quad \forall k \in K, \forall t \in T, \tag{4.23}$$

$$z_{kpt} \ge 0, \quad \forall k \in K, \ \forall p \in P, \ \forall t \in T, \tag{4.24}$$

$$y_a^{low} u_a \le y_a \le y_a^{up} u_a, \quad \forall a \in A. \tag{4.25}$$

Якщо $C_a = 0$, то розв'язки булевих задач (4.16) – (4.20) та (4.21) – (4.25) співпадають з розв`язками ЛП-задач (4.1) – (4.5) та (4.6) – (4.10). Якщо $C_a > 0$, то мінімізуватися будуть сумарні витрати на вартість та створення пропускних спроможностей дуг, що доповнюють уже існуючі у мережі. Вибір відповідних дуг мережі буде залежати від коефіцієнтів $C_a > 0$, $a \in A$.

Один із можливих способів розв'язання задач (4.16) – (4.20) і (4.21) – (4.25) полягає у представленні їх на мові моделювання AMPL (A Mathematical

Programming Language) (Fourer, Gay, and Kernighan 2003) та використанні сучасного програмного забезпечення для розв’язання задач цілочислового лінійного програмування, наприклад, програми Gurobi (Gurobi 2023) з NEOS-сервера (NEOS Server 2023). У таблицях 4.2 та 4.3 наведено результати тестування програми Gurobi для розв'язання задачі (4.16) – (4.20) на прикладі орієнтованої мережі Net(6,16) (рисунок 4.3). Тут сценарії відмов відповідають сценаріям (a) – (f) з книги (Шор та ін. 2005, с. 114–115):

a) відсутність одиничних відмов;

b) відмовити може будь-яка, але одна дуга, за винятком дуг, зв'язаних з вершиною 6;

c) одна відмова, коли одночасно відмовляють чотири дуги: (1,2), (2,1), (2,4) та (4,2);

f) будь-яка з дуг, але тільки одна, може зменшити свою пропускну спроможність в два рази.

Таблиця 4.2

Мінімальні за сумарною вартістю пропускні спроможності дуг мережі Net(6,16)

N	(i,j)	c_{ij}	$y_a^0 = y_{ij}^0 = 0$				$y_{ij}^0 = \left(y_{ij}^*\right)_a$		
			a	b	c	f	b	c	f
1	(1, 2)	1.5	10	50	0	46.66	40	0	20
2	(1, 3)	1	20	80	20	53.33	60	0	30
3	(1, 4)	1	20	40	30	16.66	20	10	10
4	(2, 4)	1	30	30	0	10	0	0	0
5	(2, 5)	1	10	50	50	26.66	40	40	20
6	(3, 4)	1	60	80	60	53.33	20	0	0
7	(4, 5)	1	40	50	80	46.66	10	40	0
8	(3, 6)	1	50	50	50	100	0	0	50

9	(2, 1)	1.5	10	50	0	46.66	40	0	20
10	(3, 1)	1	20	80	20	53.33	60	0	30
11	(4, 1)	1	20	40	30	16.66	20	10	10
12	(4, 2)	1	30	30	0	10	0	0	0
13	(5, 2)	1	10	50	50	26.66	40	40	20
14	(4, 3)	1	60	80	60	53.33	20	0	0
15	(5, 4)	1	40	50	80	46.66	10	40	0
16	(6, 3)	1	50	50	50	100	0	0	50
$\sum_{(i,j)\in A} c_{ij} y_{ij}^*$			490	910	580	753.33	420	180	280

В таблиці 4.2 наведено два варіанти мінімальних за сумарною вартістю пропускних спроможностей дуг мережі Net(6,16) при сценаріях відмов (a), (b), (c), (f), яким відповідають нульові значення $C_a = 0$. Вони отримані програмою Gurobi при двох різних значеннях існуючих пропускних спроможностей – нульових значеннях $y_a^0 = y_{ij}^0 = 0$ та ненульових значеннях $y_{ij}^0 = \left(y_{ij}^*\right)_a$, які отримано для сценарію (a) при існуючих пропускних спроможностях $y_a^0 = y_{ij}^0 = 0$. При цьому час розв'язання дорівнював декільком секундам.

Зауважимо, що результати розрахунку в таблиці 4.2 співпадають з результатами розрахунків із книги (Шор та ін. 2005, с. 116–117), що і повинно бути, так як ЛП-задачі А із книги (Шор та ін. 2005), співпадають з ЛП-задачею (4.1) – (4.5).

Таблиця 4.3

Мінімальні за сумарними витратами на вартість та створення пропускні спроможності дуг мережі Net(6,16)

N	(i, j)	C_a	Без активних обмежень (15a)	Активна дуга $y_{11}^{up} = 20$

			a	b	c	f	a	b	C
1	(1, 2)	10	0	50	0	33.33	0	50	0
2	(1, 3)	10	20	80	20	43.33	20	80	20
3	(1, 4)	10	30	40	30	30	30	40	30
4	(2, 4)	10	40	30	0	0	40	0	0
5	(2, 5)	10	10	50	50	33.33	0	50	50
6	(3, 4)	10	60	80	60	53.33	60	80	60
7	(4, 5)	10	40	50	80	33.33	50	50	80
8	(3, 6)	10	50	50	50	100	50	50	50
9	(2, 1)	10	0	50	0	53.33	10	80	0
10	(3, 1)	10	20	80	20	53.33	20	80	30
11	(4, 1)	10	30	40	30	0	20	10	20
12	(4, 2)	10	50	30	0	30	50	30	0
13	(5, 2)	10	1.30 E-12	50	50	23.33	1.90 E-12	50	50
14	(4, 3)	10	60	80	60	73.33	60	80	70
15	(5, 4)	10	50	50	80	53.33	50	80	80
16	(6, 3)	10	50	50	50	100	50	50	50
$\sum_{a\in A_a} C_a u_a + \sum_{a\in A_a} c_a y_a$			640	1070	700	896.67	645	1075	710

В таблиці 4.3 наведено оптимальні (мінімальні за сумарними витратами на вартість та створення) додані пропускні спроможності дуг мережі Net(6,16) для ненульових значень $C_a = 10$ при тих же сценаріях відмов (a), (b), (c), (f). Вони отримані за допомогою програми Gurobi для варіанту без активних обмежень (4.20) та для варіанту з обмеженням зверху на пропускну спроможність для

всього однієї дуги $y_{11}^{up} = y_{(4,1)}^{up} = 20$. Для варіанту з обмеженням зверху на пропускну спроможність в таблиці 4.3 відсутній варіант (f). Це пов'язано з тим, що розв'язок задачі для варіанту (f) без активних обмежень (4.20) повністю співпадає з розв'язком для варіанту (f) з обмеженням для дуги 11. Із таблиці 4.3 видно, що активне обмеження зверху на пропускну спроможність навіть однієї дуги (зменшено на 10 одиниць порівняно з зі значенням із розв'язку задачі для варіанту (а) без активних обмежень (4.20)) може призвести до досить суттєвого перерозподілу оптимальних пропускних спроможностей дуг (дуги 4,5,9). Час розв'язання задач для таблиці 4.3 також сягав декількох секунд.

4.4 Опуклі задачі А та Р. Декомпозиційні алгоритми

Математичні моделі задач знаходження пропускних спроможностей дуг відмовостійкої мережі можуть представлені як опуклі оптимізаційні задачі, в яких мінімізується цільова опукла функція (може бути як гладкою, так і негладкою) від невідомих $Y = \{y_a, \forall a \in A\}$ – пропускних спроможностей дуг мережі. Частковим випадком опуклих задач є ЛП-задачі А та Р, для них опукла функція є лінійною функцією $f_1(Y) = \sum_{a \in A} c_a y_a$ та мінімізує сумарні витрати за вартістю пропукних спроможностей дуг відмовостійкої мережі. Квадратична гладка функція $f_2(Y) = \sum_{a \in A} \left(y_a - y_a^e \right)^2$ і негладка функція $f_3(Y) = \sum_{a \in A} \left| y_a - y_a^e \right|$, де y_a^e, $a \in A$, – деякі «бажані» пропускні спроможності дуг, дозволяють знайти пропускні спроможності дуг мережі з мінімальним відхиленням від «бажаних» пропускних спроможностей дуг за критерієм найменших квадратів та критерієм найменших модулів. Нижче розглянемо задачі мінімізації опуклої функції $f(Y) = f_i(Y)$, де $i \in \{1,2,3\}$, при обмеженнях ЛП-задач, прибравши в (4.5) та (4.10) двосторонні обмеження на невідомі пропускні спроможності дуг.

Тоді формулювання опуклої задачі А має такий вигляд: знайти

$$f_A^* = \min_{x,y}\left\{ f(Y) + \sum_{t\in T}\sum_{a\in A} Q_a y_{at} \right\}, \tag{4.26}$$

за обмежень

$$\sum_{k\in K} x_{akt} - y_{at} \le \mu_{at}\left(y_a^0 + y_a\right), \quad \forall a \in A,\ \forall t \in T, \tag{4.27}$$

$$\sum_{a\in A_i^+} x_{akt} - \sum_{a\in A_i^-} x_{akt} = \begin{cases} d_k, & \text{якщо } i = s(k); \quad \forall i \in V, \\ -d_k, & \text{якщо } i = r(k); \quad \forall k \in K, \\ 0 \text{ у протилежному випадку}; & \forall t \in T, \end{cases} \tag{4.28}$$

$$x_{akt} \ge 0, \quad \forall a \in A,\ \forall k \in K,\ \forall t \in T, \tag{4.29}$$

$$y_{at} \ge 0, \quad \forall a \in A, \forall t \in T,\ \ y_a \ge 0, \quad \forall a \in A. \tag{4.30}$$

Формулювання опуклої задачі P має такий вигляд: знайти

$$f_P^* = \min_{z,y}\left\{ f(Y) + \sum_{t\in T}\sum_{a\in A} Q_a y_{at} \right\}, \tag{4.31}$$

за обмежень

$$\sum_{k\in K}\sum_{p\in P_k} \delta_{kpa} z_{kpt} - y_{at} \le \mu_{at}\left(y_a^0 + y_a\right), \quad \forall a \in A,\ \forall t \in T, \tag{4.32}$$

$$\sum_{p\in P_k} z_{kpt} = d_k \quad \forall k \in K, \forall t \in T, \tag{4.33}$$

$$z_{kpt} \ge 0, \quad \forall k \in K,\ \forall p \in P,\ \forall t \in T, \tag{4.34}$$

$$y_{at} \ge 0, \quad \forall a \in A, \forall t \in T,\ \ y_a \ge 0, \quad \forall a \in A. \tag{4.35}$$

Для забезпечення сумісності систем обмежень введено додаткові невід'ємні змінні $y_{at} \ge 0$, $\forall a \in A$, $\forall t \in T$. Вони мають такий зміст: y_{at} відповідає тому значенню пропускної спроможності дуги $a \in A$, якого не вистачає при виникненні t-ї одиничної відмови мережі для виконання обмежень (4.27) та (4.32), що відповідають цим $a \in A$ та $t \in T$. «Штрафні» множники Q_{at} вибираються такими, щоб оптимальні значення y_{at}^* дорівнювали нулю. Якщо структура одиничних відмов така, що відмовостійке функціонування мережі $N(V, A)$ неможливе, то ненульові оптимальні значення

y^*_{at} будуть характеризувати ті критичні місця в мережі $N(V,A)$, через які неможливо забезпечити її відмовостійке функціонування.

З структури опуклих задач (рисунок 4.4) легко бачити, що змінні $Y=\{y_a, \forall a\in A\}$ є зв'язуючими в обох задачах. Тому для їх розв'язання доцільно використовувати схему декомпозиції за змінними Y (Шор 1979). При цьому координуючі (зовнішні) задачі полягають в мінімізації негладких опуклих функцій $F_A(Y)$ та $F_P(Y)$ від зв'язуючих змінних Y і для знаходження їх мінімумів можна застосовувати r-алгоритм (Шор 1979), (Stetsyuk 2017a), (Стецюк 2017), який вважається одним із ефективних методів негладкої оптимізації (Стецюк та ін. 2021), (Стецюк 2014), (Sergienko 2012).

Внутрішня підзадача, яку потрібно розв'язати для обчислення субградієнта функції $F_A(Y)$, буде пов'язана з пошуком значень двоїстих змінних до обмежень (4.37) для задачі лінійного програмування такого вигляду: знайти

$$\varphi^*_A(t)=\min_{x,y}\sum_{a\in A} Q_a y_{at} \tag{4.36}$$

за обмежень

$$\sum_{k\in K} x_{akt}-y_{at}\le \mu_{at}\left(y^0_a+\bar{y}_a\right),\quad \forall a\in A, \tag{4.37}$$

$$\sum_{a\in A_i^+} x_{akt}-\sum_{a\in A_i^-} x_{akt}=\begin{cases} d_k, & \textit{якщо } i=s(k);\\ -d_k, & \textit{якщо } i=r(k);\ \forall i\in V,\ \forall k\in K,\\ 0 & \textit{у протилежному випадку;}\end{cases} \tag{4.38}$$

$$x_{akt}\ge 0,\quad \forall a\in A,\ \forall k\in K, \tag{4.39}$$

$$y_{at}\ge 0,\quad \forall a\in A. \tag{4.40}$$

Де $\bar{y}_a$ – відомі поточні (на даний момент) значення пропускних спроможностей дуг. При обчисленні субградієнта функції $F_A(Y)$ внутрішню підзадачу потрібно розв'язувати $|T|$ раз.

Для розв'язання внутрішньої підзадачі (4.36) – (4.40) можна застосувати як схему декомпозиції за обмеженнями з використанням r-алгоритму, так і

стандартні програми для розв'язування задач лінійного програмування (наприклад, методи внутрішніх точок (Дикин и Зоркальцев 1980), (Зоркальцев, Пержабинский, и Стецюк 2015), що дозволяють врахувати блочну структуру підзадачі). При використанні *r*-алгоритму потрібна тільки підпрограма для знаходження найкоротших шляхів в орієнтованій мережі.

Щоб обчислити субградієнт функції $F_P(Y)$ потрібно $|T|$ раз розв'язати підзадачу знаходження двоїстих змінних до обмежень (4.42) для такої задачі лінійного програмування: знайти

$$\varphi_A^*(t) = \min_{x,y} \sum_{a \in A} Q_a y_{at} \tag{4.41}$$

при обмеженнях:

$$\sum_{k \in K} \sum_{p \in P_k} \delta_{kpa} z_{kpt} - y_{at} \le \mu_{at} \left(y_a^0 + \bar{y}_a \right), \ \forall a \in A, \tag{4.42}$$

$$\sum_{p \in P_k} z_{kpt} = d_k \quad \forall k \in K, \tag{4.43}$$

$$z_{kpt} \ge 0, \quad \forall k \in K, \ \forall p \in P, \tag{4.44}$$

$$y_{at} \ge 0, \ \ \forall a \in A, \tag{4.45}$$

де $\bar{y}_a$ – відомі поточні значення пропускних спроможностей дуг, що доповнюють уже існуючі.

Підзадача (4.41) – (4.45) є простішою, ніж відповідна підзадача (4.36) – (4.40), і для її розв'язування можна використовувати r-алгоритм у комбінації зі схемою декомпозиції за обмеженнями (4.42).

4.5 Програмне забезпечення: програми SolverA та SolverP

Для розв’язання опуклих задач знаходження оптимальних пропускних спроможностей дуг для побудови відмовостійкої орієнтованої мережі реалізовані такі програми: програма **SolverA** для задачі (4.26) – (4.30) та програма **SolverP** для задачі (4.31) – (4.35) (мова програмування ФОРТРАН**).**

Алгоритми розв'язування обох задач базуються на подвійному використанні двох схем декомпозиції (одна в одній): схема декомпозиції за змінними, якими є невідомі значення доданих пропускних спроможностей дуг орієнтованої мережі; схема декомпозиції за обмеженнями (для розв'язання підзадач, які виникають для кожної одиничної відмови при фіксованих значеннях доданих пропускних спроможностей дуг). Відповідні вказаним схемам декомпозиції задачі негладкої оптимізації розв'язуються за допомогою r-алгоритму з адаптивним регулюванням крокового множника, де параметри r-алгоритму вибрані таким чином: коефіцієнт розтягу простору $\alpha = 4$, а параметри адаптивного регулювання крокового множника – $n_h = 3$, $q_1 = 1$, $q_2 = 1.1$. Максимальна кількість ітерацій, відведена r-алгоритму для розв'язування підзадач, вибрана рівною 500.

В результаті роботи програм SolverA та SolverP отримуємо такі параметри відмовостійкої орієнтованої мережі $N(V, A)$: мінімальні по сумарній вартості пропускні спроможності дуг мережі $N(V, A)$, що доповнюють вже існуючі, тобто для кожної дуги $a \in A$ знаходиться y_a^* – ресурс пропускної спроможності дуги a, що доповнює вже існуюче значення пропускної спроможності цієї дуги y_a^0; достатня умова того, що неможливо виконати усі вимоги на передачу об'ємів потоків, щоб орієнтована мережа була відмовостійкою. Це може бути з двох причин: (a) структура мережі $N(V, A)$ така, що задовільнити вимоги до потоків неможливо; (b) структура одиничних відмов мережі $N(V, A)$ така, що задовільнити вимоги до потоків неможливо. Достатньою умовою невиконання усіх вимог на передачу об'ємів потоків в мережі є нерівність $\sum_{t \in T} \sum_{a \in A} Q_{at} y_{at}^* \gg 0$, яка має місце тоді і тільки тоді, коли в f_A^* або f_P^* вносить ненульовий вклад «штрафна» частина цільової функції.

RATFOR програму SolverA наведено в додатку А. Загальну структуру програми SolverA наведено на рисунку 4.5, де вказані блоки виконують такі функції.

Головна програма SolverA керує процесом знаходження розв'язку опуклої задачі А. В програмі встановлюються штрафні параметри (на основі вхідних даних), запускається процесс розв'язування задачі (4.26) – (4.30), аналізується отриманий розв'язок, результати розрахунку виводяться в файл протоколу роботи програми. Програма SolverA використовує підпрограми ReadDataA та CoordA.

Підпрограма ReadDataA читає вхідні дані задачі та перевіряє їх коректність за рядом ознак. Вхідні дані читаються послідовно з трьох файлів, які містять: (a) структуру орієнтованої мережі; (b) об'єми потоків між парами вершин, що пересилаються по мережі; (c) сценарій відмов у мережі.

Підпрограма CoordA реалізує роботу r-алгоритму для розв'язування координуючої негладкої задачі, пов'язаної зі схемою декомпозиції за змінними (використовує підпрограму FGCoord). Підпрограма FGCoord обчислює значення негладкої координуючої функції та її субградієнта, для чого послідовно для кожного з пошкоджень використовується підпрограма DualA.

Підпрограмма DualA знаходить двоїсті змінні до обмежень (4.37) підзадачі (4.36) – (4.40). Для їх знаходження за допомогою r-алгоритму розв'язується задача максимізації негладкої ввігнутої функції, що відповідає схемі декомпозиції за обмеженнями (4.37). Підпрограма FGDual обчислює значення функції та її субградієнта для підпрограми DualA, для чого використовується метод знаходження найкоротшого шляху в орієнтованій використовується метод знаходження найкоротшого шляху в орієнтованій мережі (Ahuja, Magnanti, and Orlin 1993, p. 137).

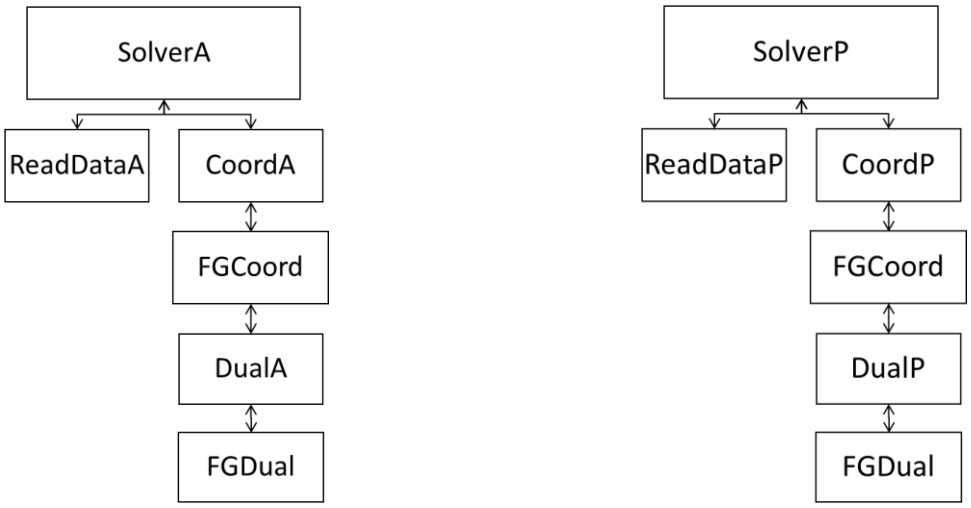


Рис. 4.5. Блок-схеми програм SolverA та SolverP

За аналогічною схемою, реалізовано програму SolverP (рисунок 4.5), де відповідні блоки виконують такі функції.

Головна програма SolverP керує процесом знаходження розв'язку опуклої задачі P. В програмі встановлюються штрафні параметри (на основі вхідних даних), запускається алгоритм розв'язування задачі (4.31) – (4.35), аналізується отриманий розв'язок, результати розрахунку виводяться в файл протоколу роботи програми. Програма SolverP використовує підпрограми ReadDataP та CoordP.

Підпрограма ReadDataP читає вхідні дані задачі та перевіряє їх коректність. Вхідні дані читаються послідовно з чотирьох файлів, які містять: (a) структуру орієнтованої мережі; (b) об'єми потоків, що пересилаються по мережі; (c) структуру пошкоджень мережі; (d) допустимі шляхи для передачі потоків в мережі.

Підпрограма CoordP реалізує роботу *r*-алгоритму для розв'язування координуючої негладкої задачі, пов'язаної зі схемою декомпозиції за змінними *Y* для задачі (4.31) – (4.35). Підпрограма FGCoord обчислює значення негладкої координуючої функції та її субградієнта, які використовуються програмою

CoordP. Для обчислення цих значень послідовно для кожного з пошкоджень використовується підпрограма DualC.

Підпрограмма DualC знаходить двоїсті змінні до обмежень (4.42) підзадачі (4.41) – (4.45) за допомогою *r*-алгоритму. Підпрограма FGDual обчислює значення функції та суперградієнта для підпрограми DualC.

Тестування та дослідження ефективності програм SolverA та SolverP проводилося для описаних вище лінійної функції $f_1(Y)$, квадратичної функції $f_2(Y)$ та нелінійної негладкої функції $f_3(Y)$ на прикладі орієнтованої мережі Net(6,16). Для порівняння було вибрано відому програму IPOPT (Ipopt 2023) з NEOS-сервера. Результати тестування показали, що час розв'язання опуклих тестових задач з одиничними відмовами дуг за допомогою програми SolverA може бути суттєво менший за час розв'язання таких задач за допомогою програми IPOPT. Так, наприклад, для сценарію b) час розв'язання задачі A з функцією $f_1(Y)$ за допомогою програми SolverA дорівнює 2.5 сек., з функцією $f_2(Y)$ – 2.7 сек., з функцією $f_3(Y)$ – 2.6 сек., а за допомогою програми IPOPT відповідно 97 сек., 424 сек. та 99 сек. При цьому в усіх випадках було знайдено оптимальні розв`язки.

4.6 Висновки до четвертого розділу

1. Розглянуто математичні моделі двох класів задач знаходження пропускних спроможностей дуг відмовостійкої орієнтованої мережі. У першому класі задач (задача A) для передачі потоків можуть використовуватись всі можливі шляхи в мережі. У другому класі задач (задача P) для передачі потоків задіяні тільки шляхи із наперед заданої множини шляхів. Математичні моделі представлені задачами лінійного, змішаного булевого лінійного та нелінійного програмування з блочною структурою матриці обмежень.

2. Розроблено декомпозиційні методи на основі *r*-алгоритму Шора. Наведено програми SolverA та SolverP для розв’язання опуклих задач знаходження оптимальних пропускних спроможностей дуг для побудови відмовостійкої орієнтованої мережі. Результати обчислювальних експериментів з цими програмами демонструють їхню конкурентоспроможність як порівняти з програмою IPOPT.

РОЗДІЛ 5. ОПТИМІЗАЦІЙНІ ЗАДАЧІ ДЛЯ МАКСИМАЛЬНОГО K-ПЛЕКСА

У п’ятому розділі описано дослідження двох оптимізаційних задач, призначених для знаходження максимального k-плекса у неорієнтованому графі. Перша задача є квадратичною оптимізаційною задачею, для неї описано застосування техніки лагранжевих двоїстих оцінок. Друга задача є задачею булевого лінійного програмування, для неї описано алгоритм знаходження усіх розв'язків.

5.1 Загальні відомості про k-плекс

Нехай $G=(V,E)$ – неорієнтований граф із множиною вершин $V=\{1,...,n\}$ та множиною ребер E. Ребро графа G, що зв'язує вершини $i\in V$ та $j\in V$, умовимося позначати $(i,j)\in E$. Для графа G будемо використовувати також іншу форму його представлення: $G=(V,\Gamma)$, де $\Gamma=\{\Gamma(i),\ i=\overline{1,n}\}$, а $\Gamma(i)$ – кінцеві вершини тих дуг, у яких початковою вершиною є вершина i. Кількість ребер графа G в обох представленнях зв'язані співвідношенням: $|E|=\frac{1}{2}\sum_{i\in V}|\Gamma(i)|$. Комплементарний до G граф будемо позначати $\overline{G}=(V,\overline{E})$, і $\overline{G}=(V,\overline{\Gamma})$, де $(i,j)\in\overline{E}$ і $\overline{\Gamma}=\{\overline{\Gamma}(i),\ i=\overline{1,n}\}$.

Означення 5.1. Підмножина вершин S із V називається k-плексом графа G, якщо ступінь кожної вершини в індукованому підграфі $G[S]$ (підграфі, породженому підмножиною S) є не менша, ніж $|S|-k$. Тобто $S\subset V$ є k-плексом, якщо виконується така умова:

$$\deg_{G[S]}(i)=|\Gamma(i)\cap S|\geq|S|-k\quad\forall i\in S.$$

k-Плекс є максимальним по включенню (maximal), якщо він не міститься ні в якому іншому k-плексі. Найбільший з максимальних по включенню k-плексів називається максимальним (maximum), його розмір називається k-плексним числом графа G та позначається $\rho_k(G)$ (Seidman and Foster 1978). Очевидно, що 1-плекс є клікою графа G, тому що ступінь кожної вершини в індукованому підграфі $G[S]$ не менша, ніж $|S|-1$, а це означає, що кожна з вершин у підграфі $G[S]$ зв'язана з усіма іншими вершинами, тобто підграф $G[S]$ є повним підграфом (клікою) графа G. У даному випадку $\rho_1(G)=\omega(G)$, де $\omega(G)$ – клікове число графа G, що відповідає розміру максимальної кліки в графі G.

Поняття co-k-плекса графа G, також введене в (Seidman and Foster 1978), є узагальненням поняття незалежної множини вершин графа G. При $k=1$ co-k-плекс збігається з незалежною множиною вершин графа. При $k>1$ co-k-плекс є ослабленням поняття незалежної множини вершин графа (відомої також як внутрішньо стійка множина).

Означення 5.2. Підмножина вершин S з V називається co-k-плексом графа G, якщо виконується така умова:

$$\deg_{G[S]}(i)=\left|\Gamma(i)\cap S\right|\le k-1 \quad \forall i\in S.$$

Отже, $S\subset V$ є co-k-плексом, якщо ступінь кожної вершини в індукованому підграфі $G[S]$ є не більшою, ніж $k-1$. Очевидно, що co-1-плекс є незалежною множиною вершин графа G, оскільки ступінь кожної вершини в індуційованому підграфі $G[S]$ дорівнює нулю, а це означає, що кожна з вершин у підграфі $G[S]$ не зв’язана з жодною з інших вершин підграфа $G[S]$. Відзначимо, що co-k-плекс і k-плекс для графа G знаходяться в такому ж зв’язку як кліка графа G і незалежна множина вершин графа G. Тому підмножина S є co-k-плексом графа G тоді й тільки тоді, коли S є k-плексом для комплементарного графа G.

5.2 Квадратичні обмеження для k-плекса

Нехай вершині $i \in V$ $(i = 1,2,...)$ відповідає булева змінна $x_i \in \{0,1\}$ така, що

$$x_i = \begin{cases} 1, & \textit{якщо } i \in S, \\ 0, & \textit{якщо } i \in V \setminus S. \end{cases}$$

Булеві змінні x_i, $i = \overline{1,n}$ будуть описуватися за допомогою квадратичних обмежень-рівностей

$$x_i^2 - x_i = 0 \quad \forall i \in V. \tag{5.1}$$

Побудуємо такі квадратичні обмеження, щоб підмножина S була k-плексом. Ці обмеження повинні задавати вимоги на те, щоб ступінь кожної вершини $i \in S$ у підграфі $G[S]$ була не меншою за $|S| - k$, інакше кажучи, щоб у підграфі $G[S]$ кількість дуг, що виходять із кожної вершини $i \in S$ була не меншою за $|S| - k$.

Нехай вершина i належить підмножині S, тобто $x_i = 1$. Позначимо $N_e(i)$ ступінь вершини i у підграфі $G[S]$, тобто кількість дуг, що виходять з вершини $i \in S$. Тоді в підграфі $G[S]$ ступені вершин із підмножини S задаються за допомогою сімейства співвідношень:

$$N_e(i) = \sum_{j \in \Gamma(i)} x_j, \quad \forall i \in S. \tag{5.2}$$

З рівняння $|S| = \sum_{j \in V} x_j$ та умови, що множина S є k-плексом, одержуємо нерівності

$$N_e(i) \geq |S| - k = \sum_{j \in \Gamma(i)} x_j - k, \quad \forall i \in S. \tag{5.3}$$

Із співвідношень (5.2) та (5.3) одержуємо сімейство нерівностей

$$\sum_{j \in \Gamma(i)} x_j \geq \sum_{j \in V} x_j - k, \quad \forall i \in S, \tag{5.4}$$

при виконанні яких всі ті вершини $i \in V$, для яких $x_i = 1$, будуть утворювати k-плекс.

Однак, нерівності типу (5.4) не будуть виконуватися для тих вершин $i \in V$, для яких $x_i = 0$, тобто для всіх $x_i \in V \setminus S$. Для того, щоб одержати квадратичні нерівності, що будуть справедливими і для змінних $x_i = 0$, досить обидві частини нерівності вигляду (5.4), що відповідає вершині i, помножити на змінну x_i. З огляду на невід'ємність змінної x_i знак нерівності після множення не зміниться, і в результаті одержимо такі нерівності:

$$x_i \left(\sum_{j \in \Gamma(i)} x_j \right) \geq x_i \left(\sum_{j \in V} x_j - k \right), \ \forall i \in V,$$

які можна переписати у вигляді

$$\sum_{j \in \Gamma(i)} x_i x_j \geq \sum_{j \in V} x_i x_j - k x_i, \ \forall i \in V. \qquad (5.5)$$

Квадратичні нерівності (5.5) разом із обмеженнями (5.1) повністю описують умови, за яких вершини i належать k-плексу. Дійсно, нерівності (5.5) будуть справедливі для тих вершин i, для яких $x_i = 1$, тому що вони переходять в обмеження (5.4). Нерівності (5.5) будуть справедливі також для усіх вершин i, для яких $x_i = 0$, тому що вони переходять у тривіальну нерівність $0 \geq 0$.

Зрозуміло, що за допомогою обмежень у вигляді рівностей (5.1) та у вигляді нерівностей (5.5) можна описати допустимі булеві розв'язки, що відповідають k-плексу. При цьому зміст обмежень (5.5) буде пов'язаний із інтерпретацією ступеня вершини, як цього вимагає поняття k-плекса, тобто ступінь вершини $i \in S$ більше або дорівнює $|S| - k$. Дійсно, права частина обмеження (5.5) для вершини i, що належить k-плексу, вказує кількість ребер, що виходять з i-ї вершини, з урахуванням того, що $x_i x_j = 1$ лише тоді, коли обидві змінні x_i та x_j дорівнюють одиниці.

Нерівність (5.5) можна спростити. Зауважимо, що

$$\sum_{j\in V} x_j = \sum_{j\in\Gamma(i)} x_j + \sum_{j\in\overline{\Gamma}(i)} x_j + x_i .$$

Тоді нерівності (5.5) можна переписати так:

$$\sum_{j\in\Gamma(i)} x_i x_j \geq \left(\sum_{j\in\Gamma(i)} x_i x_j + \sum_{j\in\overline{\Gamma}(i)} x_i x_j + x_i^2 \right) - kx_i, \ \ \forall i \in V ,$$

звідки

$$\sum_{j\in\overline{\Gamma}(i)} x_i x_j \leq kx_i - x_i^2, \ \ \forall i \in V .$$

З урахуванням того, що $x_i = x_i^2$ (див. формулу (5.1)), останні нерівності можна переписати як

$$\sum_{j\in\overline{\Gamma}(i)} x_i x_j \leq (k-1)x_i, \ \ \forall i \in V . \qquad (5.6)$$

Нерівності (5.6) разом із рівностями (5.1) ми покладемо в основу квадратичної моделі для знаходження максимального k-плекса графа G.

Зауважимо, що квадратичним нерівностям (5.6) можна надати інший зміст, ніж нерівностям (5.5). Нерівності (5.6) пов'язані з комплементарним графом $\overline{G}$ і описують таку підмножину вершин S, що ступінь вершини в індуційованому цією підмножиною підграфі $\overline{G}[S]$ була не більшою за $(k-1)$. Дійсно, для тих вершин $i \in V$, для яких $x_i = 1$, нерівності (5.6) рівносильні таким нерівностям

$$\sum_{j\in\overline{\Gamma}(i)} x_j \leq (k-1), \ \ \forall i \in S ,$$

а для тих вершин i, для яких $x_i = 0$, вони рівносильні тривіальним нерівностям $0 \leq 0$. Тому опис підмножини S за допомогою нерівностей (5.6) та рівностей (5.1) логічно інтерпретувати як опис co-k-плекса для комплементарного графа $\overline{G}$.

5.3 Квадратична булева задача для $\rho_k(G)$

Враховуючи, що обмеження (5.1) та (5.6) описують множину допустимих варіантів утворення k-плекса, для знаходження максимального k-плекса графа G оптимізаційну квадратичну задачу можна сформулювати в такій формі:

$$\rho_k(G) = \max_x \sum_{i \in V} x_i \tag{5.7}$$

за обмежень

$$\sum_{j \in \overline{\Gamma}(i)} x_i x_j \le (k-1) x_i, \ \ \forall i \in V, \tag{5.8}$$

$$x_i^2 - x_i = 0, \ \ \forall i \in V. \tag{5.9}$$

Зрозуміло, що задачу (5.7) – (5.9) можна інтерпретувати як задачу знаходження максимального $co\text{-}k$-плекса графа $\overline{G}$. Із (5.7) – (5.9) легко одержати формулювання квадратичної оптимізаційної задачі для $co\text{-}k$-плекса графа $\overline{G}$. Для цього досить у сумі лівої частини обмеження (5.8) замість підсумовування по $j \in \overline{\Gamma}(i)$ використовувати підсумовування по $j \in \Gamma(i)$. Інакше кажучи, якщо обмеження (5.8) замінити на

$$\sum_{j \in \Gamma(i)} x_i x_j \le (k-1) x_i, \ \ \forall i \in V, \tag{5.10}$$

то ми одержимо формулювання квадратичної оптимізаційної задачі знаходження максимального $co\text{-}k$-плекса графа $\overline{G}$.

Формулювання задачі (5.7) – (5.9) можна одержати із задачі булевого лінійного програмування, запропонованої в (Balansundaram, Butenko and Hicks 2011):

$$\rho_k(G) = \max_x \sum_{i \in V} x_i \tag{5.11}$$

за обмежень

$$\sum_{j \in \overline{\Gamma}(i)} x_j \le (k-1) x_i - \overline{d}_i (1 - x_i), \ \ \forall i \in V, \tag{5.12}$$

$$x_i \in \{0,1\},\ \ \forall i \in V, \tag{5.13}$$

де $\overline{d}_i = \left|\overline{\Gamma}(i)\right|$. Лінійні обмеження (5.12) побудовані за схемою, подібною тій, що використовувалася у попередньому розділі при переході від обмежень (5.4), справедливих для $i \in S$ (тобто коли $x_i = 1$), до обмежень (5.5), що є справедливими також для $i \in V \setminus S$ (тобто коли $x_i = 0$). Однак, правило для того, щоб обмеження (5.12) виконувалися для будь-яких $i \in V \setminus S$, тут буде іншим. Так, коли $x_i = 1$, тобто обмеження (5.12) виконуються як нерівності

$$\sum_{j \in \overline{\Gamma}(i)} x_j \le (k-1),\ \ \forall i \in S,$$

які повинні бути справедливими для $co\text{-}k$-плекса графа $\overline{G}$, що збігається з k-плексом графа G. Коли $x_i = 0$, то обмеження (5.12) переходять у нерівності

$$\sum_{j \in \overline{\Gamma}(i)} x_j \le \overline{d}_i = \left|\overline{\Gamma}(i)\right|,\quad \forall i \in V \setminus S,$$

які є справедливими, бо в якості верхньої границі на ступені вершин, що не входять у $co\text{-}k$-плекс графа $\overline{G}$, використовується максимальна можлива кількість ребер, що виходять з кожної з вершин графа $\overline{G}$. Ті з цих обмежень, де не всі змінні під знаком суми дорівнюють одиниці, будуть надлишковими.

Із задачі лінійного булевого програмування (5.11) – (5.13) легко одержати квадратичну задачу (5.7) – (5.9). Для цього слід обмеження (5.13) замінити на відповідний нелінійний аналог (5.9), а обмеження, яке відноситься до i-ї вершини з (5.12), помножити на змінну x_i. В силу невід'ємності змінних x_i, $i = 1,2,\ldots$, знаки нерівностей при множенні не зміняться, і в результаті одержимо

$$\sum_{j \in \overline{\Gamma}(i)} x_i x_j \le (k-1)x_i^2 + \overline{d}_i (1-x_i)x_i,\ \ \forall i \in V,$$

звідки, з урахуванням того, що $(1-x_i)x_i = x_i - x_i^2 = 0$ для всіх $i \in V$, приходимо до обмежень (5.8).

Зрозуміло, що кожна з задач (5.7) – (5.9) та (5.11) – (5.13) має свої переваги та свої недоліки. Так, наприклад, найсуттєвіша перевага квадратичної задачі

над лінійною булевою полягає в тому, що незначним удосконаленням задачі (5.7) – (5.9) можна сформулювати квадратичні оптимізаційні задачі знаходження максимальних за розміром підмножин графа із сильнішими властивостями, ніж k-плекс або co-k-плекс, – достатньо лише посилити вимогу на включення вершин у ці підмножини. Так, наприклад, умовимося під "строгим" k-плексом графа G розуміти підмножину його вершин, для яких ступінь вершини дорівнює $|S|-k$. Аналогічно введемо поняття строгого co-k-плекса: в ньому ступінь вершини дорівнює рівно $k-1$. Тоді для того, щоб сформулювати квадратичні задачі знаходження максимальних із цих підмножин, досить в обмеженнях (5.8) та (5.10) замість нерівностей використовувати рівності. Інакше кажучи, для знаходження "строгого" k-плекса графа G обмеження (5.8) слід замінити на обмеження

$$\sum_{j\in\overline{\Gamma}(i)} x_i x_j \le (k-1)x_i, \quad \forall i \in V,$$

а для знаходження строгого co-k-плекса графа G обмеження (5.10) слід замінити на такі:

$$\sum_{j\in\Gamma(i)} x_i x_j \le (k-1)x_i, \quad \forall i \in V.$$

До переваги лінійної булевої задачі над квадратичною можна віднести той факт, що для задачі (5.11) – (5.13) легко підраховувати верхні оцінки для $\rho_k(G)$ за допомогою релаксації обмеження (5.13). У ряді випадків, наприклад, коли $\rho_k(G)$ буде більше $n/3$, ці оцінки, як правило, можуть виявитися ефективними оцінками зверху для $\rho_k(G)$. У результаті для деяких спеціальних графів на базі методу гілок та границь можна реалізувати швидкі алгоритми для знаходження $\rho_k(G)$.

Знаходження верхніх оцінок для квадратичної задачі (5.7) – (5.9) є більш трудомістким, ніж для релаксованої задачі (5.11) – (5.13). Так, наприклад, якщо в такій якості використовувати лагранжеві двоїсті оцінки (Shor 1998), (Shor and Stetsyuk 2001), то знаходження таких оцінок за допомогою методів

недиференційованої оптимізації вимагатиме більше часу, ніж у випадку задач лінійного програмування. Однак, для ряду графів лагранжеві двоїсті оцінки можуть виявитися значно точнішими верхніми оцінками, ніж лінійні оцінки. Більш того, існує резерв для уточнення лагранжевих двоїстих оцінок задачі (5.7) – (5.9): це введення функціонально надлишкових обмежень (Shor 1998).

Лінійні обмеження (5.12) можна використовувати для побудови функціонально надлишкових обмежень з метою покращити точність лагранжевих двоїстих оцінок у багатоекстремальних квадратичних задачах (5.7) – (5.9). Якщо скористатися схемою, що використовувалася Н.З. Шором для задачі про максимальну незалежну множину вершин графа (Shor 1998, с. 250), то при цьому до задачі (5.7) – (5.9) додаються два види функціонально надлишкових обмежень. Обмеження першого виду одержано домноженням кожного з лінійних обмежень у (5.12) на ті змінні x_l, які не входять у це обмеження, тобто, додаються $n(n-1)$ надлишкових обмежень вигляду

$$\sum_{j\in\bar{\Gamma}(i)} x_i x_j \le (k-1)x_l x_i + \bar{d}_i(1-x_i)x_l,\ \ \forall i,l\in V,\ i\ne l. \qquad (5.14)$$

Функціонально надлишкові обмеження другого типу можна одержати з лінійних обмежень (5.12), помноживши на $1-x_l$, $l=1,2,\ldots$. Тут уже можна використовувати $i=l$, бо вони дають нові квадратичні обмеження у формі нерівностей. У результаті маємо n^2 обмежень

$$\sum_{j\in\bar{\Gamma}(i)} x_l x_j - \sum_{j\in\bar{\Gamma}(i)} x_j \le (k-1)x_l x_i + \bar{d}_i(1-x_i)x_l,\ \ \forall i,l\in V. \qquad (5.15)$$

Зауважимо, що використання надлишкових обмежень (5.14), (5.15) значно збільшує розміри квадратичної задачі. Однак, якщо до попередньої задачі додавати тільки невелику кількість тих надлишкових обмежень, які уточнюють двоїсту оцінку для наступної задачі, то тоді загальна кількість обмежень у кінцевій квадратичній задачі буде невеликою. За такою ж схемою можна використовувати інші види функціонально надлишкових обмежень для булевих задач, які розглядалися у роботах (Стецюк 2005), (Стецюк 2006).

5.4 Максимальний 1-плекс та максимальна кліка

Оскільки для графа G 1-плекс збігається з клікою, природно, що знаходження максимального 1-плекса графа G у задачі (5.7) – (5.9) відповідає знаходженню максимальної кліки графа G. Якщо $k=1$, то права частина нерівностей (5.8) дорівнює нулю, і задача (5.7) – (5.9) переходить у таку квадратичну оптимізаційну задачу:

$$\rho_1(G)=\omega(G)=\max\sum_{i\in V}x_i \tag{5.16}$$

за обмежень

$$\sum_{j\in\overline{\Gamma}(i)}x_ix_j\le 0,\ \ \forall i\in V, \tag{5.17}$$

$$x_i^2-x_i=0,\ \ \forall i\in V. \tag{5.18}$$

Задачу (5.16) – (5.18) можна розглядати як одне з можливих квадратичних формулювань задачі про знаходження максимальної кліки графа G. Задачі (5.16) – (5.18) відповідає рівно n квадратичних обмежень нерівностей (5.17).

Більш відомим квадратичним формулюванням задачі для максимальної кліки графа G є квадратична оптимізаційна задача (Balansundaram, Butenko and Hicks 2011):

$$\omega(G)=\max\sum_{i\in V}x_i \tag{5.19}$$

за обмежень

$$x_ix_j=0,\ \ \forall(i,j)\in\overline{E}, \tag{5.20}$$

$$x_i^2-x_i=0,\ \ \forall i\in V. \tag{5.21}$$

Задачу (5.19) – (5.21) можна інтерпретувати як очевидний наслідок задачі (5.16) – (5.18), тому що всі змінні x_i, $i=\overline{1,n}$ є невід'ємними. Дійсно, згідно з квадратичними обмеженнями (5.17), сума невід'ємних добутків пар змінних повинна бути не більше нуля, а це еквівалентно (за умови невід'ємності) системі квадратичних рівностей, де кожен окремий добуток двох змінних, котрі

входять в обмеження (5.17), дорівнює нулю. Це й становить зміст обмежень (5.20).

Яка з квадратичних моделей є кращою для знаходження максимальної кліки графа G? Формально задача (5.16) – (5.18) має ту перевагу, що вона містить меншу кількість квадратичних обмежень, ніж задача (5.19) – (5.21). Однак, вона програє в точності лагранжевої двоїстої оцінки.

Нехай $\psi^*_{\rho_1}(G)$ – оптимальна лагранжева оцінка для задачі (5.16) – (5.18), а $\psi^*_{\omega}(G)$ – оптимальна лагранжева оцінка для задачі (5.19) – (5.21). Кожна з них буде верхньою оцінкою для $\omega(G)$. Верхня оцінка $\psi^*_{\rho_1}(G)$ буде, як правило, гіршою за оцінку $\psi^*_{\omega}(G)$. Це можна пояснити тим, що в задачі (5.19) – (5.21) ми маємо більшу свободу у виборі множників Лагранжа, що відповідають обмеженням (5.20): кожному з ребер комплементарного графа $\overline{G}$ буде відповідати свій множник Лагранжа. Для обмежень (5.17) таких множників Лагранжа усього n, тобто свій множник Лагранжа буде відповідати кожній з вершин графа G. Крім того, свобода на вибір множників Лагранжа для обмежень (5.17) обмежена ще й тим, що для кожного з цих множників потрібно врахувати його невід'ємність. Урахування невід'ємності обмежує можливості вибору множників Лагранжа в задачі (5.16) – (5.18) у порівнянні з випадком, коли обмеження (5.17) замінюються на

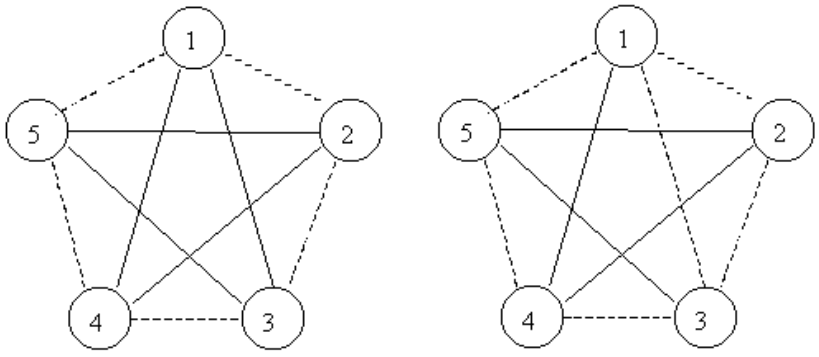
$$\sum_{j \in \overline{\Gamma}(i)} x_i x_j = 0, \ \ \forall i \in V$$

(вони випливають із обмежень (5.20), якщо їх згрупувати по вершинах графа G). Зауважимо, що квадратичну задачу з цими обмеженнями замість обмежень (5.17) можна розглядати як задачу знаходження максимального «строгого» 1-плекса графа G. Природно, що лагранжева двоїста оцінка для квадратичної задачі, що відповідає максимальному «строгому» 1-плексу графа G, буде в багатьох випадках точнішою за оцінку $\psi^*_{\rho_1}(G)$.

Проілюструємо поводження лагранжевих двоїстих оцінок $\psi^*_{\rho_1}(G)$ та $\psi^*_{\omega}(G)$ на прикладі двох графів G_1 і G_2 (рис. 5.1), що складаються усього з п'яти вершин. Тут ребра графів G_1 і G_2 позначено суцільною лінією, а ребра комплементарних графів $\overline{G}_1$ і $\overline{G}_2$ – пунктирною. Для графа G_1 (рис. 5.1а) системи обмежень для задач (5.16) – (5.18) та (5.19) – (5.21) будуть такими:

$$x_1x_2 + x_1x_5 \le 0, \quad x_1x_2 = 0,$$
$$x_1x_2 + x_2x_3 \le 0, \quad x_2x_3 = 0,$$
$$x_2x_3 + x_3x_4 \le 0, \quad x_3x_4 = 0,$$
$$x_3x_4 + x_4x_5 \le 0, \quad x_4x_5 = 0,$$
$$x_4x_5 + x_1x_5 \le 0, \quad x_1x_5 = 0.$$

Лагранжеві двоїсті оцінки $\psi^*_{\rho_1}(G)$ та $\psi^*_{\omega}(G)$ є однаковими й дорівнюють $\sqrt{5}$. Вони є неточними оцінками зверху, тому що $\omega(G_1) = 2$. Приклад для графа G_1 демонструє, що лагранжеві двоїсті оцінки $\psi^*_{\rho_1}(G)$ та $\psi^*_{\omega}(G)$ можуть співпадати.



а) б)

Рис. 5.1. Приклади графів G_1 та G_2.

Це твердження не виконується для графа G_2 на рис. 5.1 б), що відрізняється від G_1 тим, що з нього вилучене ребро (1,3). Для графа G_2 системи обмежень задач (5.16) – (5.18) та (5.19) – (5.21) будуть такими:

$$
\begin{aligned}
&x_1x_2 + x_1x_3 + x_1x_5 \le 0, \quad && x_1x_2 = 0,\\
&x_1x_2 + x_2x_3 \le 0, && x_1x_3 = 0,\\
&x_1x_3 + x_2x_3 + x_3x_4 \le 0, && x_2x_3 = 0,\\
&x_3x_4 + x_4x_5 \le 0, && x_3x_4 = 0,\\
&x_4x_5 + x_1x_5 \le 0 && x_4x_5 = 0,\\
& && x_1x_5 = 0.
\end{aligned}
$$

На цей раз оцінки $\psi^*_{\rho_1}(G)$ та $\psi^*_{\omega}(G)$ – різні. Зокрема, оцінка $\psi^*_{\omega}(G) = 2.0$ є точною верхньою оцінкою для $\omega(G_2)$. Оцінка $\psi^*_{\rho_1}(G) = 2.15222$ не є точною і потребує уточнення. Отже, приклад графа G_2 показує, що лагранжева двоїста оцінка для задачі (5.19) – (5.21) має перевагу над відповідною оцінкою задачі (5.16) – (5.18).

Якщо для графа G_2 обмеження нерівності замінити на обмеження у формі рівностей, що відповідає знаходженню максимального «строгого» 1-плекса графа G_1, то лагранжева двоїста оцінка стане рівною двом і буде точною верхньою оцінкою для $\omega(G_1)$.

Якщо потрібно знайти більш точну лагранжеву двоїсту оцінку для максимальної кліки графа G за допомогою одної з двох розглянутих квадратичних задач, то краще користуватися квадратичною задачею у формі (5.19) – (5.21), ніж квадратичною задачею у формі (5.16) – (5.18), що для 1-плекса випливає з квадратичного формулювання (5.7) – (5.9). Аналогічна ситуація буде мати місце і для квадратичної задачі (Shor 1998), що пов'язана із знаходженням максимальної незалежної множини вершин графа.

Однак, ситуація може змінитися, якщо мова йтиме не про знаходження оцінок $\psi^*_{\rho_1}(G)$ та $\psi^*_{\omega}(G)$, а про їх поліпшення за допомогою додавання функціонально надлишкових обмежень. Тоді квадратична задача (5.16) – (5.18) може мати переваги хоча б тому, що до неї можна додати більшу кількість функціонально надлишкових обмежень. Крім того, коли $k \ge 2$, то для

знаходження максимального k-плекса навряд чи можна знайти альтернативу квадратичному формулюванню (5.7) – (5.9). Тому дослідження точності лагранжевих двоїстих оцінок для задачі (5.7) – (5.9), що підсилена за рахунок додавання функціонально надлишкових квадратичних обмежень, становить значний інтерес. У результаті може виявитися, що безпосереднє перенесення таких оцінок на окремий випадок $k=1$ для ряду графів може привести до досить хороших по точності верхніх оцінок для $\omega(G)$.

Для уточнення верхніх оцінок можна використовувати квадратичні нерівності з двох сімейств функціонально надлишкових квадратичних обмежень – це сімейства квадратичних обмежень (5.14) та (5.15). Для графів G_1 та G_2 з рис. 5.1 нижче буде показано, що за допомогою функціонально надлишкових обмежень із сімейств (5.14) і (5.15) лагранжеві двоїсті оцінки можна зробити точними верхніми оцінками для $\rho_2(G_1)$ та $\rho_2(G_2)$.

5.5 Уточнення лагранжевих оцінок для максимального 2-плекса

Розглянемо задачу знаходження максимального 2-плекса для графа G_1 з рис. 5.1а). Вона формулюється у вигляді такої квадратичної задачі:

$$\rho_2(G_1)=\max_{x\in\mathbb{R}^5}(x_1+x_2+x_3+x_4+x_5) \tag{5.22}$$

$$\text{за обмежень}\begin{cases} x_1x_2+x_1x_5\le x_1,\\ x_1x_2+x_2x_3\le x_2,\\ x_2x_3+x_3x_4\le x_3,\\ x_3x_4+x_4x_5\le x_4,\\ x_4x_5+x_1x_5\le x_5, \end{cases} \tag{5.23}$$

$$x_i^2-x_i=0,\quad i=\overline{1,5}. \tag{5.24}$$

Тут $\rho_2(G_1)=3$. У той же час лагранжева двоїста оцінка для задачі (5.22) – (5.24) дорівнює 3.618034. Вона є менш точною, ніж лінійна верхня оцінка, що виходить у результаті релаксації задачі булевого програмування

(5.11) – (5.13) і дорівнює 10/3 = 3.33333. Задачі лінійного програмування відповідають такі лінійні обмеження:

$$\begin{cases} x_1 + x_2 + x_5 \le 2, \\ x_1 + x_2 + x_3 \le 2, \\ x_2 + x_3 + x_4 \le 2, \\ x_3 + x_4 + x_5 \le 2, \\ x_1 + x_4 + x_5 \le 2, \end{cases} \tag{5.25}$$

які випливають з обмежень (5.12).

Як лагранжева двоїста оцінка, так і лінійна оцінка є неточними верхніми оцінками. Покращити лагранжеву двоїсту оцінку, що відповідає задачі (5.22) – (5.24), можна, додаючи до квадратичної задачі (5.22) – (5.24) функціонально надлишкові обмеження із сімейства (5.14). Вони побудовані домноженням лінійних обмежень (5.25) на змінні x_i, $i = \overline{1,n}$. Послідовне додавання таких обмежень робить лагранжеву двоїсту оцінку більш точною. При послідовному додаванні функціонально надлишкових квадратичних обмежень із сімейства (5.14), вказаних у табл. 5.1, спостерігається монотонне зменшення оцінки ψ^*. У результаті додавання чотирьох додаткових обмежень верхня оцінка ψ^* стає точнішою за лінійну. А в результаті додавання всіх п'яти обмежень оцінка ψ^* стає точною верхньою оцінкою для $\rho_2(G_1)$.

Таблиця 5.1

Покращення верхньої оцінки для задачі (5.22) – (5.24)

$+n$	Лінійні обмеження	$\times x_i$	Квадратичні обмеження	ψ^*
+1	$x_1 + x_2 + x_5 \le 2$	$\times x_2$	$x_1x_2 + x_2x_5 \le x_2$	3.56878
+2	$x_1 + x_2 + x_3 \le 2$	$\times x_3$	$x_1x_3 + x_2x_3 \le x_3$	3.52727
+3	$x_2 + x_3 + x_4 \le 2$	$\times x_4$	$x_2x_4 + x_2x_4 \le x_4$	3.4292

				9
+4	$x_3 + x_4 + x_5 \le 2 \times x_5$		$x_3x_5 + x_4x_5 \le x_5$	3.21298
+5	$x_1 + x_4 + x_5 \le 2 \times x_1$		$x_1x_4 + x_1x_5 \le x_1$	3.00000

Схожа, але дещо інша, ситуація має місце і для задачі знаходження максимального 2-плекса для графа G_2 з рис. 5.1б). Вона формулюється у вигляді квадратичної задачі

$$\rho_2(G_2) = \max_{x \in \mathbb{R}^5} (x_1 + x_2 + x_3 + x_4 + x_5) \tag{5.26}$$

за обмежень

$$\begin{cases} x_1x_2 + x_1x_3 + x_1x_5 \le x_1, \\ x_1x_2 + x_2x_3 \le x_2, \\ x_1x_2 + x_2x_3 + x_3x_4 \le x_3, \\ x_3x_4 + x_4x_5 \le x_4, \\ x_4x_5 + x_1x_5 \le x_5, \end{cases} \tag{5.27}$$

$$x_i^2 - x_i = 0, \quad i = \overline{1,5}, \tag{5.28}$$

і їй відповідає $\rho_2(G_2) = 3$. Лагранжева двоїста оцінка для задачі (5.26) – (5.28) дорівнює 3.37332 і є точнішою, ніж лінійна верхня оцінка, що дорівнює 3.4. Задачі лінійного програмування відповідають такі лінійні обмеження виду (5.12):

$$\begin{cases} 2x_1 + x_2 + x_3 + x_5 \le 3, \\ x_1 + x_2 + x_3 \le 2, \\ x_1 + x_2 + 2x_3 + x_4 \le 3, \\ x_3 + x_4 + x_5 \le 2, \\ x_1 + x_4 + x_5 \le 2. \end{cases} \tag{5.29}$$

Враховуючи, що лагранжева двоїста оцінка є точнішою за лінійну, може скластися враження, що точну оцінку ψ^* простіше одержати за допомогою додавання функціонально надлишкових обмежень (5.14), тобто тих, котрі

побудовані з лінійних обмежень (5.29) домноженням на змінні x_i, $i=\overline{1,5}$. Однак, це не так, і максимальна по точності лагранжева двоїста оцінка, яку можна досягти на цьому шляху, дорівнює 3.02316. Лише при додаванні функціонально надлишкових обмежень у вигляді (5.15) можна домогтися того, щоб лагранжева двоїста оцінка стала точною. Наприклад, після додавання до квадратичної задачі (5.26) – (5.28) функціонально надлишкового обмеження виду (5.14)

$$x_1x_1 + x_1x_4 + x_1x_5 - 2x_1 \le 0,$$

яке одержано із п'ятого обмеження (5.29) домноженням на x_1, оцінка ψ^* стає рівною 3.21576. Після додавання обмеження виду (5.15)

$$-x_1x_3 - x_1x_4 - x_1x_5 + 2x_1 + x_3 + x_4 + x_5 \le 2,$$

яке отримано домноженням четвертого обмеження з (5.29) на $(1-x_1)$, оцінка ψ^* стає рівною 3.01571. А після додавання ще й обмежень виду (5.15)

$$-2x_1x_2 - x_2^2 - x_2x_3 - x_2x_5 + 2x_1 + 4x_2 + x_3 + x_5 \le 3,$$

які одержано домноженням першого обмеження з (5.29) на $(1-x_2)$, оцінка ψ^* стає точною й дорівнює 3.0.

5.6 Застосування GLPK для знаходження всіх розв'язків

Задача (5.11) – (5.13) може мати як один, так і багато розв'язків. Для того, щоб знайти усі її розв'язки, можна використати алгоритм (Стецюк, Слабоспицька, та Ушакова 2016), який полягає у послідовному доповненні задачі додатковим лінійним обмеженням, яке відсікає уже знайдені розв'язки. Нехай уже знайдено m максимальних k-плексів S_j, $j=\overline{1,m}$. Додамо до задачі (5.11) – (5.13) таке сімейство лінійних обмежень:

$$\sum_{i\in V(S_j)} x_i \le \rho_k(G) - 1/2, \quad j=\overline{1,m}. \tag{5.30}$$

Якщо в задачі (5.11) – (5.13), (5.30) значення цільової функції дорівнює значенню цільової функції у задачі (5.11) – (5.13), то це означає, що ми знайшли новий максимальний k-плекс, який не співпадає ні з одним із m існуючих. Якщо в задачі (5.11) – (5.13), (5.30) значення цільової функції є меншим за значення цільової функції у задачі (5.11) – (5.13), то це означає, що ми отримали достатню умову того, що інших максимальних k-плексів не існує, окрім тих m, які були знайдені раніше.

Описаний алгоритм для пошуку $n_{sol} < ncalls$ розв`язків у задачі (5.11) – (5.13) реалізований мовою Octave. Задачі лінійного булевого програмування (5.11) – (5.13) та (5.11) – (5.13), (5.30) розв’язуються за допомогою octave-функції GLPK (Guo et al. 2010), яка використовує відомий пакет GLPK (GNU Linear Programming Kit) для розв'язання задач лінійного програмування та змішаного цілочислового програмування. Зауважимо, що за кожний послідовний запуск функції GLPK алгоритм або знаходить ще один максимальний k-плекс, або закінчує свою роботу, якщо всі n_{sol} k-плексів уже знайдено. У випадку, якщо кількість розв’язків у задачі (5.11) – (5.13) є більшою за ncalls, то в результаті роботи алгоритму отримаємо ncalls максимальних k-плексів. Для того, щоб продовжити пошук нових k-плексів, потрібно збільшити величину ncalls.

Алгоритм перевірено на ряді тестових прикладів. Дані для графа читаються з текстового файла у DIMACS-форматі. Нижче наведемо результати обчислювальних експериментів для пошуку всіх максимальних k-плексів для графа «1zc.128» (128 вершин, 2240 ребер) при різних значеннях $k = \overline{1,8}$ (табл. 5.1) та для пошуку всіх максимальних клік у чотирьох графах типу «1zc» з 128, 256, 512, 1024 вершинами (табл. 5.2). Кількості вершин та ребер для графів «1zc.256», «1zc.512» та «1zc.1024» наведені в першій колонці табл. 5.2. Максимальний за розмірами граф «1zc.1024» містить 1024 вершин та 33280 ребер. Обчислювальні експерименти здійснювались на комп’ютері з процесором Intel Core i5-9400f: частота 2.9 Ггц та оперативна пам’ять 16 Гб.

В табл. 5.2 наведено кількості знайдених розв'язків задачі (5.11) – (5.13) (колонка n_{col}); найменші (колонка $t_{\min}$) та найбільші (колонка $t_{\max}$) значення часу, який було затрачено на розв'язання однієї із задач (5.11) – (5.13), (5.11) – (5.13), (5.35); середній час (колонка t_{avr}) визначався як сумарний час, затрачений програмою GLPK на розв'язання задач (5.11) – (5.13), (5.11) – (5.13), (5.35), розділений на величину ($n_{sol}+1$).

Таблиця 5.2

Затрати часу (в секундах) на пошук n_{sol} максимальних k-плексів у графі «1zc.128»: $k=\overline{1,8}$, $ncalls=1001$

k	n_{sol}	$t_{\min}$	$t_{\max}$	t_{avr}		k	n_{sol}	$t_{\min}$	$t_{\max}$	t_{avr}
1	2	1.09	1.11	1.10		5	630	5.93	49.04	14.25
2	2	2.61	2.71	2.66		6	56	4.68	17.80	7.85
3	2	10.32	11.39	10.84		7	196	2.74	9.73	4.66
4	728	2.92	16.08	7.14		8	>1000	19.99	218.16	58.61

Таблиця 5.3

Затрати часу (в секундах) на пошук максимальних клік у графах типу «1zc» з 128, 256, 512, 1024 вершинами: $ncalls=3$

$n,\|E\|$	Знайдені максимальні кліки	t_{avr}
128, 2240	$\{64,96,112,120,124,126,127,128\}$ $\{1,2,3,5,9,17,33,65\}$	1.10
256, 5632	$\{1,2,3,5,9,17,33,65,129\}$ $\{128,192,224,240,248,252,254,255,256\}$	13.43
512, 13824	{1, 2, 3, 5, 9, 17, 33, 65, 129, 257} {256, 384, 448, 480, 496, 504, 508, 510, 511, 512}	234.92
1024, 33280	{1, 2, 3, 5, 9, 17, 33, 65, 129, 257, 513} {512, 768, 896, 960, 992, 1008, 1016, 1020, 1022, 1023, 1024}	3808.30

В табл. 5.3 наведено по дві знайдені максимальні кліки для чотирьох графів типу «1zc» та доведено, що інших максимальних клік немає, так як кількість знайдених клік менша за $ncalls = 3$.

5.7 Висновки до п’ятого розділу

1. Побудовано квадратичне формулювання оптимізаційної задачі для знаходження максимального k-плекса у неорієнтованому графі. Показано, що її можна отримати із відомої лінійної моделі у формі задачі лінійного булевого програмування, домножуючи обмеження на невід’ємні змінні для кожної із вершин графа.
2. Розроблено алгоритм пошуку всіх максимальних k-плексів для неорієнтованого графа. В основі алгоритму лежить послідовне додавання до задачі лінійного булевого програмування додаткового обмеження, яке відсікає вже знайдені максимальні k-плекси. Алгоритм був реалізований на мові Octave за допомогою GLPK (GNU Linear Programming Kit). Наведено результати обчислень для пошуку всіх максимальних k-плексів при $k = \overline{1,8}$ для графа «1zc.128» та всіх максимальних клік для графів типу «lzc» з 128, 256, 512, 1024 вершинами.

РОЗДІЛ 6. ЛІНІЙНА ТА КВАДРАТИЧНА ДВОЕТАПНІ ТРАНСПОРТНІ ЗАДАЧІ

Шостий розділ присвячено трьом математичним моделям для двоетапної транспортної задачі: задачі лінійного програмування та задачі квадратичного програмування. Наведено універсальний спосіб їх розв'язання за допомогою сучасного програмного забезпечення. У підрозділі 6.1 описано формулювання двоетапної транспортної задачі та наведено її властивості. У підрозділі 6.2 наведено опис задачі на мові моделювання AMPL та результати розрахунків для тестового прикладу. У підрозділі 6.3 описано модифікацію двоетапної транспортної задачі за умови, що кількість проміжних пунктів є обмеженою. Підрозділ 6.4 присвячено реалізації модифікації мовою AMPL та результатам обчислювальних експериментів. В підрозділі 6.5 описано квадратичну двоетапну транспортну задачу та доведено єдиність її розв'язку.

6.1 Формулювання задачі та її властивості

При формулюванні класичної двоетапної транспортної задачі (Стецюк так ін. 2019), (Карагодова, Кігель, та Рожок 2007), (Наконечний та Савіна 2003) мається на увазі, що вантаж перевозиться від постачальників до споживачів тільки через проміжні пункти. Схема функціонування перевезень вантажу наведено на рисунку 6.1. В якості проміжних пунктів можуть виступати посередницькі фірми та різного роду сховища (склади).

Нехай в m пунктах постачання $A_1, ..., A_m$ є $a_1, ..., a_m$ одиниць продукції, яку потрібно перевезти до n споживачів $B_1, ..., B_n$, задовольнивши їх потреби $b_1, ..., b_n$. Для транспортування продукції від постачальників до споживачів можна задіяти l проміжних пунктів $D_1, ..., D_l$. Витрати на перевезення одиниці продукції з пункту постачання A_i до проміжного пункту D_k позначимо c_{ik}, з

проміжного пункту D_k до споживача B_j – c_{kj}. Кількість продукції, яка перевозиться від постачальника A_i до проміжного пункту D_k, позначимо x_{ik}, кількість продукції від проміжного пункту D_k до споживача B_j позначимо y_{kj}.

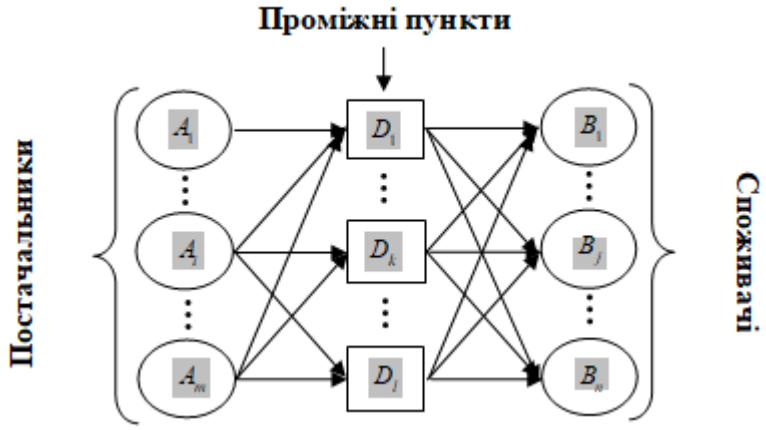


Рис. 6.1. Система «постачальники – проміжні пункти – споживачі» у двоетапній транспортній задачі.

Двоетапна транспортна задача має такий вигляд: знайти

$$f^* = \min_{x,y}\left\{ f(x,y) = \sum_{i=1}^{m}\sum_{k=1}^{l} c_{ik}x_{ik} + \sum_{k=1}^{l}\sum_{j=1}^{n} c_{kj}y_{kj} \right\} \tag{6.1}$$

за обмежень

$$\sum_{k=1}^{l} x_{ik} = a_i, \quad i = 1,\ldots,m, \tag{6.2}$$

$$\sum_{k=1}^{l} y_{kj} = b_j, \quad j = 1,\ldots,n, \tag{6.3}$$

$$\sum_{i=1}^{m} x_{ik} - \sum_{j=1}^{n} y_{kj} = 0, \quad k = 1,\ldots,l, \tag{6.4}$$

$$x_{ik} \geq 0,\ y_{kj} \geq 0, \quad i = 1,\ldots,m,\ k = 1,\ldots,l,\ j = 1,\ldots,n. \tag{6.5}$$

Задача (6.1) – (6.5) – задача лінійного програмування, яка містить $m \times l + l \times n$ змінних x_{ik}, y_{kj} та $m + n + l$ обмежень загального виду, не враховуючи обмежень (6.5) – умов на невід'ємність усіх змінних. Цільова

функція (6.1) задає сумарні витрати на транспортування продукції від постачальників до споживачів через проміжні пункти. Обмеження (6.2) означають транспортування усієї продукції $a_1, ..., a_m$ із пунктів постачання до проміжних пунктів, а обмеження (6.3) – що споживачам потрібно доставити необхідну продукцію $b_1, ..., b_n$ з проміжних пунктів. Обмеження (6.4) задають умови на те, щоб вся продукція яка приходить від постачальників до кожного проміжного пункту, була обовязково відправлена споживачам. Це визначає умови на сумісність системи обмежень (6.2) – (6.5) – системи лінійних рівностей та лінійних нерівностей. Звідси випливає наступне твердження.

Лема 6.1 (Стецюк, Ляшко, та Мазютинець 2018). Система обмежень (6.2) – (6.5) несумісна, якщо

$$\sum_{i=1}^{m} a_i \neq \sum_{j=1}^{n} b_j .$$

Лема 6.1 дає можливість перевірити вхідні дані на предмет їх коректності для сформульованої двоетапної транспортної задачі. Якщо не виконується умова леми, то тоді виконується умова $\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$. У цьому випадку система (6.2) – (6.5) – сумісна і можна переходити до розв'язання задачі (6.1) – (6.5).

6.2 AMPL-реалізація задачі та тестовий приклад

AMPL-код для опису двоетапної транспортної задачі (6.1) – (6.5) має такий вигляд:

```
param m>=2; #Кількість постачальників (пункти A)
param l>=1; #Кількість проміжних пунктів (пункти D)
param n>=2; #Кількість споживачів (пункти B)
#Вартості перевезення одиниці продукції:
param cik{i in 1..m, k in 1..l} >= 0; #від A до D
param ckj{k in 1..l, j in 1..n} >= 0; #від D до B
#Інші дані
param a{i in 1..m} >= 0; #Продукція в A
```

```
param b{j in 1..n} >= 0; #Потреби в B
check: sum{i in 1..m} a[i] = sum{j in 1..n} b[j]; #умова леми 1
#Невідомі (продукція, яку потрібно перевезти)
var x{i in 1..m, k in 1..l}  >=0; #від A до D
var y{k in 1..l, j in 1..n}  >=0; #від D до B
minimize f_opt: #Мінімізувати витрати на перевезення продукції:
sum{i in 1..m, k in 1..l} cik[i,k]*x[i,k]+ #від A до D
sum{k in 1..l, j in 1..n} ckj[k,j]*y[k,j]; #від D до B
subject to #за обмежень
con2 {i in 1..m}: #перевезення продукції з A до D
sum{k in 1..l}x[i,k] = a[i];
con3 {j in 1..n}: #задоволення потреб B з D
sum{k in 1..l}y[k,j] = b[j];
con4 {k in 1..l}: #всю продукцію потрібно доставити в B
sum{i in 1..m}x[i,k]-sum{j in 1..n}y[k,j]=0;
```

Тут оператор **param** використано для опису розмірів та даних задачі; оператор **var** – для опису змінних задачі, де враховано їх невід'ємність; оператор **check** – для перевірки сумісності обмежень (6.2) – (6.5), яка визначається лемою 1.1 та вимагає, щоб уся продукція постачальників була відправлена споживачам.

Якщо цей AMPL-код доповнити необхідними даними для конкретної задачі, то його можна використовувати для розв'язання двоетапних транспортних задач за допомогою стандартного програмного забезпечення для розв'язання задач лінійного програмування. Це можна зробити як за допомогою тих програм NEOS-сервера (NEOS Server 2023) із розділу «Linear Programming», для яких підтримуються вхідні формати даних на AMPL, так і за допомогою комерційних або з вільним доступом версій AMPL.

Тестовий приклад (Наконечний та Савіна 2003). Виробниче об'єднання складається з трьох філіалів: A_1, A_2, A_3, які виготовляють однорідну продукцію в обсягах відповідно 1000, 1500 та 1200 одиниць на місяць. Ця продукція відправляється на два склади D_1 і D_2, а потім – до п'яти споживачів B_1, B_2, …, B_5, попит яких становить відповідно 900, 700, 1000, 500 і 600 одиниць. Вартості перевезень одиниці продукції (в умовних одиницях) від виробників на склади, а потім – зі складів до споживачів наведені в табл. 6.1 і 6.2.

Таблиця 6.1

Вартість перевезень одиниці продукції (в умовних одиницях)

від виробників на склади

$A \setminus D$	D_1	D_2
A_1	2	8
A_2	3	5
A_3	1	4

Таблиця 6.2

Вартість перевезень одиниці продукції (в умовних одиницях)

зі складів до споживачів

$D \setminus B$	B_1	B_2	B_3	B_4	B_5
D_1	1	3	8	5	4
D_2	2	4	5	3	1

Для цього прикладу опис вхідних даних задачі (6.1) – (6.5) реалізує такий AMPL-код:

```
data; #Блок вхідних даних, де задаємо:
param m := 3; #Кількість постачальників A
param l := 2; #Кількість проміжних пунктів D
param n := 5; #Кількість споживачів B
#Витрати на перевезення одиниці продукції:
param cik: 1 2 := #від A (↓) до D (→)
          1 2 8
          2 3 5
          3 1 4;
param ckj: 1 2 3 4 5 := #від D (↓) до B (→)
          1 1 3 8 5 4
          2 2 4 5 3 1;
param a:= #Продукція в A
1 1000 2 1500 3 1200; #A_1, A_2, A_3
param b:= #Потреби B
1 900 2 700 3 1000 #B_1, B_2, B_3
```

```
4 500 5 600; #B_4, B_5
solve; #Розв'язати задачу (1)-(5)
#Роздрукувати цільову функцію та час розв'язання
display f_opt, _solve_time;
#Роздрукувати значення оптимальних змінних
display x; display y;
```

Результат розрахунку для тестового прикладу за допомогою відомої програми Gurobi (Gurobi 2023) на NEOS-сервері має такий вигляд:

```
NEOS Server Version 6.0
   Solver   : lp:Gurobi:AMPL
   Start    : 2020-12-19 07:51:02
   End      : 2020-12-19 07:51:06
   Host     : prod-sub-1.neos-server.org

************************************************************
You are using the solver gurobi_ampl.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Checking ampl.mod for gurobi_options...
Executing AMPL.
processing data.
processing commands.
Executing on prod-exec-3.neos-server.org

16 variables, all linear
10 constraints, all linear; 32 nonzeros
     10 equality constraints
1 linear objective; 16 nonzeros.

Gurobi 9.0.1: threads=4
Gurobi 9.0.1: optimal solution; objective 22100
1 simplex iterations
f_opt = 22100
_solve_time = 0.004989

x :=
1 1   1000
1 2      0
2 1      0
2 2   1500
3 1   1200
3 2      0
```

```
;

y :=
1 1   900
1 2   700
1 3   100
1 4   500
1 5     0
2 1     0
2 2     0
2 3   900
2 4     0
2 5   600
;
```

Для тестового прикладу задача (6.1) – (6.5) має багато розв'язків. Один із них показано на рис. 6.2. Для нього значення цільової функції дорівнює 22100 умовних одиниць, склад D_1 завантажений на 2200 одиниць, а склад D_2 – на 1500 одиниць.

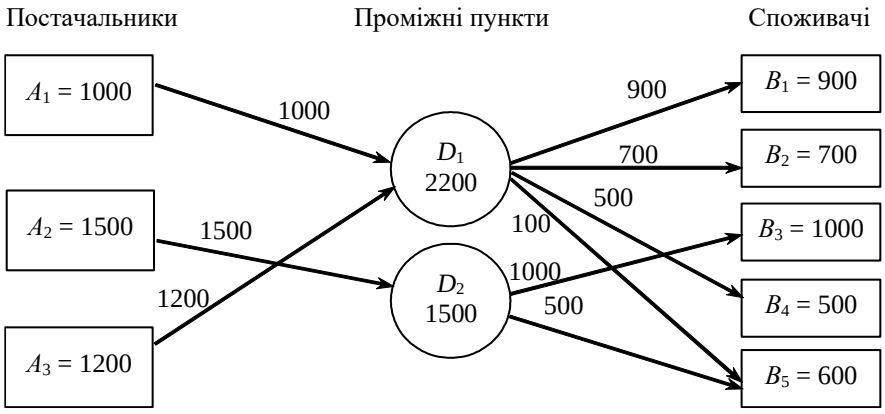


Рис. 6.2. Перший оптимальний план транспортування продукції.

Другий розв’язок показано на рис. 6.3 та характеризується більш рівномірною завантаженістю складів. Для нього значення цільової функції дорівнює 22100 умовних одиниць, склад D_1 завантажений на 2100 одиниць, а склад D_2 – на 1600 одиниць.

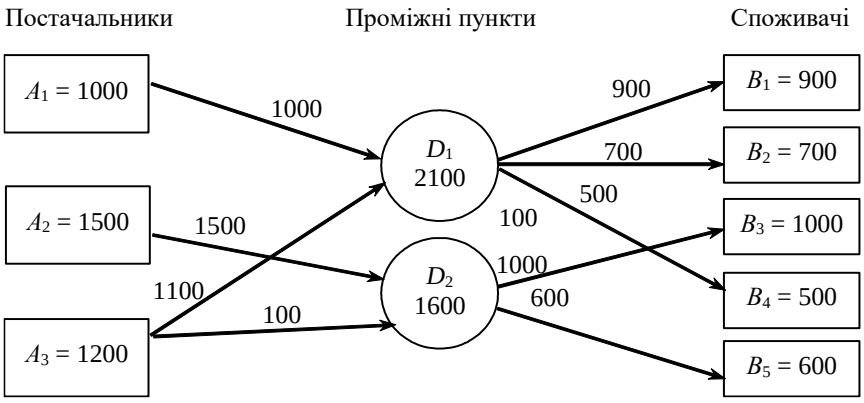


Рис. 6.3. Другий оптимальний план транспортування продукції

Неоднозначності оптимального розв'язку легко уникнути, якщо від задачі лінійного програмування (6.1) – (6.5) перейти до задачі квадратичного програмування, де цільова функція буде строго опуклою квадратичною функцією. Таку задачу розглянемо далі.

6.3 Двоетапна транспортна задача з обмеженням на кількість проміжних пунктів

Розглянемо модифікацію задачі (6.1) – (6.5) за умови, що для перевезення продукції з m пунктів постачання $A_1,...,A_m$ до n споживачів $B_1,...,B_n$ можна задіяти не більше ніж d $(1 \le d \le l)$ з l проміжних пунктів $D_1,...,D_l$, пропускні спроможності яких будемо вважати необмеженими.

Нехай $x = \{x_{ik}\}_{i=\overline{1,m}}^{k=\overline{1,l}}$, де x_{ik} – кількість одиниць продукції, яка перевозиться від постачальника A_i до пункту D_k; $y = \{y_{kj}\}_{k=\overline{1,l}}^{j=\overline{1,n}}$, де y_{kj} – кількість продукції від пункту D_k до споживача B_j; $z = \{z_k\}_{k=\overline{1,l}}$, де z_k – булева змінна, яка дорівнює одиниці, якщо проміжний пункт D_k використовується, та дорівнює нулю – в протилежному випадку.

Двоетапна транспортна задача для знаходження не більше, ніж d проміжних пунктів, які визначають найбільш економічний план перевезення продукції від постачальників до споживачів, має такий вигляд: знайти

$$f^*_{xyz} = f_{xyz}\left(x^*, y^*, z^*\right) = \min_{x,y,z}\left\{ f\left(x,y\right) = \sum_{i=1}^{m}\sum_{k=1}^{l} c_{ik}x_{ik} + \sum_{k=1}^{l}\sum_{j=1}^{n} c_{kj}y_{kj} \right\} \quad (6.6)$$

за обмежень

$$\sum_{k=1}^{l} x_{ik} = a_i, \quad i = \overline{1,m}, \quad (6.7)$$

$$\sum_{k=1}^{l} y_{kj} = b_j, \quad j = \overline{1,n}, \quad (6.8)$$

$$\sum_{i=1}^{m} x_{ik} - \sum_{j=1}^{n} y_{kj} = 0, \quad k = \overline{1,l}, \quad (6.9)$$

$$\sum_{i=1}^{m} x_{ik} \le z_k \sum_{i=1}^{m} a_i, \quad k = \overline{1,l}, \quad (6.10)$$

$$\sum_{k=1}^{l} z_k \le d, \quad (6.11)$$

$$x_{ik} \ge 0,\ y_{kj} \ge 0,\ z_k = 0 \vee 1,\ i = \overline{1,m},\ k = \overline{1,l},\ j = \overline{1,n}. \quad (6.12)$$

Задача (6.6) – (6.12) є задачею булевого лінійного програмування, яка містить $\left(m+n\right)\times l$ неперервних змінних x та y, l булевих змінних z, $m+n+2l+1$ обмежень. Цільова функція (6.6) задає сумарні витрати на транспортування продукції від постачальників до споживачів. Обмеження (6.7) означають транспортування $a_1,...,a_m$ одиниць продукції із пунктів постачання до проміжних пунктів, а обмеження (6.8) – що споживачам потрібно доставити необхідні об'єми $b_1,...,b_n$ одиниць продукції з проміжних пунктів. Обмеження (6.9) задають умови на те, щоб вся продукція, яка приходить від постачальників до кожного проміжного пункту, була обов'язково відправлена споживачам. Обмеження (6.11) означає, що задіяними повинні бути не більше, ніж d

проміжних пунктів, а обмеження (6.10) визначають, які саме проміжні пункти будуть задіяні.

Якщо для задачі (6.6) – (6.12) вибрати $d = l$, що означає, що при транспортуванні продукції задіяними можуть бути усі l проміжних пунктів, то тоді обмеження (6.10) та (6.11) можна прибрати. В результаті отримаємо добре вивчену класичну двоетапну транспортну задачу (Карагодова, Кігель, та Рожок 2007), (Наконечний і Савіна 2003), (Стецюк та ін. 2017), (Стецюк, Мазютинець, та Мілешовський 2017), (Стецюк, Ляшко, та Мазютинець 2018), (Стецюк, Лиховид, та Супрун 2020). Тому справедливі наступні твердження.

Теорема 6.1. Якщо $d = l$, то задача (6.6) – (6.12) переходить в двоетапну транспортну задачу: знайти

$$f_{xy}^{*} = f_{xy}\left(x^{*}, y^{*}\right) = \min_{x,y}\left\{ f\left(x,y\right) = \sum_{i=1}^{m}\sum_{k=1}^{l} c_{ik}x_{ik} + \sum_{k=1}^{l}\sum_{j=1}^{n} c_{kj}y_{kj} \right\} \qquad (6.13)$$

за обмежень

$$\sum_{k=1}^{l} x_{ik} = a_i, \quad i = \overline{1,m}, \qquad (6.14)$$

$$\sum_{k=1}^{l} y_{kj} = b_j, \quad j = \overline{1,n}, \qquad (6.15)$$

$$\sum_{i=1}^{m} x_{ik} - \sum_{j=1}^{n} y_{kj} = 0, \quad k = \overline{1,l}, \qquad (6.16)$$

$$x_{ik} \geq 0, \; y_{kj} \geq 0, \; i = \overline{1,m}, \; k = \overline{1,l}, \; j = \overline{1,n}. \qquad (6.17)$$

Якщо в задачі (6.13) – (6.17) загальна кількість продукції постачальників дорівнює загальному попиту всіх споживачів, тобто виконується рівність

$$\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j, \qquad (6.18)$$

то таку двоетапну транспортну задачу називають збалансованою, або закритою. Якщо умова (6.18) не виконується, то двоетапну транспортну задачу називають незбалансованою або відкритою. Останню легко звести до задачі закритого

типу: у разі перевищення загального попиту над запасами – за допомогою введення фіктивного постачальника A_{m+1} із кількістю $a_{m+1} = \sum_{j=1}^{n} b_j - \sum_{i=1}^{m} a_i$ одиниць продукції; у разі перевищення запасів над загальним попитом – за допомогою введення фіктивного споживача B_{n+1} з потребою $b_{n+1} = \sum_{i=1}^{m} a_i - \sum_{j=1}^{n} b_j$ одиниць продукції.

6.4 Про властивості задачі (6.6) – (6.12)

Згідно з теоремою 6.1 задача $(6.6)-(6.12)$ співпадає з класичною двоетапною транспортною задачею, якщо $d = l$, $1 \le d \le l$. Цей випадок відповідає тому, що d вибирається рівним правій границі інтервалу $[1, l]$ та означає, що при транспортуванні продукції від постачальників до споживачів задіяними можуть бути усі l проміжних пунктів. Якщо $d = 1$, то d вибирається рівним лівій границі інтервалу $[1, l]$, та означає, що всі постачальники відправляють продукцію у єдиний проміжний пункт, з якого кожний споживач забирає свою кількість одиниць продукції. Якщо задача $(6.6)-(6.12)$ збалансована, тобто виконується умова (6.18), то граничному випадку $d = 1$ відповідає єдиний оптимальний план $x^* = \left\{ x_{i1}^* = a_i \right\}_{i=\overline{1,m}}$, $y^* = \left\{ y_{1j}^* = b_j \right\}_{j=\overline{1,n}}$, при якому реалізується мінімальне значення цільової функції $f_{xy} = \sum_{i=1}^{m} c_{i1} a_i + \sum_{j=1}^{n} c_{1j} b_j$.

Якщо виконується умова $1 < d < l$, то тоді задача $(6.6)-(6.12)$ буде задачею лінійного булевого програмування. В обох крайніх випадках задача булевого лінійного програмування фактично відсутня, так як значення булевих змінних z визначаються однозначно. У першому випадку всі невідомі компоненти z

дорівнюють одиниці, а у другому випадку – єдина булева змінна приймає значення, яке дорівнює одиниці.

Теорема 6.2. Система обмежень (6.7) – (6.12) є несумісною, якщо

$$\sum_{i=1}^{m} a_i \neq \sum_{j=1}^{n} b_j \, .$$

Теорема 6.2 дає можливість перевірити коректність вхідних даних для задачі (6.6) – (6.12). Іншими словами, вона дає можливість впевнитися, що задача (6.6) – (6.12) є збалансованою. Якщо задача є збалансованою, то система (6.7) – (6.12) є сумісною, і переходимо до розв'язання задачі (6.6) – (6.12). Якщо задача є незбалансованою, то закінчуємо розрахунок з повідомленням «задача (6.6) – (6.12) не має розв'язку».

6.5 AMPL-реалізація задачі

Один із можливих способів розв'язання задачі (6.6) – (6.12) полягає у використанні AMPL (A Mathematical Programming Language) (Fourer, Gay, and Kernighan 2003) та сучасного програмного забезпечення для розв’язання задач цілочислового лінійного програмування, наприклад, програми Gurobi (Gurobi 2023). AMPL-код для опису оптимізаційної задачі (6.6) – (6.12) має наступний вигляд:

```
param m>=2; # Кількість постачальників (пункти A)
param l>=1; # Кількість проміжних пунктів (пункти D)
param n>=2; # Кількість споживачів (пункти B)
param d>=1<=m; # границя зверху на кількість пунктів D
# Вартості перевезення одиниці продукції:
param cik{i in 1..m, k in 1..l} >= 0; # від A до D
param ckj{k in 1..l, j in 1..n} >= 0; # від D до B
# Інші дані
param a{i in 1..m} >= 0; # Продукція в A
param b{j in 1..n} >= 0; # Потреби в B
# теорема 1.4.1
check: sum{i in 1..m} a[i] = sum{j in 1..n} b[j];
# Невідомі (продукція, яку потрібно перевезти)
var x{i in 1..m, k in 1..l} >=0; # від A до D
```

```
var y{k in 1..l, j in 1..n} >=0; # від D до B
var z{k in 1..l} binary;    # 1-задіяно D, 0-незадіяно D
# Мінімізувати витрати на перевезення продукції:
minimize f_opt:
sum{i in 1..m, k in 1..l} cik[i,k]*x[i,k]+ # від A до D
sum{k in 1..l, j in 1..n} ckj[k,j]*y[k,j]; # від D до B
subject to # за обмежень
con2 {i in 1..m}: # перевезення продукції з A до D
sum{k in 1..l}x[i,k] = a[i];
con3 {j in 1..n}: # задоволення потреб B з D
sum{k in 1..l}y[k,j] = b[j];
con4 {k in 1..l}: # всю продукцію треба доставити в B
sum{i in 1..m}x[i,k]-sum{j in 1..n}y[k,j]=0;
con5 {k in.. 1..l}: # вкл./викл. проміжний пункт
sum{i in 1..m}x[i,k]<=z[k]*sum{i in 1..m}a[i];
con6:  # вибрати не більше ніж d проміжних пунктів
sum{k in 1..l}z[k]<=d;
```

Тут оператор **param** використано для опису розмірів та даних задачі; оператор **var** – для опису змінних задачі, де враховано їх невід'ємність та булевість; оператор **check** – для перевірки сумісності обмежень (6.7) – (6.13) згідно теореми 6.2, яка вимагає, щоб уся продукція постачальників була відправлена споживачам.

Якщо цей AMPL-код доповнити необхідними даними для конкретної задачі, то його можна використовувати для розв'язання задачі (6.6) – (6.12) за допомогою стандартного програмного забезпечення для розв'язання задач цілочислового лінійного програмування.

6.6 Тестовий приклад

Виробниче об'єднання складається з трьох філіалів: A_1, A_2, A_3, які виготовляють однорідну продукцію в обсягах відповідно 200, 220 та 380 одиниць на місяць. Ця продукція відправляється на три склади D_1, D_2 і D_3, а потім – до чотирьох споживачів B_1, B_2, B_3, B_4, попит яких становить відповідно 150, 50, 350 і 250 одиниць. Вартості перевезень одиниці продукції (в

умовних одиницях) від виробників на склади, а потім – зі складів до споживачів наведено у наступних таблицях.

$A \setminus D$	D_1	D_2	D_3
A_1	5	2	5
A_2	3	1	3
A_2	4	7	1

$D \setminus B$	B_1	B_2	B_3	B_4
D_1	3	2	5	2
D_2	7	1	3	1
D_3	8	5	6	5

Для тестового прикладу опис вхідних даних задачі (6.6) – (6.12) реалізує наступний AMPL-код:

```
data; # Блок вхідних даних, де задаємо:
param m := 3; # Кількість постачальників A
param l := 3; # Кількість проміжних пунктів D
param n := 4; # Кількість споживачів B
param d := 3; # границя зверху на кількість D
# Витрати на перевезення одиниці продукції:
param cik: 1 2 3 := # від A ( ) до D ( )
          1 5 2 5
          2 3 1 3
          3 4 7 1;
param ckj: 1 2 3 4 := # від D ( ) до B ( )
          1 3 2 5 2
          2 7 1 3 1
          3 8 5 6 5;
param a:= # Продукція в A
1 200 2 220 3 380; # A_1, A_2, A_3
param b:= # Потреби B
1 150 2 50 3 350 4 250; # B_1, B_2, B_3, B_4
options solver gurobi;
solve; # Розв'язати задачу (1)-(5)
# Роздрукувати значення цільової функції та
# час розв'язання
display f_opt, _solve_time;
# Роздрукувати значення оптимальних змінних
display x; display y; display z;
```

Розв’язок задачі (6.6) – (6.12) для тестового прикладу при $d = 3$ представлений на рисунку 6.4. Для нього значення цільової функції дорівнює

3940 умовних одиниць, склад D_1 завантажений продукцією на 150 одиниць, склад D_2 – на 420 одиниць, а склад D_3 – на 230 одиниць.

Розв’язок задачі (1.4.1) – (1.4.7) для тестового прикладу при $d = 2$ представлений на рисунку 1.4.3. Для нього значення цільової функції дорівнює 4170 умовних одиниць, склад D_1 завантажений продукцією на 380 одиниць, склад D_2 – на 420 одиниць, а склад D_3 – не використовується.

Для тестування сучасного програмного забезпечення для розв'язання двоетапних транспортних задач з обмеженням на кількість проміжних пунктів створено AMPL-коди для відповідних задач булевого лінійного програмування, орієнтовані на велику кількість постачальників, проміжних пунктів та споживачів. За його допомогою досліджено ефективність сучасного програмного забезпечення з NEOS-сервера (відомих програм Gurobi та CPLEX) для розв'язання тестових задач середніх розмірів (розглядаються декілька сотень постачальників, проміжних пунктів та споживачів).

Наведемо результати розрахунку для тестової задачі з наступними розмірами:

```
param m=100; # Кількість постачальників (пункти A)
param n=250; # Кількість споживачів (пункти B)
param l=100; # Кількість проміжних пунктів (пункти D)
param d >= 1 <= 10; # границя зверху на кількість
# пунктів D
```

Для них за змінними, обмеженнями та заповненістю матриці обмежень відповідна задача булевого лінійного програмування має такі характеристики

35100 variables:

100 binary variables

35000 linear variables

551 constraints, all linear; **80200** nonzeros

450 equality constraints

101 inequality constraints

1 linear objective; **34998** nonzeros.

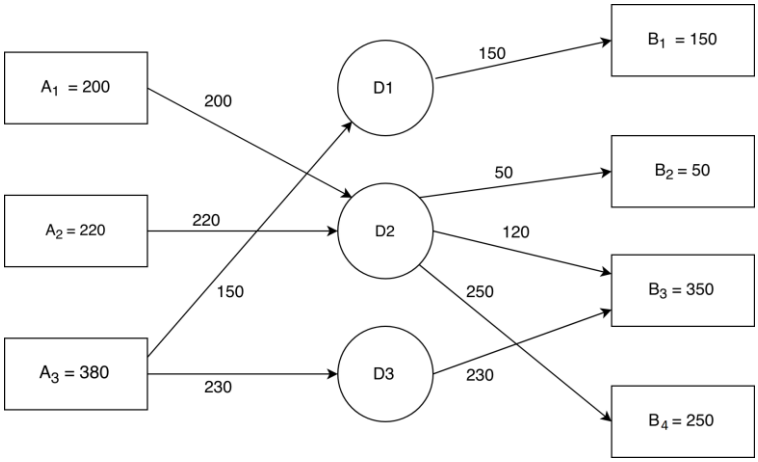


Рис. 6.4. Оптимальний план перевезень продукції при $d = 3$.

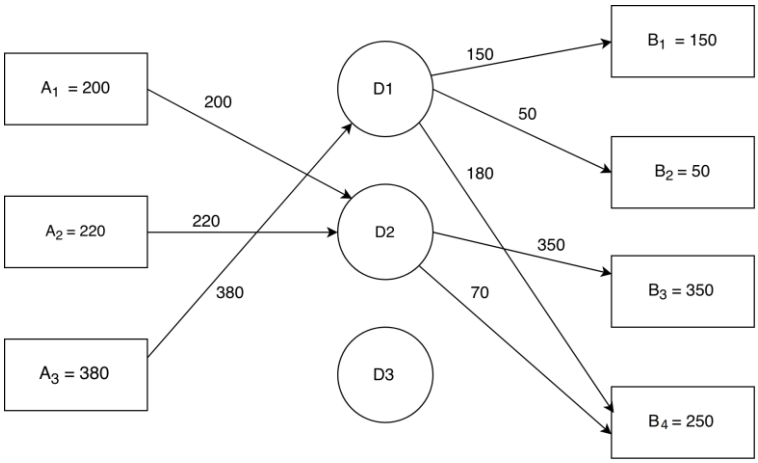


Рис. 6.5. Оптимальний план перевезень продукції при $d = 2$

Порівняння часу (в секундах), затраченого програмами Gurobi та CPLEX на розв'язання задач булевого лінійного програмування для різних значень $d \in \{2,...,10\}$, наведено у таблиці 6.3. Тут також наведено максимальну $q_{\max}$ та мінімальну $q_{\min}$ долі завантаження проміжних пунктів при реалізації оптимального плану транспортування продукції.

Таблиця 6.3

Затрати програм Gurobi та CPLEX

для $m=100$, $l=100$, $n=250$

d	f^*_{xyz}	time Gurobi (sec.)	$q_{\max}$	$q_{\min}$	time CPLEX (sec.)
2	719902	736.731	0.57	0.43	271.351
3	559784	495.499	0.36	0.29	322.735
4	483311	428.549	0.28	0.22	321.833
5	432791	478.035	0.23	0.14	324.89
6	388650	446.238	0.20	0.13	216.514
7	355522	341.772	0.18	0.12	150.851
8	331430	308.822	0.15	0.10	101.462
9	314464	438.228	0.14	0.07	106.149
10	298488	94.3449	0.13	0.06	86.6779

З таблиці 6.3 видно, що для булевої двоетапної транспортної задачі (35 000 змінних, 550 обмежень) час розв'язання за допомогою програми CPLEX складає не більше десяти хвилин, що можна використати для розміщення до 10 накопичувачів, які можуть запасати електричну енергію від 100 постачальників та можуть її передавати 250 споживачам. Аналогічні затрати часу для програм Gurobi та CPLEX будуть мати місце, якщо кількість постачальників буде дорівнювати 250, а кількість споживачів − 100.

6.7 Квадратична двоетапна транспортна задача

Її будемо розглядати у такому вигляді: знайти

$$F^* = \min_{x,y}\left\{F(x,y)=\sum_{i=1}^{m}\sum_{k=1}^{l}(\varepsilon_{ik}x_{ik}^2+c_{ik}x_{ik})+\sum_{k=1}^{l}\sum_{j=1}^{n}(\varepsilon_{kj}y_{kj}^2+c_{kj}y_{kj})\right\} \qquad (6.6)$$

за обмежень

$$\sum_{k=1}^{l} x_{ik} = a_i, \quad i = 1,\ldots,m, \tag{6.7}$$

$$\sum_{k=1}^{l} y_{kj} = b_j, \quad j = 1,\ldots,n, \tag{6.8}$$

$$\sum_{i=1}^{m} x_{ik} - \sum_{j=1}^{n} y_{kj} = 0, \quad k = 1,\ldots,l, \tag{6.9}$$

$$x_{ik} \geq 0,\ y_{kj} \geq 0, \quad i = 1,\ldots,m,\ k = 1,\ldots,l,\ j = 1,\ldots,n. \tag{6.10}$$

Якщо $\varepsilon_{ik} > 0$ та $\varepsilon_{jk} > 0$, то цільова функція (6.6) – це опукла сепарабельна квадратична функція. Відповідна їй задача (6.6) – (6.10) є задачею опуклого квадратичного програмування. Якщо для всіх $i = 1,\ldots,m$, $k = 1,\ldots,l$, $j = 1,\ldots,n$, значення $\varepsilon_{ik} = 0$ та $\varepsilon_{jk} = 0$, то задача (6.6) – (6.10) переходить у задачу лінійного програмування (6.1) – (6.5). Тут лінійні обмеження (6.7) означають постачання усієї продукції $a_1, ..., a_m$ до проміжних пунктів, а лінійні обмеження (6.8) – доставку необхідної продукції $b_1, ..., b_n$ з проміжних пунктів. Лінійні обмеження (6.9) задають умови на те, щоб вся продукція яка приходить від постачальників до кожного проміжного пункту, була обов'язково відправлена споживачам. Лінійні обмеження (6.10) задають умови на невід’ємність змінних x_{ik} та y_{kj}, $i = 1,\ldots,m$, $k = 1,\ldots,l$, $j = 1,\ldots,n$. Для обмежень (6.7) – (6.10) справедлива лема 6.1, яка дає можливість перевірити коректність вхідних даних $a_1, ..., a_m$ та $b_1, ..., b_n$. Якщо виконується умова $\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$, то система (6.7) – (6.10) є сумісною і можна переходити до розв'язання задачі (6.6) – (6.10).

Лема 6.2. Якщо $\varepsilon_{ik} > 0$, $\varepsilon_{jk} > 0$ для всіх $i = 1,\ldots,m$, $k = 1,\ldots,l$, $j = 1,\ldots,n$ та виконується умова $\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$, то задача (6.6) – (6.10) має єдиний розв'язок.

Доведення. Проведемо методом від супротивного. Нехай $x^* = \left\{x_{ik}^*\right\}_{i=1,\dots,m}^{k=1,\dots,l}$, $y^* = \left\{y_{kj}^*\right\}_{k=1,\dots,l}^{j=1,\dots,m}$ та $x^{**} = \left\{x_{ik}^{**}\right\}_{i=1,\dots,m}^{k=1,\dots,l}$, $y^{**} = \left\{y_{kj}^{**}\right\}_{k=1,\dots,l}^{j=1,\dots,m}$ – два неспівпадаючі розв'язки задачі (6.6) – (6.10). Їм відповідає оптимальне значення цільової функції $f^* = f(x^*, y^*) = f(x^{**}, y^{**})$. Обидва розв'язки x^*, y^* та x^{**}, y^{**} задовольняють обмеженням (6.7) – (6.10), тобто

$$\sum_{k=1}^{l} x_{ik}^* = a_i, \quad \sum_{k=1}^{l} x_{ik}^{**} = a_i, \quad i = 1,\dots,m, \tag{6.11}$$

$$\sum_{k=1}^{l} y_{kj}^* = b_j, \quad \sum_{k=1}^{l} y_{kj}^{**} = b_j, \quad j = 1,\dots,n, \tag{6.12}$$

$$\sum_{i=1}^{m} x_{ik}^* - \sum_{j=1}^{n} y_{kj}^* = 0, \quad \sum_{i=1}^{m} x_{ik}^{**} - \sum_{j=1}^{n} y_{kj}^{**} = 0, \quad k = 1,\dots,l, \tag{6.13}$$

$$x_{ik}^* \ge 0,\ y_{kj}^* \ge 0,\ x_{ik}^{**} \ge 0,\ y_{kj}^{**} \ge 0, \quad i = 1,\dots,m,\ k = 1,\dots,l,\ j = 1,\dots,n. \tag{6.14}$$

Точки $x^{***} = \lambda x^* + (1-\lambda)x^{**}$ та $y^{***} = \lambda y^* + (1-\lambda)y^{**}$, де $0 < \lambda < 1$, задовольняють системі обмежень (6.7) – (6.10). Дійсно, враховуючи рівності (6.11), для всіх $i = 1,\dots,m$ отримуємо

$$\sum_{k=1}^{l} x_{ik}^{***} = \sum_{k=1}^{l} \left(\lambda x_{ik}^* + (1-\lambda)x_{ik}^{**}\right) = \lambda \sum_{k=1}^{l} x_{ik}^* + (1-\lambda)\sum_{k=1}^{l} x_{ik}^{**} = \lambda a_i + (1-\lambda)a_i = a_i,$$

що означає, що точка x^{***} задовольняє рівності (6.7). Враховуючи рівності (6.12), для всіх $j = 1,\dots,n$ отримуємо

$$\sum_{k=1}^{l} y_{kj}^{***} = \sum_{k=1}^{l} \left(\lambda y_{kj}^* + (1-\lambda)y_{kj}^{**}\right) = \lambda \sum_{k=1}^{l} y_{kj}^* + (1-\lambda)\sum_{k=1}^{l} y_{kj}^{**} = \lambda b_j + (1-\lambda)b_j = b_j,$$

що означає, що точка y^{***} задовольняє рівності (6.8). Аналогічно, використовуючи рівності (6.13) та нерівності (6.14), легко показати, що точки x^{***} та y^{***} задовольняють рівності (6.9) та нерівності (6.10).

Якщо $\varepsilon_{ik} \geq 0$ та $\varepsilon_{jk} \geq 0$, то для всіх $i = 1,\ldots,m$, $k = 1,\ldots,l$, $j = 1,\ldots,n$, квадратичні функції $F_{ik}(x_{ik}) = \varepsilon_{ik} x_{ik}^2 + c_{ik} x_{ik}$ та $F_{kj}(y_{kj}) = \varepsilon_{kj} y_{kj}^2 + c_{kj} y_{kj}$ є строго опуклими, а значить квадратична функція

$$F(x,y) = \sum_{i=1}^{m}\sum_{k=1}^{l}(\varepsilon_{ik} x_{ik}^2 + c_{ik} x_{ik}) + \sum_{k=1}^{l}\sum_{j=1}^{n}(\varepsilon_{kj} y_{kj}^2 + c_{kj} y_{kj}) \qquad (6.15)$$

є також строго опуклою. Тому, якщо $0 < \lambda < 1$, то для $f(x^{***}, y^{***})$ – значення цільової функції у точці (x^{***}, y^{***}) справедливі наступні співвідношення:

$$F(x^{***}, y^{***}) = F(\lambda x^{*} + (1-\lambda)x^{**}, \lambda y^{*} + (1-\lambda)y^{**}) <$$
$$< \lambda F(x^{*}, y^{*}) + (1-\lambda)F(x^{**}, y^{**}) = \lambda F^{*} + (1-\lambda)F^{*} = F^{*},$$

з яких випливає нерівність $F(x^{***}, y^{***}) < F^{*}$. Вона суперечить тому, що (x^{*}, y^{*}) та (x^{**}, y^{**}) є розв'язками задачі (6.6) – (6.10), так як в точці (x^{***}, y^{***}), яка задовольняє обмеженням (6.7) – (6.10), значення цільової функції $F(x^{***}, y^{***})$ є меншим за мінімальне значення F^{*}. Лему 6.2 доведено.

Частковим випадком задачі (6.6) – (6.10) є задача лінійного програмування (6.1) – (6.5). Їй відповідають $\varepsilon_{ik} = 0$ і $\varepsilon_{jk} = 0$ для всіх постачальників $i = 1,\ldots,m$, проміжних пунктів $k = 1,\ldots,l$ та споживачів $j = 1,\ldots,n$. Для задачі (6.1) – (6.5) точка мінімуму не завжди є єдиною (див. тестовий приклад, підрозділ 6.2). Якщо вибирати досить малими значення величин ε_{ik} та ε_{jk}, то це дозволяє наблизити розв'язок задачі (6.6) – (6.10) до одного з розв'язків задачі (6.1) – (6.5), якщо остання має багато розв'язків. При цьому як оптимальний розв'язок буде вибрана одна з внутрішніх точок множини оптимальних розв'язків задачі лінійного програмування (6.1) – (6.5), яка визначається вибором величин ε_{ik} та ε_{jk} у задачі квадратичного програмування (6.6) – (6.10).

6.8 Висновки до шостого розділу

1. Наведено формулювання та описано властивості двох варіантів двоетапної транспортної задачі: класичної та задачі з обмеженнями на кількість проміжних пунктів. Наведено AMPL-код для розв'язання двоетапної транспортної задачі лінійного програмування за допомогою сучасного програмного забезпечення для задач лінійного програмування. Наведено та проаналізовано результати розрахунку за допомогою програми Gurobi для двоетапної транспортної задачі, яка має багато розв'язків.
2. Сформульовано двоетапну транспортну задачу квадратичного програмування та досліджено умови, при яких вона має єдиний розв’язок.

ЗАГАЛЬНІ ВИСНОВКИ

У дисертаційній роботі проведено дослідження квазіньютонівських алгоритмів та субградієнтних алгоритмів з розтягом простору, розроблено нові градієнтні та субградієнтні алгоритми з перетворенням простору, розроблено нові математичні моделі для розв'язання прикладних задач.

Основні результати дисертаційної роботи.

1. Запропоновано і описано B-форму алгоритму Давидона – Флетчера – Пауела – $DFPR(\alpha)$-алгоритм, який є проміжним між ДФП-методом та r-алгоритмами. Проведено низку обчислювальних експериментів для порівняння цього методу з r-алгоритмами на прикладі мінімізації квадратичних функцій, в тому числі й сильно яружних функцій. Окреслено можливі схеми такого типу методів для мінімізації негладких опуклих функцій.
2. Запропоновано нове сімейство субградієнтних алгоритмів з розтягом простору у напрямку модифікованої різниці двох субградієнтів у перетвореному просторі, частковим випадком якого є r-алгоритм. Результати тестових експериментів показують, що для мінімізації кусково-лінійної та квадратичної строго опуклої функцій кількість ітерацій менша, як порівняти з класичним r-алгоритмом.
3. Запропоновано оптимізаційну модель для задачі знаходження оцінок параметрів квантильної регресії. Модель формулюється як задача безумовної мінімізації кусково-лінійної функції.
4. Запропоновано оптимізаційну модель для задачі побудови S-подібної кривої. Модель формулюється як задача мінімізації гладкої функції суми нев'язок з простими двосторонніми обмеженнями на змінні. Наведено результати застосування методів BFGS та L-BFGS-B для розв'язання цієї задачі.

5. Запропоновано два класи оптимізаційних моделей для задачі знаходження пропускних спроможностей дуг відмовостійких орієнтованих мереж. Моделі описуються задачами лінійного, змішаного булевого лінійного та нелінійного програмування з блочною структурою матриці обмежень. Розроблено декомпозиційні методи на основі *r*-алгоритму та їх програмні реалізації. Результати обчислювальних експериментів з цими реалізаціями демонструють їхню конкурентоспроможність як порівняти з програмою IPOPT.
6. Побудовано квадратичне формулювання оптимізаційної задачі для знаходження максимального k-плекса у неорієнтованому графі. Розроблено алгоритм пошуку всіх максимальних k-плексів для неорієнтованого графа. В основі алгоритму лежить послідовне додавання до задачі лінійного булевого програмування додаткового обмеження, яке відсікає вже знайдені максимальні k-плекси.
7. Описано властивості двох варіантів двоетапної транспортної задачі: класичної та задачі з обмеженнями на кількість проміжних пунктів. Сформульовано двоетапну транспортну задачу квадратичного програмування та досліджено умови, при яких вона має єдиний розв’язок.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Григорак, Марія. 2017. *Інтелектуалізація ринку логістичних послуг: концепція, методологія, компетентність: монографія*. Київ: Сік Груп Україна.

Карагодова, Олена, Володимир Кігель, та Віктор Рожок. 2007. *Дослідження операцій: Навч. пос*. К.: Центр учбової літератури.

Наконечний, Степан, та Світлана Савіна. 2003. *Математичне програмування: Навч. посіб*. К.: КНЕУ.

Стецюк, Петро, Тамара Бєлих, та Олена Криворучко. 2019. Теорія та програмні реалізації r-алгоритмів Шора. У *Субградієнтні алгоритми та задачі на комбінаторних конфігураціях*, під загал. ред. П.І. Стецюка, 5–30. Київ: ПУЛЬСАРИ.

Стецюк, Петро, Василь Горбачук, Ольга Хом'як, Володимир Жидков, та Антон Супрун. 2019. *Розроблення оптимізаційних процедур для задач розташування накопичувачів електроенергії в ОЕС України в сучасних умовах технологічних змін. Етап 1. Розроблення математичних моделей, методів та програмного забезпечення для спеціальних класів двоетапних транспортних задач*. Заключний звіт про науково-дослідну роботу (№ держ. реєстрації 0119U001641). К.: Ін-т кібернетики імені В.М. Глушкова НАН України.

Стецюк, Петро, та Олександр Жмуд. 2020. «Про прискорену реалізацію оператора розтягу простору». Тези доповідей XVIII міжнародної науково-практичної конференції «Математичне та програмне забезпечення інтелектуальних систем (MPZIS-2020)», Дніпро, Листопад 18–20.

Стецюк, Петро, Олексій Лиховид, Володимир Жидков, та Антон Супрун. 2021. Оптимізаційні задачі модернізації пропускних здатностей дуг відмовостійких мереж. *Проблеми управління та інформатики* 5:5–20.

Стецюк, Петро, Олексій Лиховид, та Антон Супрун. 2020. Про лінійну та квадратичну двоетапні транспортні задачі. *Кібернетика та комп'ютерні технології* 4:5–14.

Стецюк, Петро, Володимир Ляшко, та Габріела Мазютинець. 2018. Двоетапна транспортна задача та її AMPL-реалізація. *Наукові записки НаУКМА. Комп'ютерні науки* 1:14–20.

Стецюк, Петро, Габріела Мазютинець, та Борис Мілешовський. 2017. «AMPL-реалізація двоетапної транспортної задачі». Тези XV Ювілейної Міжнародної науково-практичної конференції «Математичне та програмне забезпечення інтелектуальних систем», Дніпровський національний університет ім. Олеся Гончара, Дніпро, Листопад 22–24.

Стецюк, Петро, Олександр Міца, Олег Стрелюк, та Олександр Фесюк. 2017. Транспортна задача з обмеженнями на пропускні спроможності проміжних пунктів. *Питання прикладної математики і математичного моделювання* 17:207–219.

Стецюк, Петро, Володимир Ляшко, та Антон Супрун. 2020. «Метод BFGS для задачі побудови s-подібної кривої». *Наукові записки НаУКМА. Комп'ютерні науки* 3:102–106.

Стецюк, Петро, Ольга Слабоспицька, та Олена Ушакова. 2016. Максимальні незалежні множини вершин графа та їх застосування в керуванні проектами. *Питання прикладної математики і математичного моделювання* :151–162.

Стецюк, Петро, Марія Стецюк, Даніїл Брагін, та Микола Молодик. 2021. «Використання r-алгоритму Шора в лінійних задачах робастної оптимізації». *Кібернетика та комп'ютерні технології* 1: 29–42.

Стецюк, Петр, Виктор Стовба, и Антон Супрун. 2021. «B-форма метода Давидона – Флетчера – Пауэлла». *Журнал обчислювальної та прикладної математики* 2(136):93–110.

Стецюк, Петро, Олександр Ткаченко, та Ольга Грицай. 2020. «До побудови зовнішнього контура сопла Франкля за квадратичною кривиною». *Кібернетика та комп'ютерні технології* 1:23–31.

Стецюк, Петро, Ольга Хом'як, Єгор Блохін, та Антон Супрун. 2022. Оптимізаційні задачі для максимального k-плекса. *Кібернетика та системний аналіз* 58(4):46–58.

Супрун, Антон. 2023. «Обчислювальні експерименти для r-алгоритму з прискореною реалізацією розтягу простору». *Кібернетика та комп'ютерні технології* 2:46–54.

Шор, Наум, Іван Сергієнко, Володимир Шило, та Петро Стецюк. 2005. *Задачі оптимального проектування надійних мереж*. К.: Наук. Думка.

Борисенко, Валерий, Сергей Устенко, и Ирина Устенко. 2018. «Геометрическое моделирование *s*-образных скелетных линий профилей лопаток осевых компрессоров». Вестник двигателестроения 1:45–52.

Брэгман, Лев. 1967. «Релаксационный метод нахождения общей точки выпуклых множеств и его применение для решения задач выпуклого программирования». *ЖВМ и МФ* 7(3):200–217.

Гилл, Филипп, Уолтер Мюррей, и Маргарет Райт. 1985. Практическая оптимизация. М.: Мир.

Демьянов, Владимир, и Леонид Васильев. 1981. *Недифференцируемая оптимизация*. Серия: Оптимизация и исследование операций. М.: Наука.

Дикин, Илья, и Валерий Зоркальцев. 1980. *Итеративное решение задач математического программирования*. Новосибирск: Наука.

Еремин, Иван. 1965. «Обобщение релаксационного метода Агмона-Моцкина». *УМН* 20(122):183–187.

Ермольев, Юрий. 1966. «Методы решения нелинейных экстремальных задач». *Кибернетика* (4):1–17.

Зоркальцев, Валерий, Сергей Пержабинский, и Петр Стецюк. 2015. Поиск нормальных решений СЛАУ при двусторонних ограничениях на переменные методом внутренних точек. *Кибернетика и системный анализ* 6: 71–80.

Измаилов, Алексей, Алексей Куренной, и Петр Стецюк. 2019. Метод Левенберга – Марквардта для задач безусловной оптимизации. *Вестник Тамбовского университета. Серия: Естественные и технические науки* 24(125):60–74.

Мищенко, Александр, и Анатолий Фоменко. 2004. *Краткий курс дифференциальной геометрии и топологии*. М.: ФИЗМАТЛИТ.

Полак, Элиях. 1974. *Численные методы оптимизации. Единый подход*. М.: Мир.

Поляк, Борис. 1967. «Один общий метод решения экстремальных задач». *ДАН СССР* 174(1):33–36.

Поляк, Борис. 1969. «Минимизация негладких функционалов». *ЖВМ и МФ* 9(3):509–521.

Поляк, Борис. 1983. *Введение в оптимизацию*. М.: Наука.

Пшеничный, Борис, и Юрий Данилин. 1975. *Численные методы в экстремальных задачах*. М.: Наука.

Стецюк, Петр. 1995. «К вопросу сходимости r-алгоритмов". *Кибернетика и системный анализ* 6:173–177.

Стецюк, Петр. 1996. *Квазиньютоновские методи и r-алгоритмы*. Препр. НАН Украины. Ин-т кибернетики им. В.М. Глушкова. Киев.

Стецюк, Петр. 1997. «Ортогонализующие линейные операторы в выпуклом программировании. I». *Кибернетика и системный анализ* (3):97–119.

Стецюк, Петр. 1998. Линейные операторы в квазиньютоновских методах. *Теория и приложения методов оптимизации* :3–8.

Стецюк, Петр. 2005. О функционально избыточных ограничениях для булевых оптимизационных задач квадратичного типа. *Кибернетика и системный анализ* 6: 168–172.

Стецюк, Петр. 2006. Новые модели квадратичного типа для задачи о максимальном взвешенном разрезе графа. *Кибернетика и системный анализ* 1:63–75.

Стецюк, Петр. 2011. «Субградиентные методы с преобразованием пространства для минимизации овражных выпуклых функций». Международная

конференция, посвященная 90-летию со дня рождения академика Н.Н. Яненко «Современные проблемы прикладной математики и механики: теория, эксперимент и практика», Россия, Новосибирск, май 30 – июнь 4.

Стецюк, Петр. 2014. *Методы эллипсоидов и r-алгоритмы*. Кишинэу: Эврика.

Стецюк, Петр. 2017. «Теория и программные реализации r-алгоритмов Шора». *Кибернетика и системный анализ* 53(5): 43–57.

Стецюк, Петр. 2018. Двойственные оценки в квадратичных экстремальных задачах. Кишинэу: Эврика.

Стецюк, Петр, и Д. Буханцов. 2002. «К ускорению метода эллипсоидов с помощью использования шарового слоя». *Теория оптимальных решений* :63–70.

Хачиян, Леонид. 1979. «Полиномиальный алгоритм в линейном программировании». *Вычисл. математика и матем. физика* 20(2):51–68.

Шор, Наум. 1972. «О классе почти-дифференцируемых функций и одном методе минимизации функций этого класса». *Кибернетика* (4):65–70.

Шор, Наум. 1975. «Исследование сходимости метода градиентного типа с растяжением пространства в направлении разности двух последовательных субградиентов». *Кибернетика* (4):48–53.

Шор, Наум. 1977. «Метод отсечения с растяжением пространства для решения задач выпуклого программирования». *Кибернетика* (1):94–95.

Шор, Наум. 1979. *Методы минимизации недифференцируемых функций и их приложения*. К.: Наук. думка.

Шор, Наум, и Владимир Гершович. 1979. «Об одном семействе алгоритмов для решения задач выпуклого программирования». *Кибернетика* (4):62–67.

Шор, Наум, и Владимир Гершович. 1982. «Метод эллипсоидов, его обобщения и приложения». *Кибернетика* 5:61–69.

Шор, Наум, и Николай Журбенко. 1971. «Метод минимизации, использующий операцию растяжения пространства в направлении разности двух последовательных субградиентов». *Кибернетика* (3):51–59.

Шор, Наум, Николай Журбенко, Алексей Лиховид, и Петр Стецюк. 2003. «Развитие алгоритмов недифференцируемой оптимизации и их приложения». *Кибернетика и системный анализ* (4):80–94.

Шор, Наум, и Сергей Стеценко. 1989. *Квадратичные экстремальные задачи и недифференцируемая оптимизация*. К.: Наук. думка.

Юдин, Давид, и Аркадий Немировский. 1976. «Информационная сложность и эффективные методы решения выпуклых экстремальных задач». *Экономика и матем. методы* 12(1):357–369.

Agmon, Shmuel. 1954. “The relaxation method for linear inequalities”. *Canad. J. Math*. (6):382−92.

Ahuja, Ravindra, Thomas Magnanti, and James Orlin. 1993. *Network flows: Theory, algorithms, and applications*. Engle-wood Cliffs, NJ: Prentice-Hall.

Armijo, Larry. 1966. “Minimization of functions having Lipschitz continuous first partial derivatives”. *Pacific J. Math*. 16(1): 1–3.

Balansundaram, Balabhaskar, Sergiy Butenko, and Illya Hicks. 2011. “Clique Relaxations in Social Network Analysis: The Maximum k-plex Problem”. *Operations Research* 59(1):133–142.

Bonnans, J.F., J.C. Gilbert, Claude Lemaréchal, and Claudia A. Sagastizábal. 2006. *Numerical Optimization: Theoretical and Practical Aspects*. Springer-Verlag, Berlin, Heidelberg.

Boyd, Stephen, and Almir Mutapcic. 2007. *Subgradient Methods*. Lecture notes for EE364b. Stanford University.

Broyden, Charles. 1967. “Quasi-Newton methods and their application to function minimisation”. *Mathematics of Computation* 21:368–381.

Broyden, Charles. 1970. “The convergence of a class of double-rank minimization algorithms”. *Journal of the Institute of Mathematics and Its Applications* 6:76–90.

Byrd, Richard, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. 1995. “A Limited Memory Algorithm for Bound Constrained Optimization”. *SIAM Journal on Scientific and Statistical Computing* 16(5):1190–1208.

Camerini P., L. Fratta L, and F. Maffioli. 1975. “On improving relaxation methods by modified gradient techniques”. *Math. Program* (3):26–34.

Conn, Andrew R., Nicholas I.M. Gould, and Philippe L. Toint. 1991. “Convergence of quasi-Newton matrices generated by the symmetric rank one update”. *Mathematical Programming* 50:177–195.

Davidon, William. 1959. *Variable metric method for minimization* (Technical report ANL-5990). Argonne National Laboratory: Argonne, IL.

Dennis, John, and Jorge Morè. 1977. “Quasi-Newton Methods, Motivation and Theory”. *SIAM Review* 19(1):46–89.

Fletcher, Roger. 1970. “A New Approach to Variable Metric Algorithms”. *Computer Journal* 13(3):317–322.

Fletcher, Roger, and M. J.D. Powell. 1963. “A Rapidly Convergent Descent Method for Minimization”. *The Computer Journal* 6(2):163–168.

Fletcher, Roger, and C.M. Reeves. 1964. “Function minimization by conjugate gradients”. *The Computer Journal* 7(2):149–154.

Fourer, Robert, David Gay, and Brian Kernighan. 2003. *AMPL, A Modeling Language for Mathematical Programming*. Belmont: Duxburry Press.

Gill, Philip, Walter Murray, and Margaret H. Wright. 1981. *Practical Optimization*. Academic Press, London.

Goldfarb, Donald. 1970. “A Family of Variable Metric Updates Derived by Variational Means”. *Mathematics of Computation* 24(109):23–26.

Grötschel, Martin, László Lovasz, and Alexander Schriyver. 1981. “The ellipsoid method and its consequences in combinatorial optimization”. *Combinatorica* 1(2):169–197.

Guo, Jiong, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. 2010. A more relaxed model for graph-based data clustering: s-plex cluster editing. *SIAM J. Discrete Math.* 24(4):1662–1683.

Gurobi. 2023. "The Leader in Decision Intelligence Technology – Gurobi Optimization". Accessed September 15. https://www.gurobi.com.

Haelterman, Robby. 2009. "Analytical study of the Least Squares Quasi-Newton method for interaction problems". PhD diss., Universiteit Gent.

Ipopt. 2023. "COIN-OR Interior Point Optimizer IPOPT". Accessed September 15. https://github.com/coin-or/Ipopt

Kelley, J. 1960. "The cutting plane method for solving convex problems". *J. Soc. For Industr. and Appl. Math*. 8(4):703–12.

Khalfan H.F., R. H. Byrd, and R.B. Schnabel. 1993. "A theoretical and experimental study of the symmetric rank one update". *SIAM Journal on Optimization* 3:1–24.

König, Hermann, and Diethard Pallaschke. 1981. "On Khachian's algorithm and minimal ellipsoids". *Numerische Mathematik* (36):211–223.

Koenker, Roger. 2017. Computation methods for quantile regression. In *Handbook of Quantile Regression*. Chapman-Hall, 57–58.

Koenker, Roger, and Gilbert Bassett. 1978. "Regression Quantiles." *Econometrica* 46(1):33–50.

Kuntsevich, A.V., and F. Kappel. 1997. *SolvOpt: The solver for local nonlinear optimization problems*. Institute for Mathematics, Karl-Franzens University of Graz, Austria.

Lemaréchal, Claude. 1975. An extension of Davidon methods to nondifferentiable problems. *Math. Progr. Study* 3:95–109.

Motzkin, Theodore, and Isaac Schoenberg. 1954. "The relaxation method for linear inequalities". *Canad. J. Math*. (6):393−404.

NEOS Server. 2023. "NEOS Solvers". Accessed September 15. https://neos-server.org/neos/solvers/

Nocedal, Jorge, and Stephen Wright. 2006. Numerical Optimization. New York, NY, USA: Springer.

Polak, E., and G. Ribière. 1969. "Note sur la convergence de méthodes de directions conjuguées". *Revue Française d'Automatique, Informatique, Recherche Opérationnelle* 3(1):35–43.

Seidman, Stephen, and Brian Foster. 1978. "A graph theoretic generalization of the clique concept". *J. of Math. Sociology* 6: 139–154.

Sergienko, Ivan. 2012. *Methods of optimization and systems analysis for problems of transcomputational complexity*. New York, Heidelberg, Dordrecht, London: Springer.

Shanno, David. 1970. "Conditioning of quasi-Newton methods for function minimization". *Mathematics of Computation* 24(111):647–656.

Shor, Naum. 1985. *Minimization Methods for Non-Differentiable Functions*. Berlin: Springer-Verlag.

Shor, Naum. 1998. *Nondifferentiable Optimization and Polynomial Problems*. London/Boston/Dordrecht: Kluwer Academic Publishers.

Shor, Naum, and Petro Stetsyuk. 1997. “Modified *r*-algorithm to find the global minimum of polynomial functions”. *Cybernetics and Systems Analysis* 33(4):482–497.

Shor, Naum, and Petro Stetsyuk. 2001. “Dual Solution of Quadratic-Type Problems by r-algorithm (subroutine DSQTPr)”. Abstracts of the Second International Workshop "Recent Advances in Non-Differentiable Optimization", Kyiv, October 1–4.

Stetsyuk, Petro. 2017a. Shor’s r-algorithms: theory and practice. In *Optimization Methods and Applications: In Honor of the 80th Birthday of Ivan V. Sergienko*, edited by Sergiy Butenko, Panos Pardalos, and Volodymyr Shylo, 495–520. Springer International Publishing.

_____. 2017b. “Theory and Software Implementations of Shor’s *r*-Algorithms”. *Cybernetics and Systems Analysis* 53:692–703.

Suprun, Anton. 2020. “Computational aspects of quantile regression”. Тези доповідей XVIII міжнародної науково-практичної конференції «Математичне та програмне забезпечення інтелектуальних систем (MPZIS-2020)», Дніпро, Листопад 18–20.

Wolfe, Philip. 1969. “Convergence Conditions for Ascent Methods”. *SIAM Review* 11(2): 226–235.

ДОДАТОК А

RATFOR ПРОГРАМА SolverA

```
# Програма SolverA знаходить мінімальні за сумарною
# вартістю значення пропускних спроможностей дуг
# орієнтованої мережі, які гарантують виконання всіх
# вимог на передачу заданих обсягів потоку в мережі
# при можливих одиничних відмовах  мережі
# (див. Модель А (4.1) – (4.5))
# Програма реалізована на основі схем декомпозиції
# за змінними та обмеженнями. Негладкі задачі
# вирішуються r-алгоритмом.

# Максимальні розміри мережі і вимог на пересилку потоків
define MaxArc 50  # Максимальне число дуг в мережі N(V,A)
define MaxVert 25 # Максимальне число вершин мережі N(V,A)
# Максимальне число поставок MaxVert*(MaxVert-1)
define MaxDemand 600
# MaxVert + 1, використовується для адресації вимог
define MaxVert1 26
# Максимально число поломок (включаючи і нульову)
define MaxTr 51
# MaxTr + 1, використовується для адресації поломок
define MaxTr1 52
define MaxArcTr 200 # Максимальне число дуг у всіх відмовах
# Використовувані номери каналів для роботи з файлами
define irdr1 1  # читається мережа
define irdr2 2  # читаються вимоги для потоків в мережі
define irdr3 3  # читаються поломки в мережі
define iprint 6 # друкується протокол роботи програми
# Постійні параметри r-алгоритму
define alp 4.d0 # коефіцієнт розтягу простору
# число одновимірних спусків без збільшення кроку
define nh 3
# коефіцієнт збільшення кроку якщо перевищено nh
define q2 1.1d0
define maxitn 500 # максимальне число ітерацій
define epsg 1.d-4 # точність зупинки по нормі субградієнта

# Опис даних задачі
# всі дані описані так
implicit real*8 (a-h,o-z),integer*2(i-n)

# Блок /NetVA/: орієнтована (directed) мережа N(V,A)
common/NetVA/nVert,nArc,iv(MaxArc),jv(MaxArc),
```

```
             cost(MaxArc),y0(MaxArc),yup(MaxArc)
# nVert – кількість вершин; nArc - кількість дуг;
# Для k-оï дуги (k=1,...,nArc) задані:
# iv(k) - початкова вершина,
# jv(k) - кінцева вершина,
# cost(k) - вартість одиничної пропускної спроможності
# y0(k) - наявне значення пропускної спроможності

# Блок /Demand/: вимоги на пересилку обсягів потоку в
# мережі N(V,A)
common/Demand/is(MaxVert1),isr(MaxDemand),dsr(MaxDemand)
# is(*) - масив покажчиків на початок поставок з вершин
# is(i) - містить покажчик на початок поставок з вершини i
#         в наявний список вершини в масивах isr(*), dsr(*)
# is(nVert+1) - містить покажчик на вільне місце в isr(*) и # dsr(*)
#         В інші вершини в масиві isr(*), dsr(*)
# isr(*)  містить індекси вершин, куди постачати
# dsr(*)  містить самі обсяги поставок

# Блок /Troubl/: Поломки в мережі (перша відповідає
# звичайному режиму функціонування мережі,
# тобто нульовій поломці)
common/Troubl/nTr,nt(MaxTr1),narct(MaxArcTr),arcmut(MaxArcTr)
# nTr       - кількість поломок в мережі (включаючи
# нульову)
# nt(nTr+1) - масив покажчиків на початок поломок
# narct(*)  - містить номери дуг для поломок
# arcmu(*)  - містить коефіцієнти зменшення пропускної
# спроможності дуг для конкретної поломки

common/Data/Penalt,Zup    # блок для установки параметрів
                          # для додаткових змінних z_ijt
# Penalt - значення штрафу для змінних z_ij
# (обчислюється...)
# Zup    - верхня межа на змінні z_ij (обчислюється ....)
# Ще два масиви:  yopt(*) - оптимальні значення
# пропускних спроможностей дуг, що додаються
dimension yopt(MaxArc),y(MaxArc)
call prtime(0)
call readDataA  #Читання і підготовка даних
```

```
# Встановити штраф і верхню межу для змінних z_ijt
# (блок /Data/)
Penalt=1.d0;
for(i=1; i<=nArc; i=i+1) {
   Penalt=Penalt+cost(i)
}
Zup=0.d0;
ind=1; ist=is(nVert+1)-1
for(i=ind; i<=ist; i=i+1) {
    Zup=Zup+dsr(i)
}
write (iprint,'(/3x,a)')' Для додаткових змінних обрані:'
write (iprint,'(13x,a,3x,g17.5)')' Penalty=',Penalt
write (iprint,'(13x,a,3x,g17.5)')' Z_upper=',Zup

call prtime(1)

ny=nArc
do i=1,ny
   y(i)=100.d0
# Інші вхідні параметри r-алгоритму
h0=1.d0; epsy=1.d-5; intp=1;
call CoordA(ny,y,h0,epsy,intp,fopt,yopt,itn,istop)

write (iprint,'(/3x,a)')' Оптимізацію закінчено:'
write (iprint,'(13x,a,3x,f15.5)')' Fopt = ',fopt
write (iprint,'(13x,a,8x,i5)')' Iteration = ',itn
write (iprint,'(13x,a,12x,i5)')' Istop = ',istop
write (iprint,'(/3x,a)')' Знайдено наступний розвязок:'
write(iprint,'(2x,3(4x,a),3x,a)')'N','i','j','yopt(i,j)'
fy=0.d0
do j=1,ny{
  write(iprint,'(2x,3i5,2(2x,f10.2))')j,iv(j),jv(j),yopt(j)
  fy=fy+cost(j)*yopt(j)
}
write (iprint,'(/3x,a,2(3x,f12.1),5x,i3)')' Разом: Fopt F(yopt)',fopt,fy
call prtime(2)
stop
end

# Програма читання і перевірки вихідних даних задачі
# Підготовка блоків /NetVA/, /Demand/ і /Troubl/
```

```
subroutine readDataA
implicit real*8 (a-h,o-z),integer*2(i-n)

common/NetVA/nVert,nArc,iv(MaxArc),jv(MaxArc),
             cost(MaxArc),y0(MaxArc),yup(MaxArc)
common/Demand/is(MaxVert1),isr(MaxDemand),dsr(MaxDemand)
common/Troubl/nTr,nt(MaxTr1),narct(MaxArcTr),arcmut(MaxArcTr)

# Робочі масиви: відповідають даним з файлу вимог
dimension is1(MaxDemand),ir1(MaxDemand),dsr1(MaxDemand)
logical Fizi

# Прочитати блок /NetVA/ і перевірити максимально можливі
# межі
read(irdr1,*)nVert,nArc
if(nVert>MaxVert) {
  write(iprint,*)' Кількість вершин перевищує максимально допустиму'
  i1=MaxVert
  write(iprint,*)' nVert=',nVert,' MaxVert=',i1
  stop
}
if(nArc>MaxArc) {
  write(iprint,*)' Кількість дуг перевищує максимально допустиму, тобто'
  i1=MaxArc
  write(iprint,*)' nArc=',nArc,' MaxArc=',i1
  stop
}
write(iprint,'(/10x,a)')' Вихідні дані для моделі A'
write(iprint,'(/3x,2(a,i5))')' Структура мережі: nVert=',nVert,' nArc=',nArc
write(iprint,'(2x,3(4x,a),6x,a,5x,a)')'N','i','j','c(i,j)','y0(i,j)'
do i=1,nArc{
   read(irdr1,*)iv(i),jv(i),cost(i),y0(i)
write(iprint,'(2x,3i5,2(2x,f10.2))')i,iv(i),jv(i),cost(i),y0(i)
   if(iv(i)<=nvert & iv(i)>=1 & jv(i)<=nvert & jv(i)>=1 &
     cost(i)>=0.d0 & y0(i)>=0.d0) next
   write(iprint,*)' Error: щось не так в попередньому рядку...'
   stop
}

# Прочитати файл вимог і перевірити максимальні межі
read(irdr2,*)ndem1
if(ndem1>MaxDemand) {
```

```
  write(iprint,*)' Занадто багато вимог на пересилку потоків '
  i1=MaxDemand
  write(iprint,*)' Кількість вимог ',ndem1,' MaxDemand=',i1
  stop
}
write(iprint,'(/3x,a)')' Вимоги на пересилку потоків в мережі '
write(iprint,'(2x,3(4x,a),6x,a)')'N','s','r','d(s,r)'
do i=1,ndem1{
   read(irdr2,*)is1(i),ir1(i),dsr1(i)
   write(iprint,'(2x,3i5,2x,f10.2)')i,is1(i),ir1(i),dsr1(i)
   if(is1(i)<=nvert & is1(i)>=1 & ir1(i)<=nvert & ir1(i)>=1 &
     dsr1(i)>=0.d0) next
   write(iprint,*)' Error: щось не так в попередньому рядку ...'
   }
is(1)=1; nelms=0;
do i=1,nVert{
   do j=1,ndem1{
      if(is1(j)==i) {
        nelms=nelms+1
        isr(nelms)=ir1(j)
        dsr(nelms)=dsr1(j)
      }
   }
   is(i+1)=nelms+1
}

# Прочитати і заповнити масив поломок
nTr=1; nt(1)=1; nt(2)=1; nelms1=0;
write(iprint,'(/3x,a)')' Поломки в мережі '
write(iprint,'(2x,3(4x,a),6x,a)')'N','i','j','mu(i,j)'
repeat{
   read(irdr3,*)ntt
   if(ntt==0) break
   nTr=nTr+1
   if(nTr>MaxTr) {
      write(iprint,*)' Занадто багато поломок в мережі '
      i1=MaxTr
      write(iprint,*)' Всього ',nTr,' MaxTr=',i1
      stop
   }
   Fizi=.true.
   for(i=1; i<=ntt; i=i+1){
```

```
      nelms1=nelms1+1
      if(nelms1>MaxArcTr) {
         write(iprint,*)' Занадто багато дуг в масиві поломок'
         i1=MaxArcTr
         write(iprint,*)' Всього ',nelms1,' MaxTr=',i1
         stop
      }
      read(irdr3,*)iv1,jv1,arcmut(nelms1)
      nTr1=nTr-1
      if(Fizi) {
        write(iprint,'(2x,3i5,2x,f10.2)')nTr1,iv1,jv1,arcmut(nelms1)
        Fizi=.false.
      }
      else {
        write(iprint,'(7x,2i5,2x,f10.2)')iv1,jv1,arcmut(nelms1)
      }
      narct(nelms1)=0
      for(j=1; j<=nArc; j=j+1){
         if(iv(j)!=iv1) next
         if(jv(j)!=jv1) next
         narct(nelms1)=j
         break
      }
      if(narct(nelms1)!=0 & arcmut(nelms1)>=0.d0 &
         arcmut(nelms1)<=1.d0 ) next
      write(iprint,*) 'Error: Невірно вказана остання поломка'
      stop
   }
   nt(nTr+1)=nelms1+1
}
return
end

# Підпрограма CoordA, r-алгоритм вирішує координуючу
# задачу мінімізації негладкої опуклої функції
# для схеми декомпозиції за змінними
# Вхідні параметри:
# n - розмірність простору змінних
# h0 - початковий крок
# epsx - критерій зупинки по аргументу
# x(n) - початкова точка
# Вихідні параметри:
```

```
# itn - число витрачених ітерацій
# istop - код зупинки (2-по epsg, 3-по epsx,
# 4-по числу ітерацій, 5 - не знайдений мінімум у напрямку)
# xr (n) - знайдена точка мінімуму функції
# fr - значення функції в точці мінімуму
subroutine CoordA(n,x,h0,epsx,intp,fr,xr,itn,istop)
implicit real*8(a-h,o-z),integer*2(i-n)
dimension x(n),xr(n)
# Робочі масиви: b(n,n) - матриця оберненого перетворення
# простору
# g(n),g1(n),g2(n) – проміжні вектори
dimension b(MaxArc,MaxArc),g(MaxArc),g1(MaxArc),g2(MaxArc)
# Установка нуля і початкових параметрів
dzero=1.d-20; w=1./alp-1.
hs=h0
itn=0; lp=itn+intp
do i=1,n{
   do j=1,n
      b(j,i)=0.d0
   b(i,i)=1.d0
   }
call FGCoord(x,n,f,g)
fr=f
do i=1,n{
   g1(i)=g(i); xr(i)=x(i)
}
nls=0; nlsa=0
write(iprint,3000)
write(iprint,3100) itn,f,fr,nlsa,nls
# Основне тіло програми, що реалізує алгоритм
for(itn=1; itn<=maxitn; itn=itn+1){
# Критерій зупинки по нормі градієнта
   dg=0.d0
   do i=1,n
      dg=dg+g(i)*g(i)
   istop=2
   if(dsqrt(dg)<=epsg) break
# Обчислити субградієнт в перетвореному просторі
   do i=1,n{
      d=0.d0
      do j=1,n
         d=d+b(j,i)*g(j)
```

```
      g2(i)=d
   }
# Обчислити і нормувати різницю субградієнтів
   dg=0.d0
   do i=1,n{
      g(i)=g2(i)-g1(i)
      dg=dg+g(i)*g(i)
   }
   dg=dsqrt(dg)
   if(dg>dzero) {
     dg=1.d0/dg
     do i=1,n
        g(i)=dg*g(i)
   }
# Обчислити нормований субградієнт в перетвореному просторі
   d=0.d0
   do i=1,n
      d=d+g(i)*g2(i)
   d=w*d
   d1=0.d0
   do i=1,n{
      g1(i)=g2(i)+d*g(i)
      d1=d1+g1(i)**2
   }
   d1=1./dsqrt(d1)
   do i=1,n
      g2(i)=d1*g1(i)
# Перерахувати матрицю B
   do i=1,n{
      d=0.d0
      do j=1,n
         d=d+b(i,j)*g(j)
      d=w*d
      do j=1,n
         b(i,j)=b(i,j)+d*g(j)
   }
# Обчислити напрямок руху
   dg=0.d0
   do i=1,n{
      d=0.d0
      do j=1,n
         d=d+b(i,j)*g2(j)
```

```
      g(i)=d
      dg=dg+d*d
   }
   dg=dsqrt(dg)
# Одновимірний спуск у напрямку
   ls=0; dx=0.d0; istop=5
   20 continue
     ls=ls+1
     dx=dx+hs*dg
     do i=1,n
        x(i)=x(i)-hs*g(i)
     call FGCoord(x,n,f,g2)
     if(f<fr) {
       fr=f
       do i=1,n
          xr(i)=dabs(x(i))
     }
     d=0.d0
     do i=1,n
        d=d+g(i)*g2(i)
     if(ls>500) break  # Останов: istop = 5
     if(ls>nh) hs=hs*q2
   if(d>0.d0) go to 20
   nls=nls+ls; nlsa=nlsa+ls
   if(itn==lp) {
     write(iprint,3100) itn,f,fr,nlsa,nls
      lp=lp+intp; nlsa=0
   }
   do i=1,n
      g(i)=g2(i)
   istop=3
   sum=0.d0
   do i=1,n
      sum=sum+x(i)*x(i)+1.d0
   sum=dsqrt(sum)
   if(dabs(dx/sum)<epsx) break
   istop=4
}
if(istop==4) return
if(itn==lp) write(iprint,3100) itn,f,fr,nlsa,nls
return
3000 format(/10x,' Протокол процесу негладкої оптимізації '/2x,
```

```
            '  Itn.',7x,'..f(x)..',13x,'...f(x_r)...',
            5x,'LS',4x,'LSa')
3100 format(2x,i5,2(2x,1pd18.10),3(2x,i5))
end
# Програма FGCoord - обчислює значення функції і
# субградієнта для негладкої функції яка відповідає
# декомпозиції за змінними
subroutine FGCoord(y,ny,fy,gy)
implicit real*8 (a-h,o-z),integer*2(i-n)
# Загальні блоки даних
common/NetVA/nVert,nArc,iv(MaxArc),jv(MaxArc),
            cost(MaxArc),y0(MaxArc),yup(MaxArc)
common/Troubl/nTr,nt(MaxTr1),narct(MaxArcTr),arcmut(MaxArcTr)
dimension y(ny),gy(ny)
# Робочі масиви
dimension xmu1(MaxArc),y1(MaxArc),u(MaxArc),ur(MaxArc)
fy=0.d0
do i=1,ny{
   gy(i)=cost(i)
   fy=fy+cost(i)*dabs(y(i))
}
nu=nArc;
for(it=1;it<=nTr;it=it+1){
   do i=1,nu
      xmu1(i)=1.d0
   ind=nt(it); ist=nt(it+1)-1;
   for(i=ind; i<=ist; i=i+1){
      xmu1(narct(i))=arcmut(i)
   }
   do i=1,nu{
      u(i)=0.d0  # старт з нульової точки
      # значення правої частини в підзадачі
      y1(i)=xmu1(i)*(y0(i)+dabs(y(i)))
   }
   # Параметри внутрішнього r-алгоритму
   h0=1.d0; epsu=1.d-5; intp=-1;
   call DualA(nu,u,h0,epsu,intp,fr,ur,itn,istop,y1)
   if(istop>3) {
      write (iprint,*)' *** Warning: Не розвязано внутрішню підзадачу'
      write (iprint,*)'   t =',it, '    istop= ',istop
   }
   fy=fy+fr
```

```
 # write (iprint,*)'   t =',it, '    fr= ',fr
   do i=1,nu
      gy(i)=gy(i)-xmu1(i)*ur(i)
}
do i=1,nu
   # урахування заміни на модуль
   if(y(i)<0.d0) gy(i)=-gy(i)
return
end
# Підпрограма DualA, r-алгоритм максимізує увігнуту
# негладку функцію для схеми декомпозиції за обмеженнями
# Вхідні параметри:
# n - розмірність простору змінних
# h0 - початковий крок
# epsx - критерій зупинки по аргументу
# x(n) - початкова точка
# Вихідні параметри:
# itn - число витрачених ітерацій
# istop - код зупинки (2-по epsg, 3-по epsx, 4-по числу ітерацій,
# 5 - не знайдений мінімум у напрямку, 10 - по несумісності)
# xr (n) - знайдена точка максимуму функції
# fr - значення функції в точці максимуму
subroutine DualA(n,x,h0,epsx,intp,fr,xr,itn,istop,y)
implicit real*8(a-h,o-z),integer*2(i-n)
dimension x(n),xr(n),y(n)
# Робочі масиви:
# b(n,n) - матриця оберненого перетворення простору
# g(n),g1(n),g2(n) - зберігання проміжних векторів
dimension b(MaxArc,MaxArc),g(MaxArc),g1(MaxArc),g2(MaxArc)
# Установка нуля і початкових параметрів
dzero=1.d-20; w=1./alp-1.
hs=h0
itn=0; lp=itn+intp
do i=1,n{
   do j=1,n
      b(j,i)=0.d0
   b(i,i)=1.d0
   }
call FGDual(x,n,f,g,y)
fr=f
do i=1,n{
   g1(i)=g(i); xr(i)=dabs(x(i))
```

```
}
nls=0; nlsa=0
if(intp>0) {
  write(iprint,3000)
  write(iprint,3100) itn,f,fr,nlsa,nls
  }
# Основне тіло програми, що реалізує алгоритм
for(itn=1; itn<=maxitn; itn=itn+1){
# Критерій зупинки по нормі градієнта
   dg=0.d0
   do i=1,n
      dg=dg+g(i)*g(i)
   istop=2
   if(dsqrt(dg)<=epsg) break
# Обчислити субградієнт в перетвореному просторі
   do i=1,n{
      d=0.d0
      do j=1,n
         d=d+b(j,i)*g(j)
      g2(i)=d
   }
# Обчислити і нормувати різницю субградієнтов
   dg=0.d0
   do i=1,n{
      g(i)=g2(i)-g1(i)
      dg=dg+g(i)*g(i)
   }
   dg=dsqrt(dg)
   if(dg>dzero) {
     dg=1.d0/dg
     do i=1,n
        g(i)=dg*g(i)
   }
# Обчислити нормований субградієнт в перетвореному просторі
   d=0.d0
   do i=1,n
      d=d+g(i)*g2(i)
   d=w*d
   d1=0.d0
   do i=1,n{
      g1(i)=g2(i)+d*g(i)
      d1=d1+g1(i)**2
```

```
   }
   d1=1./dsqrt(d1)
   do i=1,n
      g2(i)=d1*g1(i)
# Перерахувати матрицю B
   do i=1,n{
      d=0.d0
      do j=1,n
         d=d+b(i,j)*g(j)
      d=w*d
      do j=1,n
         b(i,j)=b(i,j)+d*g(j)
   }
# Обчислити напрямок руху
   dg=0.d0
   do i=1,n{
      d=0.d0
      do j=1,n
         d=d+b(i,j)*g2(j)
      g(i)=d
      dg=dg+d*d
   }
   dg=dsqrt(dg)
# Одновимірний спуск у напрямку
   ls=0; dx=0.d0; istop=5
   20 continue
     ls=ls+1
     dx=dx+hs*dg
     do i=1,n
        x(i)=x(i)+hs*g(i)
     call FGDual(x,n,f,g2,y)
     if(f>fr) {
       fr=f
       do i=1,n
          xr(i)=dabs(x(i))
     }
     d=0.d0
     do i=1,n
        d=d+g(i)*g2(i)
     if(ls>500) break  # Останов: istop = 5
     if(ls>nh) hs=hs*q2
   if(d>0.d0) go to 20
```

```
   nls=nls+ls; nlsa=nlsa+ls
   if(itn==lp & intp>0) {
     write(iprint,3100) itn,f,fr,nlsa,nls
      lp=lp+intp; nlsa=0
   }
   do i=1,n
      g(i)=g2(i)
   istop=3
   sum=0.d0
   do i=1,n
      sum=sum+x(i)*x(i)+1.d0
   sum=dsqrt(sum)
   if(dabs(dx/sum)<epsx) break
   istop=4
}
if(istop==4) return
if(intp>0) write(iprint,3100) itn,f,fr,nlsa,nls
return
3000 format(/10x,' Протокол процесу негладкої оптимізації '/2x,
             '  Itn.',7x,'..f(x)..',13x,'...f(x_r)...',
             5x,'LS',4x,'LSa')
3100 format(2x,i5,2(2x,1pd18.10),3(2x,i5))
end
# Підпрограма FGDual обчислює функцію і субградієнт для
# Dual
subroutine FGDual(u,nu,fu,g,y1)
implicit real*8 (a-h,o-z),integer*2(i-n)
# Загальні блоки даних
common/NetVA/nVert,nArc,iv(MaxArc),jv(MaxArc),
             cost(MaxArc),y0(MaxArc),yup(MaxArc)
common/Demand/is(MaxVert1),isr(MaxDemand),dsr(MaxDemand)
common/Data/Penalt,Zup
# Вхідні параметри:
# nu - кількість множників Лагранжа (дорівнює числу дуг)
# u(nu) - вектор множників Лагранжа
# y1(nu) - вектор правих частин зв'язуючих обмежень
# Вихідні параметри:
# fu - значення функції координуючої задачі,
# g(nu) - субградієнт координуючої задачі,
dimension u(nu),g(nu),y1(nu)
# Робочі масиви
# cu(MaxArc) - вектор множників Лагранжа, u_i = dabs(z_i)
```

```
# pot(MaxVert) - вектор потенціалів для знаходження найкоротшого шляху
# ipred(MaxVert) - вектор посилань на дуги для розшифровки найкоротших шляхів
# використовується при зупинці алгоритму найкоротших шляхів
dimension cu(MaxArc),pot(MaxVert),ipred(MaxVert)
logical fizi
# Присвоїти початкові значення функції і субградієнту з
# урахуванням заміни на модуль, що відповідає
# cu(i) = dabs(u(i))
fu=0.d0;
do i=1,nu{
   cu(i)=dabs(u(i)) # запамятати заміну на модуль
   g(i)=-y1(i)      # урахування вектора правих частин
   fu=fu+g(i)*cu(i)
}
# константи для найкоротших шляхів
dmax=1.d10; dmax1=dmax-1.d0; deps=1.d-8;
for(i=1; i<=nVert; i=i+1) {
   # покажчики кому постачати з вершини i
   ind=is(i); ist=is(i+1)-1
   # якщо нікому, то нічого не робити
   if(ind>ist) next
   for(j=1; j<=nVert; j=j+1){    # ініціалізація масивів,
      pot(j)=dmax; ipred(j)=-1;  # потенціалів і посилань
      }
   # відкрити початок (вершина i)
   pot(i)=0.d0; ipred(i)=0;
   # цикл перерахунку потенціалів
   repeat{
      # для алгоритму найкоротших шляхів
      fizi=.true.
      for(j=1; j<=narc; j=j+1){
         if(pot(iv(j))>dmax1) next
         a1=pot(iv(j))+cu(j)
         if(a1<pot(jv(j))-deps) {
           # запамятати найкращий потенціал
           pot(jv(j))=a1;
           # і номер дуги, звідки він прийшов
           ipred(jv(j))=j;
           fizi=.false.
         }
      }
      # Зупинка, якщо нічого покращувати
```

```
      if(fizi) break
   }
   for(j=ind; j<=ist; j=j+1){
      i1=isr(j);
      if(ipred(i1)==-1) {
        write(iprint,*) ' Error 1: не знайдено найкоротший шлях з '
        write(iprint,*) '          вершини',i,' до вершини',i1
        stop
      }
      fu=fu+pot(i1)*dsr(j)   # скоригувати fu
      repeat {               # скоригувати субградієнт
         i0=ipred(i1)
         g(i0)=g(i0)+dsr(j)
         i1=iv(i0)
         # розшифровку найкоротшого шляху закінчено
         if(i1==i) break
      }
   }
}
# Лінійний штраф (\sum_{i,j}C_ij}+1)*Z_ijt,
# 0 <= z_ijt <= \sum_nd dem(i)
for(i=1; i<=nu; i=i+1) {
   zij=Penalt-cu(i)
   if(zij>=0.d0) next
   fu=fu+zij*Zup
   g(i)=g(i)-Zup
}
for(i=1; i<=nu; i=i+1) {
   # урахування неактивних обмежень
   if(cu(i)<1.d-8 & g(i)<1.d-8) g(i)=0.d0
   # урахування заміни змінних на їх модуль
   if(u(i)<0.d0) g(i)=-g(i)
}
return
end
subroutine prtime(icode)
common /time/ih0,im0,is0,iss0
integer*2 ih0,im0,is0,iss0
integer*2 ih,im,is,iss
integer*2 i1,i2,i3,i4
integer values(8)
if(icode==0){
```

```
  call date_and_time(VALUES=values)
  ih0 = values(5)
  im0 = values(6)
  is0 = values(7)
  iss0 = values(8)
  return
  }
call date_and_time(VALUES=values)
ih = values(5)
im = values(6)
is = values(7)
iss = values(8)
i1=ih-ih0
if(im<im0) {
  im=im+60
  i1=i1-1
  }
i2=im-im0
if(is<is0) {
  is=is+60
  i2=i2-1
  }
i3=is-is0
if(iss<iss0) {
  iss=iss+1000
  i3=i3-1
  }
i4=iss-iss0
write (*,'(a,3(i2,1h.),i3)')' time ',i1,i2,i3,i4
return
end


#############################################################
# Вхідні файли для тестового прикладу (див. підрозділ 4.5)
#############################################################
# fort.1
#############################################################
    6    16
    1    2      1.50     0.0
    1    3      1.00     0.0
    1    4      1.00     0.0
```

```
    2    4       1.00      0.0
    2    5       1.00      0.0
    3    4       1.00      0.0
    4    5       1.00      0.0
    3    6       1.00      0.0
    2    1       1.50      0.0
    3    1       1.00      0.0
    4    1       1.00      0.0
    4    2       1.00      0.0
    5    2       1.00      0.0
    4    3       1.00      0.0
    5    4       1.00      0.0
    6    3       1.00      0.0

############################################################
# fort.2
############################################################
    30
    1    2        10.00
    2    1        10.00
    1    3        10.00
    3    1        10.00
    1    4        10.00
    4    1        10.00
    1    5        10.00
    5    1        10.00
    1    6        10.00
    6    1        10.00
    2    3        10.00
    3    2        10.00
    2    4        10.00
    4    2        10.00
    2    5        10.00
    5    2        10.00
    2    6        10.00
    6    2        10.00
    3    4        10.00
    4    3        10.00
    3    5        10.00
    5    3        10.00
    3    6        10.00
    6    3        10.00
```

```
    4     5        10.00
    5     4        10.00
    4     6        10.00
    6     4        10.00
    5     6        10.00
    6     5        10.00

#############################################################
# fort.3 без поломок a)
#############################################################
    0

#############################################################
# fort.3 поломка b)
#############################################################
   1
   1     2    0.d0
   1
   1     3    0.d0
   1
   1     4    0.d0
   1
   2     4    0.d0
   1
   2     5    0.d0
   1
   3     4    0.d0
   1
   4     5    0.d0
   1
   2     1    0.d0
   1
   3     1    0.d0
   1
   4     1    0.d0
   1
   4     2    0.d0
   1
   5     2    0.d0
   1
   4     3    0.d0
   1
```

```
   5    4   0.d0
  0

#################################################################
# fort.3 поломка c)
#################################################################
   4
   1    2   0.d0
   2    1   0.d0
   2    4   0.d0
   4    2   0.d0
   0

#################################################################
# fort.3 поломка f)
#################################################################
   1
   1    2   0.5d0
   1
   1    3   0.5d0
   1
   1    4   0.5d0
   1
   2    4   0.5d0
   1
   2    5   0.5d0
   1
   3    4   0.5d0
   1
   4    5   0.5d0
   1
   2    1   0.5d0
   1
   3    1   0.5d0
   1
   4    1   0.5d0
   1
   4    2   0.5d0
   1
   5    2   0.5d0
   1
   4    3   0.5d0
```

```
1
5     4    0.5d0
1
6     3    0.5d0
1
3     6    0.5d0
0
```

ДОДАТОК Б

СПИСОК ПУБЛІКАЦІЙ ЗА ТЕМОЮ ДИСЕРТАЦІЇ ТА ВІДОМОСТІ ПРО АПРОБАЦІЮ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЇ

Публікації, в яких опубліковано
основні наукові результати дисертації:

1. Стецюк, Петро, Володимир Ляшко, та Антон Супрун. 2020. «Метод BFGS для задачі побудови S-подібної кривої». *Наукові записки НаУКМА*, *Комп'ютерні науки* 3:102–106.
2. Стецюк, Петро, Олексій Лиховид, та Антон Супрун. 2020. «Про лінійну та квадратичну двоетапні транспортні задачі». *Кібернетика та комп'ютерні технології* 4:5–14.
3. Стецюк, Петро, Олексій Лиховид, Володимир Жидков, та Антон Супрун. 2021. «Оптимізаційні задачі модернізації пропускних здатностей дуг відмовостійких мереж». *Проблеми керування та інформатики* 5:5–20.
4. Стецюк, Петр, Виктор Стовба, и Антон Супрун. 2021. «B-форма метода Давидона–Флетчера–Пауэлла». *Журнал обчислювальної та прикладної математики* 2(136):93–110.
5. Stetsyuk, Petro, Olha Khomiak, Yehor Blokhin, and Anton Suprun. 2022. "Optimization Problems for the Maximum k-Plex". *Cybernetics and Systems Analysis* 58(4):46–58.
6. Стецюк, Петро, Олексій Лиховид, Володимир Жидков, та Антон Супрун. 2023. Оптимізаційні задачі модернізації пропускних спроможностей дуг відмовостійких мереж. У *Методи негладкої оптимізації в прикладних задачах*, 11–34. Київ: ЛАЗУРИТ ПОЛІГРАФ.
7. Стецюк, Петро, Ольга Хом'як, Олександр Жмуд, та Антон Супрун. 2023. Двоетапна транспортна задача з обмеженням на кількість проміжних пунктів. У *Методи негладкої оптимізації в прикладних задачах*, 68–81. Київ: ЛАЗУРИТ ПОЛІГРАФ.
8. Стецюк, Петро, Ольга Хом'як, та Антон Супрун. 2023. Оптимізаційні задачі для максимального k-плекса. У *Методи негладкої оптимізації в прикладних задачах*, 206–229. Київ: ЛАЗУРИТ ПОЛІГРАФ.

9. Супрун, Антон. 2023. «Обчислювальні експерименти для $r(\alpha)$-алгоритму з прискореною реалізацією розтягу простору». *Кібернетика та комп'ютерні технології* 2:46–54.

Публікації, що засвідчують апробацію матеріалів дисертації

1. Suprun, Anton. 2020. "Computational aspects of quantile regression". Тези доповідей XVIII міжнародної науково-практичної конференції «Математичне та програмне забезпечення інтелектуальних систем (MPZIS-2020)», Дніпро, Листопад 18–20.
2. Стецюк, Петро, та Антон Супрун. 2021. "Прискорення GUROBI та CPLEX для задачі комівояжера". Міжнародний сателітний симпозіум "Інтелектуальні рішення - 2021-С", Київ, Вересень 29.
3. Супрун, Антон, та Андрій Івлічев. 2021. "Інтерактивна програма для задачі побудови та аналізу плоскої кривої з квадратичною кривиною". Міжнародний сателітний симпозіум "Інтелектуальні рішення - 2021-С", Київ, Вересень 29.
4. Супрун, Антон. 2021. "Модифікація r-алгоритму для задачі квантильної регресії". Тези доповідей XIX міжнародної науково-практичної конференції «Математичне та програмне забезпечення інтелектуальних систем (MPZIS-2020)», Дніпро, Листопад 18–20.

Відомості про апробацію результатів дисертації

Результати дисертації доповідались та обговорювались на:

- XVIII міжнародній науково-практичній конференції «Математичне та програмне забезпечення інтелектуальних систем (MPZIS-2020)», 18–20 листопада, 2020, м. Дніпро, Україна;

- Міжнародному сателітному симпозіумі «Інтелектуальні рішення – 2021-С», 29 вересня, 2021, м. Київ, Україна;
- фаховому семінарі відділу методів негладкої оптимізації Інституту кібернетики імені В.М. Глушкова НАН України (24 жовтня 2023 року).