Міністерство освіти і науки, молоді та спорту України
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

Мельник Тарас Петрович

ОПТИМІЗАЦІЯ СТРУКТУРИ КОМП'ЮТЕРНОЇ МЕРЕЖІ З ВИКОРИСТАННЯМ МЕТОДУ ПОСЛІДОВНИХ ПОСТУПОК

Спеціальність 8.05010201 – Комп'ютерні системи та мережі
Дипломна робота за освітньо-кваліфікаційним рівнем «магістр»

Студент групи КСМм - 51
Мельник Т. П.

_____
підпис
Науковий керівник
к.т.н., проф. Теслюк В. М.

_____
підпис

Нормоконтролер

_____
Прізвище, ініціали Підпис

Дипломну роботу допущено до захисту
«___»_____ 2012 р.
Зав. кафедри КІ
Березький О. М. _____
Підпис

Тернопіль – 2012 р.

РЕФЕРАТ

Дипломна робота на тему «Оптимізація структури комп'ютерної мережі з використанням методу послідовних поступок» на здобуття освітньо-кваліфікаційного рівня «Магістр» зі спеціальності «Комп'ютерні системи та мережі» написана обсягом 100 сторінок і містить 15 ілюстрацій, 2 таблиці, 2 додатки та 22 джерела.

Метою роботи є аналіз комп'ютерних мереж та методів багатокритеріальної оптимізації, та розробка моделі оптимізації структури комп'ютерної мереежі.

Методи досліджень базуються на основних положенях теорії оптимізації, методі послідовних поступок та критеріях комп'ютерних мереж.

Розроблено структуру підсистеми розв'язку задач багатокритеріальної оптимізації методом послідовних поступок, яка включає підсистему вводу вхідних даних, підсистему розрахунків, підсистему виведення даних та інтерфейс.

Побудована модель оптимізації структури компютерної мережі, яка може включати такі критерії як вартість, надійність, пропускна здатність та інтенсивність відмов з відповідними обмеженнями.

Описано розлоблене програмне та технічне забезпечення підсистеми для розв'язання задач багатокритеріальної оптимізації.

Наведені результати розв'язку тестових задач багатокритеріальної оптимізації методом послідовних поступок.

Описані особливості інтерфейсу підсистеми розв'язку задач багатокритеріальної оптимізації та алгоритм їх рішення з допомогою розробленої підсистеми.

Результати роботи можуть бути використані при проектуванні та оптимізації комп'ютерної мережі.

Можливими напрямками подальших досліджень є продовження робіт по аналізу та удосконаленню багатокритеріальної оптимізації структури мережі.

Ключові слова: ОПТИМІЗАЦІЯ, МЕРЕЖА, СТРУКТУРА, МЕТОД ПОСЛІДОВНИХ ПОСТУПОК, МОДЕЛЬ ОПТИМІЗАЦІЇ, КРИТЕРІЇ.

ABSTRACT

Thesis: "Optimization of computer network using the method of successive concessions" on the education and qualification of "Master" specialty "Computer systems and networks" written volume contains 100 pages and 15 figures, 2 tables, 2 applications and 22 sources.

The aim is the analysis of computer networks and methods of multiobjective optimization and development of model structure optimization computer mereezhi.

Methods based on the main provisions of optimization theory, the method of successive concessions and criteria for computer networks.

The structure subsystem problem solving multiobjective optimization method of successive concessions, including subsystem input input data, calculations subsystem, subsystem and output interface.

The model structure optimization of computer network that may include such criteria as cost, reliability, capacity and failure rate with appropriate restrictions.

Described rozloblene software and technical support for subsystems for solving multiobjective optimization problems.

The results of test problem solving multiobjective optimization method of successive concessions.

We describe the features of the interface subsystem problem solving multiobjective optimization algorithm and its solution using the developed subsystems.

The results may be used in the design and optimization of computer network.

Possible directions for further research is continuing work on analysis and improvement of the multicriterion optimization of network structure.

Keywords: OPTIMIZATION, NETWORK, STRUCTURE, METHOD OF SUCCESSIVE CONCESSIONS, OPTIMIZATION MODEL, CRITERIA.

ЗМІСТ

ВСТУП

Методи розв’язання задач математичного програмування з одним критерієм інтенсивно розроблялися останні 40 років. Сучасні задачі математичного програмування набули бурхливого розвитку з появою персональних комп’ютерів (ПК). З розвитком епохи інформаційних технологій, виникла необхідність розв’язувати задачі, які характеризується більш ніж одним критерієм. Керівники підприємств, набагато більше, ніж раніше, відчувають необхідність оцінювати альтернативні рішення з погляду декількох критеріїв.

Результати дослідження задач планування і управління показують, що в реальній постановці ці задачі є багатокритерійними. Так, вираз, що часто зустрічається в житті “досягти максимального ефекту при найменших витратах” вже по суті означає ухвалення рішення за двома критеріями. Оцінка діяльності підприємств і планування, як системи прийняття рішень, проводиться на основі більше десяти критеріїв: виконання плану виробництва за об'ємом, по номенклатурі, плану реалізації, оцінки за показниками рентабельності, продуктивності праці, тощо.

Раніше, при дослідженні проблеми багатокритеріальності, часто всі критерії, окрім одного, вибраного домінуючим, приймалися як обмеження і оптимізація проводилася по домінуючому критерію. Такий підхід до рішення практичних задач значно знижує ефективність ухвалюваних рішень.

Вперше проблема оптимізації векторного критерію була сформульована економістом Парето в 1896 р. Окрім того, на сьогодні, досить часто використовуються і інші методи, такі як методи цільового програмування та метод поступок. Застосування методів багатокритеріальної оптимізації дає змогу суттєво покращити вихідні параметри об’єктів розроблення, економити матеріальні та людські ресурси. Тому тема магістерської роботи “Оптимізація структури комп’ютерної мережі з використанням методу поступок” є актуальною. Відповідно об’єктом дослідження є процеси розроблення комп’ютерних мереж, а предметом дослідження – оптимізаційна модель та засоби розв’язання задач оптимізації з використанням методу поступок.

СПИСОК УМОВНИХ СКОРОЧЕНЬ

ПЗ	Програмне забезпечення
ТЗ	Технічні засоби
ООП	Об’єктно-орієнтоване програмування
САПР	Система автоматизованого проектування
ПК	Персональний комп’ютер
ГА	Генетичні алгоритми
МДР	Множина допустимих рішень
ЛП	Лінійне прграмування
БД	База даних

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА РОЗРОБКА СТРУКТУРИ ПІДСИСТЕМИ АВТОМАТИЗАЦІЇ РОЗВ’ЯЗАННЯ ЗАДАЧ БАГАТОКРИТЕРІАЛЬНОЇ ОПТИМІЗАЦІЇ

Сучансна наука про об'єктивні причини функціонування і розвитку суспільства користується різноманітними кількісними характеристиками, а тому використовує велику кількість математичних методів. Наприклад, в економіці використовуються оптимізаційні методи, які складають основу математичного програмування, теорії ігор, мережевого планування, теорії масового обслуговування та інших прикладних наук. Багато задач прийняття рішень, що виникають на виробництві, в економіці та інших областях людської діяльності, можуть бути зведені до побудови відповідної математичної моделі, обчисленню цільової функції, яка оцінює процес функціонування системи, і знаходження її оптимального (для визначення можна вважати мінімального) значення. Як правило, побудовані цільові функції досить складні і можуть мати ряд особливостей, у зв’язку з чим їхня мінімізація зв’язана зі значними обчислювальними складностями. До цих особливостей у першу чергу необхідно віднести властивість багатоекстремальності.

Значні обчислювальні складності, пов’язані з мінімізацією багатоекстремальних та інших видів функцій стандартними методами, а також безумовна важливість цих класів задач для різних практичних додатків (задач оптимального вибору технічних, економічних, екологічних та інших систем) робить дуже актуальною проблему створення способу оптимізації, здатного ефективно вирішувати ці задачі. Розробка систем автоматизації розв’язання сучасних економічних задач дозволяє набути практичні навички в області проектування сучасних САПР та розширити знання в цій області.

1.1 Аналіз предметної області та її актуальність

Багато задач прийняття рішень, що виникають на виробництві, проектуванні комп’ютерних систем, в економіці та інших областях людської діяльності, можуть бути зведені до побудови відповідної математичної моделі, обчисленню цільової функції, яка оцінює процес функціонування системи, і знаходження її оптимального (для визначення можна вважати мінімального) значення.

Як правило, побудовані цільові функції досить складні і можуть мати ряд особливостей, у зв’язку з чим їхня мінімізація зв’язана зі значними обчислювальними складностями. До цих особливостей у першу чергу необхідно віднести властивість багатоекстремальності.

Значні обчислювальні складності, пов’язані з мінімізацією багатоекстремальних та інших видів функцій стандартними методами, а також безумовна важливість цих класів задач для різних практичних додатків (задач оптимального вибору технічних, економічних, екологічних та інших систем) робить дуже актуальною проблему створення способу оптимізації, здатного ефективно вирішувати ці задачі.

Алгоритми, що пропонуються для рішення практичних задач оптимізації, базуються на адаптивному, випадковому пошуку, створеному ще в 1967 р.

Роки їх практичного використання, а також проведеного статистичного дослідження, дозволили зробити висновок про безсумнівну достатню практичну ефективність і одночасно про економічність запропонованих способів оптимізації.

Терміном оптимізація в літературі позначають процес або послідовність операцій що дозволяють одержати уточнене рішення. Причому основне завдання в даному випадку полягає у виборі серед безліч можливих рішень такого, яке було б в певному значенні кращим або, як то кажуть оптимальним. В області вичислювальних методів розв'язку задач оптимального вибору в теперішній час вже отримані основні важливі теоретичні результати, розроблені методи, видані монографії та підручники. В той же час, залишаються задачі, дослідження в яких

не завершені, чи завершені порівняно недавно. Задачі лінійного програмування (ЗЛП) полягають в максимізації лінійної функції при лінійних обмеженнях. Теорія лінійного програмування містить як класичні, сформовані розділи (теорія двоїстості, алгоритм симплекс-метода, результат Хачияна про поліноміальний розв'язок задач лінійного програмування), так і розділи, які бурхливо розвиваються (алгоритми внутрішньої точки) і відкриті задачі (питання про існування поліноміальну модифікацію симплекс-метода). Теорія лінійного програмування розвивалась в роботах таких вчених, як Л.В.Канторович, Т.И.Купманс, Дж.Данциг, Ф.Л.Хічкок, Д.Б.Юдін, А.С.Немировский, Н.З.Шор, Л.Г.Хачиян. Одним із основних методів лінійного програмування, які широко використовуються і розвиваються, є симплекс-метод, запропонований Дж.Данцигом та метод внутрішньої точки, запропонований Н.Кармаркаром. Лінійне програмування зв'язано з класичною теорією лінійних нерівностей, яка розвинута в роботах Ж.Б.Фур'є, Дж.Фаркаша, Г.Мінковского, А.Штейниця, Г.Вейля, Т.С.Моцкіна і інших.

Значна увага в роботі приділена задачам оптимального вибору достатньо загального виду, включаючи багатокритеріальність. В теорії оптимізації для таких задач представлено два напрями: з одної сторони, дослідження по аналітичним умовах оптимальності і теорії двоїстості, з іншої - розробка теорії обчислювальних методів і побудова алгоритмів розв'язку екстремальних задач.

З проблем застосування методів багатокритеріальної (векторної) оптимізації в даний час існує багато публікацій. Розвиток і розширення масштабів використання методів векторної оптимізації було обумовлено вимогою практичної реалізації системного підходу до рішення задач в системі інвестиційного планування, а також необхідністю значного підвищення у результаті економічної ефективності виробництва і посиленням конкурентних позицій підприємства на цільовому ринку.

Багатоцільова оптимізація є одним з універсальних і ефективних методом для розв’язання проблем між критеріями в математичних моделях. Діалектика полягає в тому, що в більшості випадків недоцільно направляти зусилля лише на досягнення однієї мети, але необхідно прагнути до отримання досить хорошого

шуканого плану, результату, рішення що забезпечує досягнення деякої сукупності найбільш важливих цілей. Такий погляд на рішення проблеми знаходиться в повній відповідності з вимогою використання принципів системного підходу. Багатоцільовий підхід є необхідною методичною основою для подолання неточності та неповноти початкових даних, які виникають в сучасних задачах математичного програмування.

Найбільш відомими методами для вирішення задач багатокритеріальної оптимізації є:

- метод рівномірної оптимізації;
- метод справедливого компромісу;
- метод головного критерію;
- метод послідовних поступок;
- метод ідеальної точки;
- метод згортання критеріїв.

Можливими шляхами рішення проблем багатокритерійної оптимізації може бути застосування різних згорток і способів нормалізації. Також одним з можливих варіантів рішення задач багатокритерійної оптимізації є використання еволюційних (генетичних) алгоритмів.

Генетичні алгоритми (ГА) - адаптивні методи пошуку, які останнім часом часто використовуються для вирішення завдань функціональної оптимізації. Вони засновані на генетичних процесах біологічних організмів. Відзначимо, що ефективність ГА сильно залежить від таких деталей, як метод кодування рішень, оператори, настройки параметрів, приватний критерій успіху. Теоретична робота, відбита в літературі, присвяченій ГА, не дає підстав говорити об вироблення яких-небудь строгих механізмів для чітких прогнозів.

Останніми роками реалізовано багато генетичних алгоритмів і в більшості випадків вони мало схожі на класичний ГА. З цієї причини в теперішній час під терміном «генетичні алгоритми» розуміється не одна модель, а достатньо широкий клас алгоритмів, часом мало схожих один від одного. Дослідники експериментували з різними типами уявлень, операторів кросовера і мутації, спеціальних операторів, і різних підходів до відтворення і відбору. Рішення задачі

багатокритерійної оптимізації за допомогою ГА можна проводити по з тією особливістю, що визначення міри придатності хромосом можна визначати на основі якого-небудь методу багатокритерійної оптимізації (наприклад, метод справедливого компромісу) або принципу оптимальності або оптимізувати суперкритерій.

В теперішній час існують деякі проблеми багатокритеріальної оптимізації. Перша проблема пов'язана з вибором принципу оптимальності, який строго визначає властивості оптимального рішення і відповідає на питання, в якому сенсі оптимальне рішення перевершує всі інші допустимі рішення. На відміну від завдань однокритерійної оптимізації, у яких тільки один принцип оптимальності $f(x_o)>=f(x)$, в даному випадку є велика кількість різних принципів і кожен принцип може приводити до вибору різних оптимальних рішень. Це пояснюється тим, що доводиться порівнювати вектори ефективності на основі деякої схеми компромісу. В математичному відношенні ця проблема еквівалентна задачі упорядкування векторних множин, а вибір принципу оптимальності - вибору відношення порядку.

Друга проблема пов'язана з нормалізацією векторного критерію ефективності. Вона викликана тим, що дуже часто локальні критерії, що є компонентами вектора ефективності, мають різні масштаби вимірювання, що і утрудняє їх порівняння. Тому доводиться приводити критерії до єдиного масштабу вимірювання, тобто нормалізувати їх.

Третя проблема пов'язана з урахуванням пріоритету (або різного ступеня важливості) локальних критеріїв. Хоча при виборі рішення і слід добиватися найвищої якості по всіх критеріях, проте ступінь досконалості по кожному з них, як правило, має різну значимість. Тому звичайно для врахування пріоритету вводиться вектор розподілу важливості критеріїв, за допомогою якого коректується принцип оптимальності або проводиться диференціація масштабів вимірювання критеріїв.

До вищесказаного можна додати також те, що труднощі викликає одночасна наявність в сучасних задачах багатокритеріальної оптимізації якісних і кількісних критеріїв, а саме - перехід від якісних в кількісні критерії для подальшої

оптимізації математичної моделі. Крім того часом є складно правильно підібрати вагові коефіцієнти.

1.2 Основні елементи теорії оптимізації

Оптимізація – визначення экстремума розглянутої функції, тобто вибір найкращого варіанта з безлічі можливих. Терміном «оптимізація» в літературі позначають процес або послідовність операцій, що дозволяють отримати уточнене рішення. Хоча кінцевою метою оптимізації є відшукання якнайкращого або «оптимального» рішення, зазвичай доводиться задовольнятися покращенням відомих рішень, а не доведенням їх до досконалості. Тому під оптимізацією розуміють скоріше прагнення до досконалості, яка, можливо, і не буде досягнута. Завдання ухвалення рішення полягає у виборі серед множини можливих рішень (їх називають також варіантами, планами і т. п.) такого рішення, яке було б в певному значенні кращим або, як то кажуть, оптимальним. Зручно вважати, що вибір рішення проводить деяка особа, що приймає рішення, яке переслідує цілком певну мету. Залежно від конкретної ситуації в ролі особи, що приймає рішення, може виступати як окрема людина (інженер, науковий співробітник і т. п.), так і цілий колектив (група фахівців, зайнята рішенням однієї задачі). Кожне можливе рішення характеризується певним ступенем досягнення мети. Відповідно до цього у особи, що приймає рішення, є свої уявлення про переваги і недоліки рішень, на підставі якого одному рішенню, віддається перевага над іншим. Оптимальне рішення - це рішення, яке з погляду особи, що приймає рішення, переважно інших можливих рішень. Таким чином, поняття оптимального рішення пов'язане з перевагами особи, що ухвалює рішення. Ці переваги на практиці виражаються в різній формі, і їх математична формалізація може скласти складне завдання, оскільки особа, що ухвалює рішення, як правило, не може ясно і чітко сформулювати їх.

Мета теорії прийняття рішень і полягає в розробці методів, які допомогли б особі, що приймає рішення, якнайповніше і точно виразити свої переваги в

рамках відповідної математичної моделі і кінець кінцем обґрунтовано вибрати дійсно оптимальне рішення.

Перш ніж приступити до обговорення питань оптимізації, введемо ряд визначень.

Проектні параметри (шукані змінні). Цим терміном позначають незалежні змінні параметри, які повністю і однозначно визначають вирішувану задачу проектування.

Проектні параметри - невідомі величини, значення яких обчислюються в процесі оптимізації. Як проектні параметри можуть служити будь-які основні або похідні величини, що служать для кількісного опису системи. Так, це можуть бути невідомі значення довжини, маси, часу, температури. Число проектних параметрів характеризує ступінь складності даного завдання проектування. Звичайне число проектних параметрів позначають через п, а самі проектні параметри через х з відповідними індексами. Таким чином п проектних параметрів даного завдання позначатимемо через $x_1, x_2, x_3,..., x_n$.

Цільова функція (критерій якості). Це вираз, значення якого ОПР (особа, що приймає рішення) прагне зробити максимальним або мінімальним. Цільова функція дозволяє кількісно порівняти два альтернативні рішення. З математичної точки зору цільова функція описує деяку (n+1) -вимірну поверхню. Її значення визначається проектними параметрами

$$L = L(x_1, x_2,..., x_n).$$

Прикладами цільової функції, що часто зустрічаються в інженерній практиці, є вартість, вага, міцність, габарити, ККД. Якщо є тільки один проектний параметр, то цільову функцію можна представити з допомогою кривої на площині. Якщо проектних параметрів два, то цільова функція зображатиметься поверхнею в просторі трьох вимірювань. При трьох і більш проектних параметрах поверхні, що задаються цільовою функцією, називаються гіперповерхнями і не піддаються зображенню звичайними засобами. Топологічні властивості поверхні цільової функції відіграють велику роль в процесі оптимізації, оскільки від них залежить вибір найбільш ефективного алгоритму.

Цільова функція у ряді випадків може приймати найнесподіваніші форми.

Наприклад, її не завжди вдається виразити в замкнутій математичній формі, в інших випадках вона може бути шматково-гладкою функцією. Для задання цільовій функції іноді може знадобитися таблиця технічних даних (наприклад таблиця стану водяної пари) або може знадобитися провести експеримент. В ряді випадків проектні параметри приймають тільки цілі значення. Прикладом може служити число зубів в зубчатій передачі або число болтів у фланці. Іноді проектні параметри мають тільки два значення - так чи ні. Якісні параметри, такі як задоволення, яке випробовує покупець, що придбав виріб, надійність, естетичність, теж можливо враховувати в процесі оптимізації, хоча їх складно охарактеризувати кількісно. Проте в якому б вигляді не була представлена цільова функція, вона повинна бути однозначною функцією проектних параметрів.

В ряді завдань оптимізації потрібне введення більш однієї цільової функції. Іноді одна з них може виявитися несумісною з іншою. Прикладом служить проектування літаків, коли одночасно потрібно забезпечити максимальну міцність, мінімальну вагу і мінімальну вартість. В таких випадках конструктор повинен ввести систему пріоритетів і поставити у відповідність кожній цільовій функції деякий безрозмірний множник. В результаті з'являється «функція компромісу», що дозволяє в процесі оптимізації користуватися однією складеною цільовою функцією.

Пошук мінімуму і максимуму. Одні алгоритми оптимізації пристосовані для пошуку максимуму, інші - для пошуку мінімуму. Проте незалежно від типу вирішуваної задачі на екстремум можна користуватися одним і тим же алгоритмом, оскільки завдання мінімізації можна легко перетворити на завдання пошуку максимуму, помінявши знак цільової функції на зворотний.

Множина допустимих рішень (МДР) - простір рішення. Так називається область, визначена всіма п проектними параметрами. Простір рішення не такий великий, як може здаватися на перший погляд, оскільки вона зазвичай обмежена рядом умов, пов'язаних з фізичною суттю завдання. Обмеження можуть бути такими сильними, що завдання не матиме жодного задовільного рішення. Слід зазначити, що дуже часто у зв'язку з обмеженнями оптимальне значення цільової функції досягається на одній з меж області множини допустимих рішень задачі.

Локальний оптимум. Так називається точка простору рішень, в якій цільова функція має найбільше значення в порівнянні з її значеннями в усіх інших точках її найближчого околу.

Досить часто простір проектування містить багато локальних оптимумів і слід дотримуватися обережності, щоб не прийняти перший з них за оптимальне рішення задачі.

Глобальний оптимум. Глобальний оптимум - це оптимальне рішення для всієї множини допустимих рішень. Воно краще за всі інші рішення, відповідні локальному оптимуму, і саме його шукає ОПР. Можливий випадок декількох рівних глобальних оптимумів, розташованих в різних частинах простору проектування.

1.3 Призначення комп’ютерних мереж та їх різновиди

Появу комп'ютерних мереж можна розглядати як важливий крок у розвитку комп'ютерної техніки на шляху розширення її можливостей, а, отже, і на шляху розширення інтелектуальних можливостей людини у різних сферах діяльності. Вдосконалювання апаратури та програмних засобів досягло такого рівня, що встановити та експлуатувати комп’ютерну мережу може більш-менш освічений користувач. Об’єднувати комп'ютери в мережі почали близько 35 років тому.

Комп'ютерна мережа — це система розподіленого опрацювання інформації, що складається як мінімум із двох комп'ютерів, які взаємодіють між собою за допомогою засобів зв'язку.

Засоби зв'язку мають забезпечувати надійну передачу інформації між комп'ютерами мережі. Комп'ютери, що входять до складу мережі, виконують досить широке коло функцій, основними з яких є:

- організація доступу до мережі;
- керування передачею інформації;
- надання обчислювальних ресурсів і послуг абонентам мережі.

Абонент (вузол, хост, станція) — це пристрій, що підключений до мережі і бере активну участь в інформаційному обміні та має свою мережну адресу. Найчастіше абонентом (вузлом) мережі є комп'ютер, але абонентом також може бути, наприклад, мережний принтер або інший периферійний пристрій, що має можливість безпосередньо підключатися до мережі.

Сервером називається абонент (вузол) мережі, який надає свої ресурси іншим абонентам, але сам не використовує їх ресурси. Таким чином, він обслуговує мережу. Серверів в мережі може бути декілька, і зовсім не обов'язково, що сервер — найпотужніший комп'ютер. Виділений (dedicated) сервер — це сервер, що займається тільки мереженими завданнями. Невиділений сервер може крім обслуговування мережі виконувати і інші завдання. Специфічний тип сервера — це мережний принтер.

Клієнтом називається абонент мережі, який тільки використовує мережні ресурси, але сам свої ресурси в мережу не віддає, тобто мережа його обслуговує, а він нею тільки користується. Комп'ютер-клієнт також часто називають робочою станцією. В принципі кожен комп'ютер може бути одночасно як клієнтом, так і сервером.

Класифікація комп’ютерних мереж

Компютерні мережі можна класифікувати за територіальним призначенням, при цьому розрізняють глобальні, локальні та міські (рис. 1.1), та ще одним популярним способом класифікації мереж є їх класифікація за масштабом виробничого підрозділу.

Глобальні мережі охоплюють значні території, це може бути окрема держава, один або декілька континентів. Наприклад, мережа Internet охоплює всю земну кулю. Локальна мережа розміщується в рамках окремої організації або корпорації. Відмінності технологій локальних і глобальних мереж помітна, не дивлячись на їх постійне зближення. Наприклад, в глобальних мереж поважніша не якість зв’язку, а сам факт його існування.

Локальна обчислювальна мережа - Local Area Networks (LAN) – система, яка забезпечує на обмеженій тереторії один чи декілька каналів зв’язку, наданих приєднаним до неї абонентам для короткочасного монопольного користування.

Зазвичай до локальних мереж відносять мережі комп'ютерів, зосереджені на невеликій території (зазвичай в радіусі не більше 1-2 км), які об’єднують невелику кількість комп’ютерів. У загальному випадку локальна мережа є комунікаційною системою, що належить одній організації. Із-за коротких відстаней в локальних мережах є можливість використання дорогих високоякісних ліній зв'язку, які дозволяють, застосовуючи прості методи пересилання даних, досягати високих швидкостей обміну даними порядка 100 Мбіт/с. У зв'язку з цим послуги, що надаються локальними мережами, відрізняються широкою різноманітністю і зазвичай передбачають реалізацію в режимі on-line.

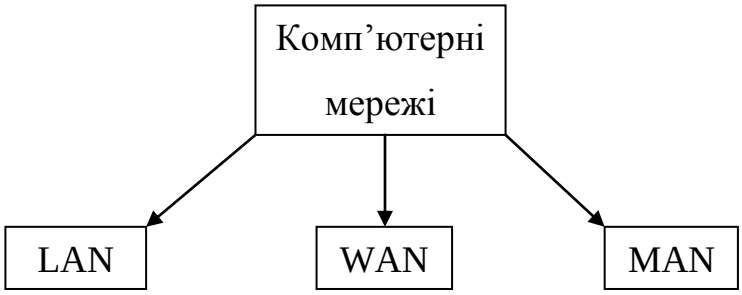

Рисунок 1.1 - Класифікація мереж за територіальною ознакою

Розглядаючи сучасні характеристики локальної мережі можна побачити, що вони об’єднують близько тисячі комп’ютерів на відстані декілька десятків кілометрів, та використовують різні середовища пересилання даних.

Локальна мережа дозволяє абоненту (користувачу) не помічати з’єднання, тобто забезпечує прозорое з’єднання. По суті, комп'ютери, зв'язані локальною мережею об'єднуються, в один віртуальний комп'ютер, ресурси якого можуть бути доступні всім користувачам, причому цей доступ не менш зручний, чим доступ до ресурсів, що входять безпосередньо в кожен окремий комп'ютер.

Головна відмінність ЛОМ від будь якої іншої – висока швидкість пересилання інформації (зараз зустрічається 1000 Мбіт/с), а також вірогідність інформації, що передається, визначається кількістю помилок на один знак. Цей показник становить $10^{-8} – 10^{-12}$ помилок/знак.

Глобальні мережі - Wide Area Networks (WAN) – об’єднують територіально розгалужені комп'ютери, які можуть знаходитися в різних містах і країнах. Оскільки прокладка високоякісних ліній зв'язку на великі відстані обходиться дуже дорого, в глобальних мережах часто використовуються вже існуючі лінії

зв'язки, спочатку призначені зовсім для інших цілей. Наприклад, багато глобальних мереж будуються на основі телефонних і телеграфних каналів загального призначення. Із-за низьких швидкостей таких ліній зв'язку в глобальних мережах (десятки кілобіт в секунду) набір послуг, що надаються, зазвичай обмежується передачею файлів, переважно не в оперативному, а у фоновому режимі. Завдяки використанню супутникових каналів звязку, сполучаються ЕОМ на відстані 15 тис. км один від одного.

Міські мережі (або мережі мегаполісів) - Metropolitan Area Networks (MAN) - є менш поширеним типом мереж. Ці мережі з'явилися порівняно недавно. Вони призначені для обслуговування території крупного міста - мегаполісу. Відстань між вузлами мережи становить 10 - 1000 км. Тоді як локальні мережі найкращим чином підходять для розділення ресурсів на коротких відстанях і широкомовних передач, а глобальні мережі забезпечують роботу на великих відстанях, але з обмеженою швидкістю і небагатим набором послуг, мережі мегаполісів займають деяке проміжне положення. Вони використовують цифрові магістральні лінії зв'язки, часто оптоволоконні, з швидкостями від 45 Мбіт/с, і призначені для зв'язку локальних мереж в масштабах міста і з'єднання локальних мереж з глобальними. Ці мережі спочатку були розроблені для передачі даних, але зараз вони підтримують і такі послуги, як відеоконференції і інтегральну передачу голосу і тексту.

За функціональним призначенням комп'ютерів, мережі прийнято поділяти на однорангові, мережі на основі серверів, гибридні.

В одноранговій мережі всі комп'ютери рівноправні, кожний з них може бути як клієнтом, так і сервером. При цьому ресурси кожного комп'ютера умовно поділяються на локальні і мережні. Локальними називаються власні ресурси кожного з комп'ютерів, незалежно від того підключений він до мережі чи ні. Мережними називається частина локальних ресурсів, які надає кожний комп'ютер в загальне користування іншим комп'ютерам. Якщо один з комп'ютерів мережі використовує ресурси іншого комп'ютера, то він виступає у ролі клієнта. Відповідно, процесор, що надає ресурси, вважається на цей момент сервером.

Одноранговa організація зазвичай використовується в невеликих мережах, що включають не більше 10 комп'ютерів.

Переваги одноранговых мереж**.** Одноранговi мережі мають цілий ряд переваг і особливо підходять для малих компаній, які не можуть дозволити собі великих витрат на дороге серверне устаткування і програмне забезпечення. Такі мережі:

- прості в інсталяції;
- не вимагають спеціальної посади адміністратора мережі;
- дозволяють користувачам самостійно управляти розділенням ресурсів;
- вартість створення невеликих мереж не дуже висока.

Недоліками одноранговых мереж є:

- має місце додаткове навантаження на комп'ютери із-за сумісного використання ресурсів.
- нездатність одноранговых вузлів обслуговувати, подібно до серверів, таке ж велике число з'єднань;
- відсутність централізованої організації, що ускладнює пошук даних;
- немає центрального місця зберігання файлів, що ускладнює їх архівацію;
- необхідність адміністрування користувачами власних комп'ютерів;
- слабка і незручна система захисту;
- відсутність централізованого управління, що ускладнює роботу з великими одноранговими мережами.

У мережах на основі серверів виділяються окремі комп'ютери для серверів і клієнтів. Для кожного виду мережних ресурсів створюється свій сервер, наприклад, файловий сервер, сервер преси, сервер бази даних тощо.

Серверні мережі і домени. Серверні середовища характеризуються наявністю в мережі серверів, що забезпечують захист мережі і її адміністрування.

Серверні мережі функціонують за наявності клієнтів. Клієнти звертаються до сервера, який надає їм різні засоби, наприклад друк або роботу з файлами. Клієнтські комп'ютери зазвичай менш могутні, чим машини в одноранговых мережах або сервери. На серверах функціонують такі ОС, як Window NT Server

або Novell NetWare. Клієнти використовують операційні системи типу MS-DOS або OS/2 2.0.

У Windows NT серверні мережі організовані в так звані домени. Домен - це сукупність мереж і клієнтів, що спільно використовують інформацію системи захисту. Захистом домена і повноваженнями на реєстрацію управляють спеціальні сервери - контроллери домена. У домені є один контроллер, званий основним (PDC, Primary Domain Controller), і допоміжні резервні контроллери (BDC, Backup Domain Controller), які виконують функції контроллера домена, коли PDC зайнятий або недоступний.

Жоден з комп'ютерів в мережі не зможе звертатися до ресурсів сервера, що розділяються, поки не пройде аутентифікацію на контроллері домена.

Серверні мережі мають такі переваги:

- потужний централізований захист;
- центральне сховище файлів, завдяки чому всі абоненти можуть працювати з одним набором даних, а резервне копіювання важливої інформації значно спрощується;
- оптимізовані виділені сервери функціонують в режимі розділення ресурсів швидше, ніж однорангові вузли;
- менш настирлива система захисту - доступ до ресурсів всієї мережі, що розділяються, - забезпечується по одному паролю;
- проста керованість при великій кількості абонентів;
- централізована організація, що запобігає втраті даних на комп'ютерах;

Серверним мережам властиві і деякі недоліки, які в основному відносяться до вартості серверного устаткування:

- дороге спеціалізоване апаратне забезпечення;
- дорогі серверні ОС і абонентські ліцензії;
- як правило, потрібний спеціальний адміністратор мережі.

У гібридних мережах більшість загальних ресурсів знаходяться на серверах, крім того, абоненти мають доступ до будь-яких ресурсів, визначених як подільні на комп'ютерах в робочій групі. Для доступу до ресурсів робочої групи, з якими

спільно працюють однорангові вузли мережі, абонентам необов'язково реєструватися на контроллері домена.

Гібридні мережі володіють перевагами як серверної моделі, так і однорангових мереж.

Гібридні обчислення страждають недоліками, характерними для серверних мереж.

Класифікації мереж за масштабом виробничого підрозділу, в межах якого діє мережа.

Розрізняють мережі відділів, мережі кампусів і корпоративні мережі.

Мережі відділів - це мережі, які використовуються порівняно невеликою групою співробітників (100 - 150 чоловік), що працюють в одному відділі підприємства (рис. 1.2). Головною метою мережі відділу є розділення локальних ресурсів, таких як додатки, дані, лазерні принтери і модеми.

Завдання управління мережею на рівні відділу відносно прості: додавання нових користувачів, усунення простих відмов, інсталяція нових вузлів і установка нових версій програмного забезпечення.

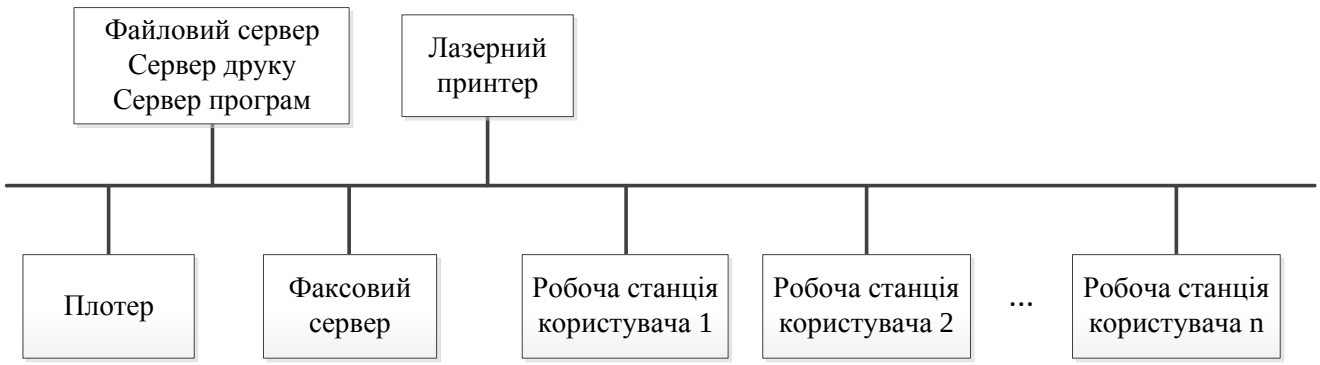

Рисунок 1.2 - Приклад мережі масштабу відділу

Мережі кампусів отримали свою назву від англійського слова campus - студентське містечко. Саме на території університетських городків часто виникала необхідність об'єднання декількох дрібних мереж в одну велику мережу. Зараз цю назву пов'язують не тільки із студентськими городками, але використовують і для позначення мереж будь-яких підприємств і організацій.

На рівні мережі кампусу виникають проблеми інтеграції неоднорідного апаратного і програмного забезпечення. Типи комп'ютерів, мережевих операційних систем, мережевого апаратного забезпечення можуть відрізнятися в

кожному відділі. Звідси витікають складнощі управління мережами кампусів. Адміністратори повинні бути в цьому випадку більш кваліфікованими, а засоби оперативного управління мережею - досконалішими.

Корпоративні мережі об'єднують велику кількість комп'ютерів на всіх територіях окремого підприємства. Для корпоративної мережі характерні:

- масштабність - тисячі призначених для користувача комп'ютерів, сотні серверів, величезні об'єми що зберігаються і передаваних по лініях зв'язку даних, безліч різноманітних застосувань;
- високий ступінь гетерогенності - типи комп'ютерів, комунікаційного устаткування, операційних систем і додатків різні;
- використання глобальних зв'язків - мережі філій з'єднуються за допомогою телекомунікаційних засобів, зокрема телефонних каналів, радіоканалів, супутникового зв'язку.

Призначення мережі

Перше призначення мереж - сумісне використання інформації. При цьому слід звернути увагу на те, яка інформація має життєво важливе значення для вашої організації? Які дані вимагають обмеження доступу або постійного доступу для всіх співробітників?

Сумісне використання апаратних засобів**.** Комп'ютери, не підключені до мережі, не дістають ефективного доступу до ресурсів, що розділяються. Наприклад, в невеликому офісі з 10 автономними комп'ютерами і одним принтером виводити інформацію на друк може тільки користувач, до ПК якого цей принтер приєднаний. Останнім доведеться записувати дані на дискету і передавати їх на комп'ютер з принтером. Така організація роботи заважатиме користувачеві ПК з принтером. Мережа дозволяє працювати з принтером всім підключеним до неї користувачам, а не тільки тому, до машини якого приєднаний пристрій друку.

Мережні комп'ютери можуть спільно працювати з наступними пристроями:

а) факс-модемами;

б) сканерами;

в) жорсткими дисками;

г) накопичувачами на гнучких дисках;

д) пристроями читання CD-ROM;

е) накопичувачами на магнітній стрічці для резервного копіювання даних;

ж) графічними пристроями;

з) з будь-якими іншими пристроями, що підключаються до комп'ютерів.

Сумісне використання програмних ресурсів. Інсталяція і настройка конфігурації програмного забезпечення в мережі значно скорочують об'єм роботи, потрібної для забезпечення доступу до комп'ютерних програм у всій організації.

Збереження інформації. Мережа дозволяє виконувати централізоване резервне копіювання інформації. Резервне копіювання - одна з найбільш важливих операцій, що входять в обов'язки адміністратора мережі. Комп'ютери - складні пристрої, і рано чи пізно користувачі стикаються з відмовами, які завжди трапляються в самий невідповідний момент. Мережеві компоненти також можуть виходити з ладу. Регулярне резервне копіювання значно полегшить життя вам і вашим користувачам.

Захист інформації**.** Мережа забезпечує важливій корпоративній інформації захищеніше середовище. При використанні автономних ПК доступ до комп'ютерів часто означає доступ до інформації, що знаходиться в них. Мережі реалізують додатковий рівень захисту за допомогою паролів. Кожному користувачеві, що працює в мережі, можна привласнити окреме облікове ім'я і пароль. В результаті мережний сервер знатиме, хто до нього звертається, і захистить інформацію, заборонивши несанкціоноване звернення до неї.

Електронна пошта**.** Однією з найбільш значних переваг, що отримуються користувачами від застосування мережі, є електронна пошта (e-mail). Замість обміну повідомленнями, директивами і зауваженнями на папері (що пов'язане з додатковими витратами і затримками) користувачі завжди можуть посилати один одному повідомлення і перевіряти їх отримання.

Будь-яка комп'ютерна мережа характеризується своєю архітектурою, яка визначається (рис. 1.3) її топологією, протоколами, інтерфейсами, мережними технічними і програмними засобами. Кожна із складових архітектури

комп'ютерної мережі характеризує її окремі властивості, і тільки їх сукупність характеризує всю мережу загалом.

Рисунок 1.3 - Компоненти архітектури комп'ютерної мережі

Топологія комп'ютерної мережі відображає структуру зв'язків між її основними елементами. Через низку причин існує відмінність між топологіями глобальних і локальних мереж. Топологія глобальних мереж характеризується достатньо складною, неоднорідною структурою. У свою чергу, топологія локальної мережі, зазвичай, має визначену структуру: лінійну, кільцеву або деревоподібну.

Протоколами називають правила взаємодії функціональних елементів мережі.

Інтерфейси — це засоби сполучення функціональних елементів мережі. Варто звернути увагу, що у ролі функціональних елементів можуть виступати як окремі пристрої, так і програмні модулі. Відповідно до цього, існують апаратні і програмні інтерфейси.

Під мережними технічними засобами мають на увазі різноманітні пристрої, що забезпечують об'єднання комп'ютерів в єдину комп'ютерну мережу. До цих пристроїв відносяться мережні контролери, вузли комутації та ін.

Мережні програмні засоби керують роботою комп'ютерної мережі і забезпечують відповідний інтерфейс з користувачами. До мережних програмних засобів належать мережні операційні системи і допоміжні (сервісні) програми.

Основні програмні і апаратні компоненти мережі.

Весь комплекс програмно-апаратних засобів мережі може бути описаний багатошаровою моделлю. У основі будь-якої мережі лежить апаратний шар стандартизованных комп'ютерних платформ. В даний час в мережах широко і успішно застосовуються комп'ютери різних класів - від персональних комп'ютерів до мейнфреймів і супер ЕОМ. Набір комп'ютерів в мережі повинен відповідати набору різноманітних завдань, що вирішуються мережею.

Другий шар - це комунікаційне устаткування. Хоча комп'ютери і є центральними елементами обробки даних в мережах, останнім часом не менш важливу роль почали грати комунікаційні пристрої. Кабельні системи, повторювачи, мости, комутатори, маршрутизатори і модульні концентратори з допоміжних компонентів мережі перетворилися на основних разом з комп'ютерами і системним програмним забезпеченням як по впливу на характеристики мережі, так і за вартістю. Сьогодні комунікаційний пристрій може бути складним спеціалізованим мультипроцесором, який потрібно конфігурувати, оптимізувати і адмініструвати.

Третім шаром, створюючим програмну платформу мережі, є операційні системи (ОС). Від того, які концепції управління локальними і розподіленими ресурсами покладені в основу мережевої ОС, залежить ефективність роботи всієї мережі. При проектуванні мережі важливо враховувати, наскільки просто дана операційна система може взаємодіяти з іншими ОС мережі, наскільки вона забезпечує безпеку і захищеність даних, до якого ступеня вона дозволяє нарощувати число користувачів, чи можна перенести її на комп'ютер іншого типу і багато інших міркувань.

Самим верхнім шаром мережевих засобів є різні мережеві застосування, такі як мережні бази даних, поштові системи, засоби архівації даних, системи автоматизації колективної роботи і ін.

Таким чином, вибір комп'ютерної мережі може бути зведений до вибору її топологи, протоколів, апаратних засобів і мережного програмного забезпечення. Кожен з цих компонентів є відносно незалежним. Наприклад, мережі з однаковою топологією можуть використовувати різні методи доступу, протоколи і мережне програмне забезпечення. У різних мережах можуть застосовуватись однакові про-

токоли і (або) мережне програмне забезпечення. Це розширює можливість вибору найбільш оптимальної архітектури комп'ютерної мережі.

1.4 Розробка структури підсистеми для розв’язання задач багатокритеріальної оптимізації методом послідовних поступок

Першим кроком при написанні програм для системи в середовищі візуального програмування Delphi є підготовка інтерфейсу користувача. Інтерфейс користувача включає в себе набір форм, у яких розміщені кнопки, поля вводу вхідних даних, мітки, тощо. Форма та всі інструменти, які наявні в руках розробника складають візуальну частину системи, тобто, те що демонструється замовнику. Інша - прихована та невидима частина інтерфейсу - містить код, завдяки якому програма і виконує корисні дії.

В бібліотеку візуальних компонентів Delphi включено множину компонентів, і їх номенклатура постійно розширюється від версії до версії. Сучасні компоненти можна використати для побудови будь-якої системи, при цьому їх буде цілком достатньо (створювати власні немає потреби). Крім того, середовища візуального програмування дозволяють навіть малодосвідченим програмістам створювати інтерфейс складних пакетів прикладних програм. Як правило, лише після розробки зручного інтерфейсу користувача починається власне програмування. Створюючи препроцесор системи і вставляючи в неї елементи керування, програмісти задають певні властивості, досягаючи тим самим «правильного» інтерфейсу користувача.

Одна з найбільш трудоємних задач при створенні програмних систем - спроектувати простий та інтуїтивно-зрозумілий інтерфейс. Сформулюємо основні вимоги до майбутнього інтерфейсу підсистеми для розв’язання задач багатокритеріальної оптимізації методом послідовних поступок:

- простий (неперевантаження форми великою кількістю кнопок, полів,тощо);
- автоматичне блокування та розблокування певних полів форм;

- введення "гарячих" клавіш для керування інтерфейсом користувача;
- реалізація засобів, які б дозволяли керувати системою з клавіатури;
- реалізація процедур контролю даних;
- вставити фокус;
- визначити зручний порядок об`єкту полів (елементів) форми.

При реалізації форм інтерфейсу підсистеми для розв’язання задач багатокритеріальної оптимізації постараємося врахувати всі вище зазначені вимоги.

Відповідно, враховуючи вищенаписане, можна побудувати структуру підсистеми призначену для автоматизованого розв’язання задач багатокритеріальної оптимізації методом послідовних поступок (рис. 1.4) Структура підсистеми буде класичною та буде складатися з трьох основних компонентів:

- препроцесор для формування вхідного завдання;
- процесор знаходження розв’язку;
- постпроцесор відображення результатів розв’язку.

Для узгодження роботи цих підсистем розроблено моніторо-діалоговий інтерфейс, який забезпечує:

- передачу управління одній з трьох підсистем за допомогою системи ієрархічних меню;
- діагностику нештатних ситуацій;
- систему діалогової взаємодії та підказок.

Структура цієї системи включає ряд модулів та підсистем, а саме: підсистема розрахунків (процесор); підсистема введення даних (препроцесор); підсистема виведення даних (постпроцесор); модуль інтерфейсу системи. Керує системою модуль інтерфейсу, який включає набір форм.

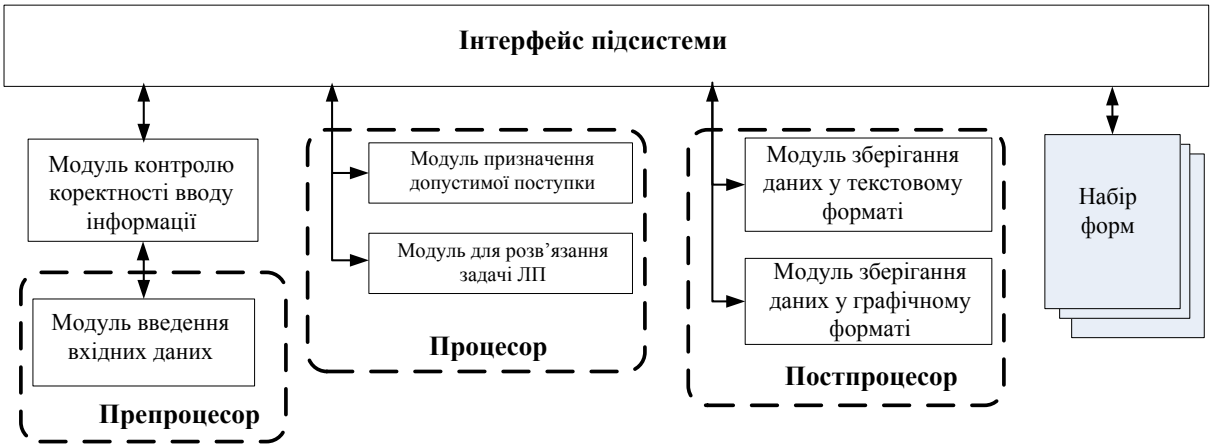

Рисунок 1.4 - Структура підсистеми

1.5 Висновки

Розроблено структуру підсистеми розв'язку задач багатокритеріальної оптимізації методом послідовних поступок, яка включає підсистему вводу вхідних даних, підсистему розрахунків, підсистему виведення даних та інтерфейс.

Проведено аналіз існуючих комп'ютерних мереж та визначено їх переваги та недоліки.

2 РОЗРОБЛЕННЯ МОДЕЛІ ТА АЛГОРИТМІВ ОПТИМІЗАЦІЇ СТРУКТУР КОМП'ЮТЕРНИХ МЕРЕЖ

2.1 Основні поняття та визначення задач багатокритеріальної оптимізації

У задачах математичного програмування з одним критерієм потрібно визначити значення цільової функція, відповідне, наприклад, мінімальним витратам або максимальному прибутку. Практично в будь-якій реальній ситуації виявимо декілька цілей, що частково або повністю суперечать одна одній.

Для прикладу наведено декілька реальних задач, що показують, наскільки широкий діапазон проблем, які можуть бути адекватно сформульовані як багатокритерійні, і які характеристики слід використовувати як критеріїв:

Планування виробництва

max { сумарний чистий дохід }

max { мінімальний чистий дохід за будь-який період }

min { число невиконаних замовлень }

min { наднормовий час }

min { запаси готової продукції }

Вибір портфеля цінних паперів

max { дохід }

min { ризик }

max { дивіденди }

min { відхилення від бажаного рівня різноманітності паперів}

Транспортування

min { вартість }

min { середній час доставки вантажів пріоритетним клієнтам}

min { витрата }

Планування очищення нафти

min { витрати }

min { кількість сирої нафти, що імпортується }

min { кількість сировини з високим вмістом сірки }

min { відхилення від заданого складу }

min { газів }

Складання кошторису капіталовкладень

max { наявність засобів }

min { попит на капітальні вкладення}

min { щорічні експлуатаційні витрати}

max {інвестиції в проекти, пов'язані з охороною навколишнього середовища}

max { інвестиція в проекти в заданому регіоні}

max { інвестиції в проекти по заданій товарній спеціалізації}

Управління лісовим господарством

max { стійкий урожай деревини }

max { людино-день відпочинку в лісі }

max { людино-день полювання в лісі }

max { ареал диких тварин }

max { число місяців випасу домашніх тварин }

min { перевищення бюджету }

Отже, для ефективного вирішення даних задач необхідно побудувати багатокритерійну математичну модель, яку потім потрібно оптимізувати, заздалегідь вибравши найбільш придатний для цього метод. Від правильної побудови математичної моделі в значній мірі залежить успіх правильного рішеня поставленої задачі, тому розглянемо основні принципи побудови математичних моделей:

- Математичні моделі будують на основі відомої змістовної постановки задачі.
- Складання математичної моделі починають з вибору змінних задачі. При цьому слід мати на увазі, що у більшості випадків від вдалого вибору цих змінних залежить простота моделі та складність її розв'язування.
- Після вибору змінних, виходячи із змістовного формулювання задачі, послідовно складають лінійні обмеження, які ці змінні повинні задовольняти. При цьому потрібно слідкувати, щоб у модель були введені

всі обмежувальні умови і в той же час не було жодної зайвої або записаної у більш жорсткій формі, ніж потрібно за умовами задачі.

- Наступним кроком є складання цільової функції, яка в математичній формі відображає заданий в умовах задачі критерій оптимізації і яка повинна бути лінійною.
- Зауважимо, що в деяких моделях зручніше цільову функцію будувати відразу після вибору змінних задачі, тобто порядок побудови моделі не є жорстким і може змінюватися.

Задача багатокритерійного математичного програмування має вигляд:

$$Max\{f_1(x) = F_1\},$$
$$Max\{f_2(x) = F_2\},$$
$$\ldots\ldots\ldots\ldots\ldots$$
$$Max\{f_k(x) = F_k\}$$

де X – множина допустимих значень змінних x; k – число цільових функцій (критеріїв); F_i – значення i-го критерію (цільової функції), *«max»* – означає, що даний критерій потрібно максимізувати або мінімізувати .

Відмітимо, що багатокритерійна задача відрізняється від звичайної задачі оптимізації наявністю декількох цільових функцій замість однієї.

За наявності в багатокритерійній задачі критеріїв з різною розмірністю з метою усунення даної проблеми використовують нормалізацію критеріїв. Способи нормалізації представлені в таблиці 2.1.

В таблиці 2.1 y – елемент простору G. G – простір елементів довільної природи, що називаються цільовими термами (у конкретних інтерпретаціях це сукупність, перелік або нумерація якісних властивостей) елементів $x \in X$.

Згортанням компонент багатоцільового критерію $f \in F$ називається відображення $g \in \{F \to R\}$, яке перетворить сукупність компонент багатоцільового показника f, відповідних цільовим термам $y \in Y$, у скалярний цільовий показник $g(f(x \mid y)) = g[\{f(x \mid y)\}, y \in Y] \in R$. Основними видами згорток є лінійні, мінімізації, максимізації і функції.

Таблиця 2.1 - Способи нормалізації критеріїв

Нормалізація	Математичний вираз
Зведення до безрозмірних величин	$\frac{f(x \mid y)}{\rho(f(x \mid y))}$
Приведення до однієї розмірності $\rho(b)$	$\frac{f(x \mid y)}{\alpha(y)}$ $\rho(f(x \mid y)) = \rho(\alpha(y)) \times \rho(b)$
Зміна інгредієнту	$-f(x \mid y) \quad 1/f(x \mid y)$
Природна нормалізація $\lambda_i(x \mid y)$	$\frac{(f(x \mid y)) - \min(f(x \mid y))}{\max(f(x \mid y)) - \min(f(x \mid y))}$
Нормалізація порівняння	$\frac{f(x \mid y)}{\max(f(x \mid y))}$
Нормалізація Савіджа	$\max(f(x \mid y)) - f(x \mid y)$
Нормалізація усереднювання	$\frac{f(x \mid y)}{\sum_{y \in Y} f(x \mid y)}$

Кобба-Дугласа вигляду:

$$g(f(x)) = \sum_{y \in Y} \alpha(y) f(x \mid y),$$
$$g(f(x)) = \min[\alpha(y) f(x \mid y) + \beta(y)],$$
$$g(f(x)) = \max[\alpha(y) f(x \mid y) + \beta(y)],$$
$$g(f(x)) = \prod_{y \in Y} \alpha(y) f(x \mid y),$$
$$g(f(x)) = \prod_{y \in Y} [\alpha(y) f(x \mid y)]^{\beta(y)}.$$

Проблеми отримання і обгрунтування вибору згорток складають основний напрям теорії корисності.

У завданнях вибору рішення, що формалізуються у вигляді моделі векторної оптимізації, першим природним кроком слід вважати виділення області компромісів (або рішень, оптимальних по Парето).

Вектор називається оптимальним по Парето рішенням, якщо не існує $x \in X$ такого, що виконані нерівності:

$$f(x \mid y) \geq f(x^0 \mid y) \forall y \in Y \wedge f(x \mid y) \neq f(x^0 \mid y).$$

Областю компромісів Γ_x називається підмножина допустимої множини рішень X, що володіє наступною властивістю, всі рішення, що належать йому, не можуть бути покращені одночасно по всіх локальних критеріях — компонентам вектора ефективності.

Оптимальне рішення, вибране на основі багатокритерійного підходу незалежно від обраного принципу оптимальності, завжди повинне належати області компромісів. Інакше його можна буде покращити і, отже, воно не є оптимальним. Таким чином, область компромісів є область потенційно оптимальних розв'язків. Отже, при виборі рішення по векторному критерію ефективності можна обмежити пошук оптимального рішення областю компромісів Γ_x, яка, як правило, значно вужча всієї області можливих значень X.

2.2 Методи та особливості побудови алгоритмів розв'язування задач багатокритеріальної оптимізації

Розглянемо основні методи розв'язання багатокритеріальних задач математичного програмування.

Принцип справедливого компромісу

Нехай всі локальні критерії, створюючі вектор ефективності, мають однакову важливість. Справедливим вважають такий компроміс, при якому відносний рівень зниження якості по одному або декільком критеріям не перевищує відносного рівня підвищення якості по решті критеріїв (менше або рівний).

Цьому принципу можна дати наступну математичну інтерпретацію. Нехай у області компромісів Γ_x задані два розв'язки x' і x'', які оцінюються критеріями $F_1(\tilde{x})$ і $F_2(\tilde{x})$. Розв'язок x' кращий ніж розв'язок x'' по критерію F_1, але

поступається йому по критерію F_2. Необхідно порівняти ці розв'язки і вибрати кращий на основі принципу справедливого компромісу.

Для порівняння цих розв'язків на основі принципу справедливого компромісу введемо міру відносного зниження якості розв'язкупо кожному з критеріїв – ціну поступки x:

$$x_1 = \frac{\Delta F_1(x', x'')}{\max F_1(x)},$$

$$x_2 = \frac{\Delta F_2(x', x'')}{\max F_2(x)}.$$

Якщо відносне зниження критерію F_1 більше, ніж критерію F_2, то слід віддати перевагу розв'язку x'. Це слідує з порівняння ціни поступки по кожному критерію.

Алгоритм рішення задачі багатокритеріальної оптимізації, оснований на принципі справедливого компромісу, включає наступні кроки.

Крок 1. Вибираємо x' і x''.

Крок 2. Обчислюємо за формулою (2.1) x_1 і x_2.

Крок 3. Якщо $x_1 > x_2$, то вибираємо x', якщо $x_1 > x_2$, то вибираємо x''.

Кінець.

Модель визначення області компромісів, а також модель справедливого компромісу інваріантні до масштабу вимірювання критеріїв. Проте перша з них є допоміжною при виборі рішення, а інша може бути використана тільки в тих ситуаціях вибору рішення, для яких ідея справедливого компромісу може бути виправдана. Тому в більшості випадків доводиться вдаватися до інших принципів оптимальності, що мають сенс тільки у разі нормалізованого простору критеріїв, коли всі локальні критерії мають єдиний масштаб вимірювання. В більшості випадків масштаби вимірювання критеріїв неоднакові, і виникає необхідність проводити нормалізацію критеріїв, тобто штучно приводити їх до єдиної міри.

Більшість принципів нормалізації грунтуються на введенні ідеального рішення, тобто рішення, що володіє ідеальним вектором ефективності. Це

ідеальне рішення можна апріорно припустити виходячи з інформації про об'єкт, а можна вирішити задачу оптимізації для кожного локального критерію і відповідно цим рішенням, значення вектора ефективності прийняти за ідеальний вектор ефективності. Тоді вибір оптимального рішення стає рівнозначним найкращому наближенню до цього ідеального вектора.

В цьому випадку замість дійсного значення критеріїв розглядаються їх відхилення від ідеального значення

$$\Delta F_j = F_{ij} - F_j ,$$

або їх безрозмірне відносне значення

$$\overline{F_j} = F_j / F_{ij} .$$

При рішенні даної проблеми використовуються обидва способи нормалізації. Таким чином, успішне рішення проблеми нормалізації багато в чому залежить від того, наскільки точно і об'єктивно вдається визначити ідеальну якість розв’язку.

Після нормалізації критеріїв ефективності задача вибору рішення набуває чіткого математичного сенсу. У теоретико-множинному відношенні вона стає завданням впорядковування обмежених векторних множин, а з погляду теорії наближень – завданням наближення в метричному скінченному просторі. Це дає можливість проводити обгрунтований вибір принципів оптимальності і виявляти їх логічний сенс.

Отже, для даного випадку принцип оптимальності ідентичний принципу наближення, а узагальнений скалярний критерій оптимізації – критерію наближення, що є деякою функцією відхилення від ідеальної функції.

Принцип слабкої оптимальності по Парето

Вектор $x' \in X$ називається слабо оптимальним по Парето розв’язку(оптимальним по Слейтеру), якщо не існує вектора $x \in X$, такого, що задовольняє умові:

$$f(x \mid y) \geq F(x' \mid y) \forall y \in Y .$$

Нехай x_{oj} (i=1,m) є оптимальними розв'язками для звичайних скалярних оптимізаційних задача, кожна з яких максимізувала компоненту $F_i(x)$ вектора F(x):

$$\max F_j(x), j = 1, m.$$

Якщо вони є максимальними розв'язком для кожної і, то вважаємо, що $F_j(x_{oj})>F_i(x_j)$ (i=1,m), де x_{oj} — оптимальне рішення задачі.

Припустимо, що S_{oj} відображає множину розв'язків, кожен з яких відповідає компоненті F_j, і

$$S_{oj} = \{x \mid F_j(x) \le F_j(x_{oj}) + a_j\},$$

де a_j - відображає допустиму кількість обмежень відповідної області по відношенню до F_j.

Тоді оптимальний достатній розв'язокце такий розв'язок, при якому мінімальний компонент (найгірший компонент) максимізовувався на множині, що задовольняє достатню умову $x \in X$ і $x \in S_{om}$. Він може бути сформульовано так:

max z

x, z при $F_j(x)>z$, $F_i(x)>F_j(x)_{oj}-a_j$, i=1,m , $x \in X$

Задача не має розв'язку, якщо a_j не настільки велике, що перетин $\{S°_j\}$ непорожній. Величини a_j повинні бути визначені на основі значень $F_j(x_{oj})$ або аналізу точності. Можна довести, що оптимальне рішення останньої задачі є оптимальне рішення по Парето.

Алгоритм рішення задачі має наступні етапи.

Крок 1. Вважаємо l=1 і вирішуємо задачу

max z

x, z при $F_j(x)> z$; $F_i(x)>F_j(x)_{oj} - a_j$, i=1,m; $x \in X$.

Знахидимо початковий розв'язокx_1 і оцінюємо цільову функцію $F(x_1)$.

Крок 2. Коли x_l задане, розкладаємо $F(x_l)$ на задовільні і незадовільні компоненти. Позначимо їх відповідно через S і $\overline{S}$. Якщо $\overline{S}$, тоді задача вважається нерозв'язною, а якщо $S \equiv \varnothing$ то x_1 — оптимальний розв'зок, що відповідає вимогам. Якщо $\overline{S}$ <> $\varnothing$ і S <> $\varnothing$, то для S_l визначається $a_{lj}>0$,

допустиме по відношенню до $F_j(x_l)$ $a_{lj}=0$ означає, що j-я цільова функція $f_j(x)$ не може приймати значення, відмінне від $f_j(x_l)$.

Крок 3. Розв'язуємо задачу

max z

x, z за умови $F_j(x) \in z$, $j \in \overline{S}l$ $F_i(x)>F_j(x)_{oj} - a_j$, $j \in S_l$ $x \in X$

Викликаємо початковий розв'язок x_l+l. Якщо $x_l+1=x_l$, то задача буде нерозв'язною; якщо $x_l+1<>x_l$, то вважаємо 1 = 1+1 і повертаємося до кроку 2.

Кінець.

Принцип наближення по всіх локальних критеріях до ідеального розв'зку

В основі даного підходу покладена ідея наближення по всіх критеріях. Нехай дано задачу багатокритерійного програмування

$$Max\{f_1(x) = F_1\},$$
$$Max\{f_2(x) = F_2\},$$
$$\ldots\ldots\ldots\ldots\ldots$$
$$Max\{f_k(x) = F_k\}$$

і задані граничні умови

$$\alpha_{\alpha 1}x_1 + \alpha_{\alpha 2}x_2 + \ldots + \alpha_{\alpha n}x_n = b_\alpha (\alpha = \overline{1, r});$$

$$\alpha_{r+\beta,1}x_1 + \alpha_{r+\beta,2}x_2 + \ldots + \alpha_{r+\beta,n}x_n \le b_{r+\beta}; (\beta = \overline{1, m-r});$$

Серед розв'язків системи потрібно відшукати таке значення вектора x, при якому локальні критерії приймуть по можливості максимальне (мінімальне) значення одночасно.

Розглянемо кожну окрему функцію і допустимо, що для кожного фіксованого і (i=1,m) вирішене завдання максимізації. Нехай відповідні оптимальні плани характеризуються векторами:

$$xoi(x_{o1}, x_{o2}, \ldots, x_{on})$$

На цих оптимальних планах визначимо значення критеріїв відповідно

$$F_{oi}=(F_i(x_{o1}), F_i(x_{o2}), \ldots, F_i(x_{on}))$$

Розглянемо вектор F(x) з компонентами F(x) і складемо квадрат Евклідової норми

$$R(x) = \left\| F(x) - F_o \right\|^2$$

де вектора F(x) - F_o, визначеного для всіх $x \in \Omega$.

Відмітимо, що F_o буде одиничним вектором в просторі вектора F(x). Назвемо його ідеальним значенням вектора F(x). Поставлене завдання тепер формулюватиметься так: дана система цільових функцій і дані умови задачі, потрібно визначити точку $x \in \Omega$, у якій функція R(x) досягає мінімуму. Таким чином, пошук векторно-оптимального плану $x \in \Omega$ у даній задачі зведено до оптимізації виразу $R(x) = \left\| F(x) - F_o \right\|^2$ на розв'язках системи лінійних нерівностей умови. Оскільки даний вираз є квадратичною функцією змінних $x_1, x_2, \ldots, x_n$ то задача пошуку векторно-оптимального плану звелася до задачі квадратичного програмування:

Задана квадратична функція R(x), визначена на множині $x \in \Omega$. Потрібно відшукати точку $x \in \Omega$, що забезпечує виконання умови $R(x^*) = \min R(x)$.

Метод квазіоптимізації локальних критеріїв (метод послідовних поступок)

В методі послідовних поступок здійснюється пошук не єдиного точного оптимуму, а деякої області розв'язків, близьких до оптимального, – квазіоптимальної множини. При цьому рівень допустимого відхилення від точного оптимуму визначається з урахуванням точності постановки завдання (наприклад, залежно від точності обчислення величини критеріїв), а також деяких практичних міркувань (наприклад, вимог точності рішення задачі).

Спочатку проводиться якісний аналіз відносної важливості критеріїв; на підставі такого аналізу критерії розташовуються і нумеруються в порядку зменшення важливості, так що головним вважається критерій F_1, менш важливий F_2, потім слідує решта локальних критеріїв $F_3, F_4, \ldots, F_m$. Максимізовується перший по важливості критерій F_1 і визначається його найбільше значення M_1. Потім призначається допустиме зниження (поступка) $d_1 \geq 0$ критерію F_1.

Визначимо нову допустиму область X(1), як підобласть X виду

$$X(1) = X_n \{ x \mid F_1(x) \geq M_1 - d_1 \}.$$

Такий підхід дозволяє значно звузити первинну допустиму область X, коли переходимо до наступного по важливості критерію.

Після цього знаходимо найбільше значення M_2 другого критерію F_2 на множині X(1), тобто за умови, що значення першого критерію повинне бути не менше, ніж $M_1 - d_1$. Знову призначається значення поступки $d_2 \geq 0$, але вже по другому критерію, яке разом з першим використовується при знаходженні умовного максимуму третього критерію, і т.д. Нарешті, максимізовувався останній по важливості критерій F_m за умови, що значення кожного критерію F_r з m—1 попередніх повинне бути не менше відповідної величини $M_r - d_r$ одержувані стратегії вважаються оптимальними.

Таким чином, оптимальною вважається стратегія, що є розв'язком останньої задачі.

Якщо критерій F_m на множині стратегій (що задовольняють обмеженням задачі m) не досягає свого найбільшого значення M_m, то рішенням багатокритерійної задачі вважають максимізуючу послідовність x_k, з послідовності множин: $X_{m-1} \subset X_{m-2} \subset \ldots X_1 \subset X$.

Практично подібні послідовності, що максимізувалися, має сенс розглядати і для того випадку, коли верхня границя в задачі m не досягається, оскільки для вирішення екстремальних задач широко застосовуються ітеративні методи.

Алгоритм рішення задачі векторної оптимізації включає наступні кроки:

Крок 1. Нехай x_0 — розв'зок задачі $\max F_1(x_0) \in X$.

Крок 2. Нехай x_k - розв'зок задачі $\max F_k(x\ ) \in X_{k-1}$, де x_k визначається після призначення поступки.

Крок 3. Якщо k<m, то встановлюємо k=k+1 і повторюємо крок 2. Якщо k = m, то x_m вважаємо оптимальним рішенням.

Кінець.

Значення поступок d_i (i=1,m) послідовно призначаються при вивченні взаємозв'язку локальних критеріїв.

Спочатку розв'язується задача про призначення допустимого зниження d_i першого критерію від найбільшого значення M_i. Практично для цього задають декілька величин поступок d_{11}, d_{21}, d_{31}, ... і шляхом рішення задачі визначають відповідні максимальні значення $M_2(d_{11})$, $M_2(d_{21})$, $M_2(d_{31})$, ... другого критерію. Далі розглядають пару критеріїв F_2 і F_3. Знов призначають пробні значення поступок, рішають задачу, відшукують найбільші значення $M_3(d_{11})$, $M_3(d_{21})$. Одержані дані аналізують, призначають d_i, переходять до наступної пари критеріїв F_3 та F_4, тощо. Нарешті, в результаті аналізу взаємного впливу критеріїв F_{m-1} і F_m вибирають значення останньої поступки d_{m-1} і знаходять оптимальні стратегії, звичайно обмежуються знаходженням однієї такій стратегії.

Таким чином, хоча формально при використанні методу послідовних поступок досить вирішити лише одне з завдань, проте для призначення значень поступок з метою з'ясування взаємозв'язку локальних критеріїв фактично доводиться вирішувати істотно більше число таких завдань.

Для вирішення багатокритерійного завдання потрібно так рангувати критерії, щоб потім зручніше було вибирати значення поступок.

Отже, можна зробити наступний висновок. Метод послідовних поступок доцільно застосовувати для вирішення тих багатокритерійних задач, в яких всі приватні критерії природним чином впорядковані по мірі важливості, причому кожен критерій настільки суттєво важливіший, ніж наступний, що можна обмежитися обліком тільки попарного зв'язку критеріїв і вибирати допустиме зниження чергового критерію з врахуванням поведінки лише одного наступного критерію.

Особливо зручним є випадок, коли вже в результаті попереднього аналізу багатокритерійної задачі з'ясовується, що можна допустити поступки лише в межах „інженерної" точності (5-10% від найбільшого значення критерію).

Метод згортання векторного критерію в суперкритерій

Одним з поширених методів рішення багатокритерійних задач є метод зведення багатокритерійної задачі до однокритерійної шляхом згортання векторного критерію в суперкритерій. При цьому кожен критерій перемножується на відповідний йому ваговий коефіцієнт (коефіцієнт важливості).

$$P(k) = \sum_{i=1}^{m} \alpha_i F_i, \alpha_i \geq 0 .$$

При цьому виникають труднощі з *правильним* підбором вагових коефіцієнтів α_i. Існують різні способи вибору коефіцієнтів α_i. Одним з них є призначення α_i залежно від відносної важливості критеріїв. Такий підбір вказаних коефіцієнтів можна виконувати згідно таблиці 2.2.

Таблиця 2.2 - Шкала відносної важливості

Інтенсивність відносної важливості	Визначення
1	Рівна важливість порівнюваних вимог
3	Помірна (слабка) перевага одного над іншим
5	Сильна (істотна) перевага
7	Очевидна перевага
9	Абсолютна (пригнічувана) перевага
2,4,6,8	Проміжні рішення між двома сусідніми оцінками

2.3 Особливості розв’язання задач багатокритеріальної оптимізації

Процедура рішення багатокритерійної задачі методом послідовних поступок полягає в наступному:

1) всі критерії розташовують і нумерують у порядку їх відносної важливості;

2) максимізовують перший, найбільш важливий критерій;

3) призначають величину допустимого зниження значення цього критерію, для покращення значення всіх інших критерії, це здійснюється шляхом введення додаткового обмеження;
4) поступово оптимізують всі інші критерії, в порядку спадання їхньої важливості, і призначають відповідні поступки;
5) робота завершується при оптимізації останнього критерію, коли призначені поступки всіх попередніх критеріїв, або на будь-якому кроці при досягнені бажаного результату;
6) проаналізувавши отримані результати, можна повторно виконати розрахунки, призначаючи інші значення поступок.

Для отримання оптимального значення по кожному критерію придатний будь-який метод рішення однокритерійної задачі даного типу В даній дипломній роботі використаний симплекс метод рішення задачі лінійного програмування.

Симплекс метод розв'язання задачі лінійного програмування базується на двох теоремах.

Теорема 1 (про оптимальні розв'язки задачі ЛП і вершини її допустимої області).

Якщо цільова функція задачі ЛП

$$\max\left\{c^T x : \sum_{j=1}^{n} x_j A_j = A_0, x \geq 0\right\}$$

набуває максимального значення в деякій допустимій точці $x^* \in D = \{x \in R^n, Ax = a_0, x \geq 0\}$, то вона набуває цього ж значення і в деякій вершині D.

Теорема 2 (про опорні розв'язки задачі ЛП і вершини її допустимої області).

Допустимий розв'язок задачі ЛП є вершиною її допустимої області тоді і тільки тоді, коли він є базисний (опорний).

Узагальнений алгоритм розв'язку задачі лінійного програмування симплекс методом складається з наступних кроків.

Крок 1. Визначають початкове допустиме рішення (початковий базис) симплексу. Змінні, що не входять в базис прирівнюються до нуля.

Крок 2. З числа небазисних (рівних нулю) змінних вибирається змінна, що буде включатися в новий базис, збільшення якої забезпечує покращання значення цільової функції. Якщо такої змінної немає, обчислення припиняються, оскільки, поточне базисне рішення оптимальне . Інакше здійснюється перехід на крок 3.

Крок 3. З числа змінних поточного базису вибирається змінна, що буде виключатися з базису, їй буде присвоєне нульове значення, при введенні до базису нової змінної, вибраної на кроці 2.

Крок 4. Знаходиться новий розв'язок, відповідно до нових значень базисних та вільних змінних, методом Джордана-Гауса перетворення системи лінійних рівнянь. Здійснюється перехід до кроку 2.

При введені додаткових обмежень в задачу можливий варіант відсутності початкового базису. Для вирішення цієї проблеми використовуються методи знаходження початкового базисного розв’зку.

Одним з них є метод великих штрафів, або як його ще називають в літературі М-метод, він полягає в наступному:

1) веденням додаткових змінних, для утворення необхідного початкового базису;
2) додаваня до цільової функції (у випадку максимізації) $-M * X$, де X – додатково введені змінні, M – дуже велике число.

Таким чином, завдяки введенню в цільову функцію додаткових доданків - якщо введені змінні не будуть дорівнювати 0, цільова функція буде прямувати до 0, що недопустимо. Отже, згідно алгоритму симплекс методу, ці змінні будуть впершу чергу виведені з базису, і більше в нього не потраплять.

Метод великих штрафів, неявно для користувача, використовується в програмі даної дипломної роботи, при введенню додаткого обмеження, вводиться додаткова змінна і модифікуються всі наступні цільові функції.

2.4 Побудова моделі оптимізації структури компютерної мережі

Побудована оптимізаційна модель комп'ютерної мережі включає цільові функції, які описують параметри структури мережі та обмеження.

Зокрема, для визначення мінімізація вартості при заміні старих елементів та ліній зв'язку КМ на нові:

де C_ρ - приведені витрати на заміну КП, які не відповідають вимогам; C_{ijn} - приведені витрати на установку *n*–го типу ліній зв'язку, які не відповідають вимогам між *i*-м та *j*-м пунктами; $y_{i\rho}$=1 якщо на *i*-е місце встановлюють ρ-й тип КП, в протилежному випадку $y_{i\rho}$ =0; x'_{ijn}=1, якщо на місце з'єднання *i*-ого та *j*-ого пунктів встановлюється n-й тип лінії зв'язку, в протилежному випадку x'_{ijn}=0; *i*=*j*=0 – місцеположення сервера; I^C - множина пунктів, у яких розміщено старі КП.

Для мінімізації інтенсивності відмов при зміні старих елементів і ліній зв'язку КМ на нові

де ω_ρ - інтенсивність відмов КП; λ_{ijn} - інтенсивність відмов ліній зв'язку;

У випадку максимізація пропускної здатності при зміні старих КП і ліній зв'язку на нові

де P_ρ - пропускна здатність нових КП, які встановлюються на місце старих для задачі реінжинірингу; P_{ijn} - пропускна здатність нових ліній зв'язку *n*–го типу між *i*-м та *j*-м.

Для визначення надійності структури комп'ютерної мережі в якості цільової функції використовуються вирази для класичної теорії надійності на основі теорем додавання, множення ймовірностей. Наприклад, маємо ланку структури комп'ютерної мережі, де послідовно включені три пристрої. При виході з ладу одного з пристроїв втрачає працездатність весь ланцюг. Імовірність виходу з ладу першого пристрою - 0.01, другого - 0.008, третього – 0.025. Знайти ймовірність того, що електричний ланцюг вийде з ладу. В процесі побудови критерію розглянемо події: A - вихід з ладу електричного ланцюга, A_1 - вихід з ладу 1-го пристрою, A_2 - вихід з ладу 2-го пристрою, A_3 - вихід з ладу 3-го пристрою.

Очевидно, що $A = A_1 + A_2 + A_3$. Передбачається, що події A_1, A_2 і A_3 несумісні, тому $P(A) = P(A_1) + P(A_2) + P(A_3) = 0.01 + 0.008 + 0.025 = 0.043$.

В обмеженнях можемо враховувати максимальні значення вартості, обмеження на мінімальну надійність КС чи КМ та ін.

2.5 Висновки

Побудована модель оптимізації структури компютерної мережі, яка може включати такі критерії як вартість, надійність, пропускна здатність та інтенсивність відмов з відповідними обмеженнями.

Наведено теоретичні основи розв'язку задач багатокритеріальної оптимізації, побудовано алгоритми, особливості та специфіку їх розв'язку. Описано основні методи розв'язку задач багатокритеріальної оптимізації, а саме: метод справедливого компромісу, метод слабкої оптимальності по Парето, метод наближення по всіх локальних критеріях до ідеального розв'зку, метод послідовних поступок та метод згортання векторного критерію в суперкритерій.

3 ОСОБЛИВОСТІ РОЗРОБКИ ПІДСИСТЕМИ ДЛЯ АВТОМАТИЗАЦІЇ РОЗВ'ЯЗАННЯ ЗАДАЧ БАГАТОКРИТЕРІАЛЬНОЇ ОПТИМІЗАЦІЇ ТА ОСНОВНІ РЕЗУЛЬТАТИ

3.1 Розробка програмного та інформаційного забезпечення

Для програмної реалізації оберано середовищє програмування Borland Delphi 7, з використанням об'єктно-орієнтованого підходу. Як мова програмування використовувався діалект мови Pascal – Object Pascal.

Програма має складається з двох основних модулів:

- MainUnit.pas – в ньому має реалізовано ввід даних, покрокове обчислення, призначення поступок, призначення додаткових змінних і штафів (метод великих штрафів), вивід результатів у різних форматах представлення інформації (текстовий, графічний), збереження результатів у зовнішніх файлах.

Опишемо основні змінні, які використовуються в модулі MainUnit.pas:

Simplex:Tsimplex – об'єкти для роботи симплекс методу;

CrutCount:word – кількість критеріїв;

ConsCount:word – кількість обмежень;

ZminCount:word – кількість змінних;

Cr:array of Tcruterij – масив критеріїв;

Con:array of Tconstrain – масив обмежень;

X:TextArray– масив змінних;

Q:TextArray – масив значень цільових функцій;

dQ:array[1..2] of TExtArray – масив значень призначених поступок;

StepNumber:word – номер поточного кроку роботи.

- SimplexUnit.pas – модуль, який реалізовує симплекс метод рішення задачі лінійного програмування.

Основні класи і типи модуля SimplexUnit.pas:

TSimplex – об'єкт, що представляє симплекс-методом;

TExtArray - динамічний масив чисел типу extended.

Основні процедури и функції TSimplex:

Create(const L:TExtArray,maximize:boolean=false) - ініціалізація об’єкта;

AddCons(B:extended; A:TExtArray, Sign:TOperation) - додавання обмеження;

B - значення коефіцієнта *Bi*;

A - масив значень коефіцієнтів *Ai*;

Функція може викликатися стільки разів, скільки обмежень потрібно додати

Solve – знаходження розв’язку;

GetMax – повернення результату (тип extended) функції оптимізації;

GetSolution - повернення координат (тип TExtArray) знайденої точки максимума;

Free - знищення об’єкта.

3.2 Розробка основних алгоритмів

Приклади розроблених алгоритмів наведено на рисунку 3.1 та рисунку 3.2. Зокрема на рисунку 3.1 зображено алгоритм роботи підсистеми, на рисунку 3.2 - алгоритм для реалізації методу послідовних поступок

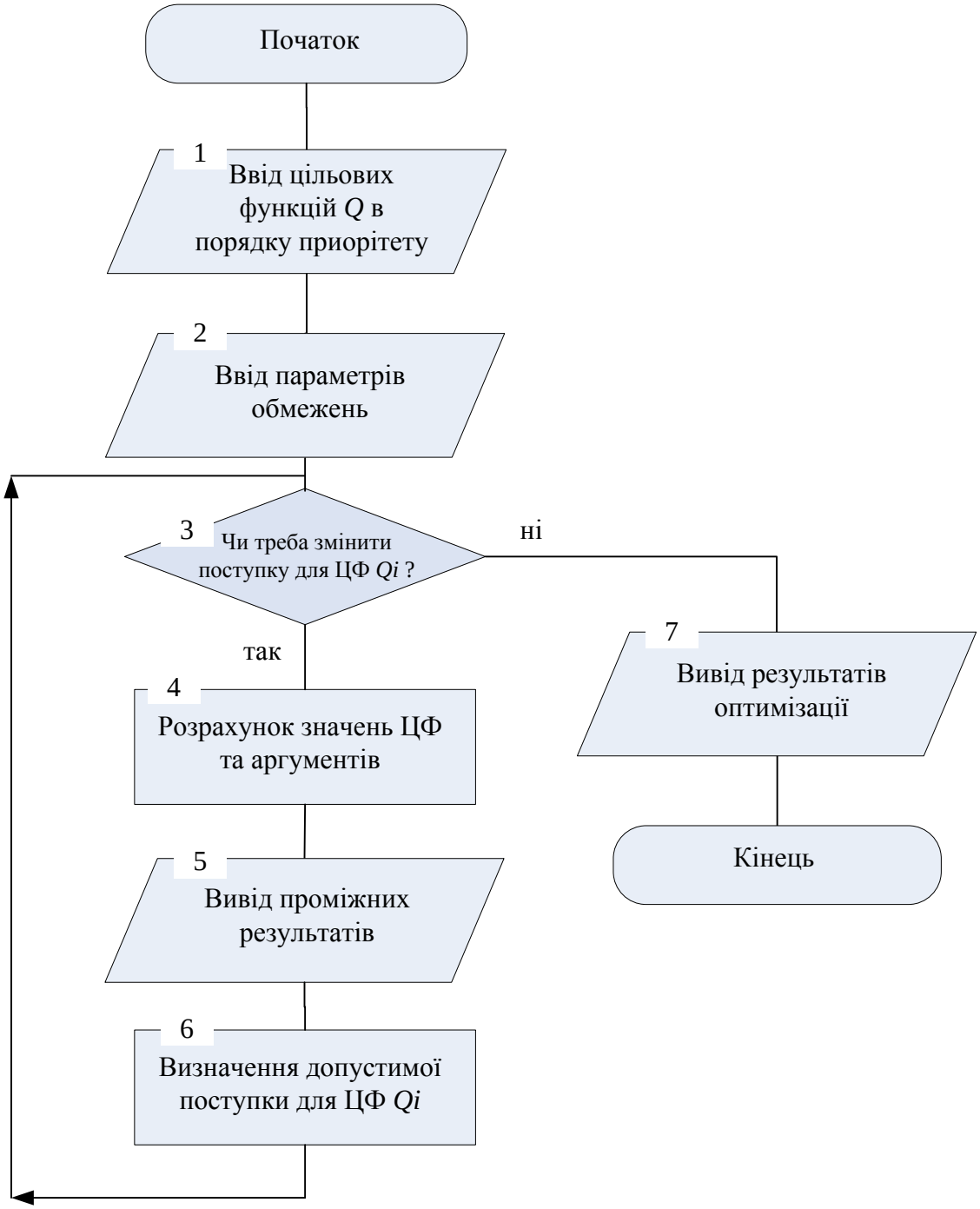

Рисунок 3.1 - Алгоритм роботи підсистеми багатокритеріальної оптимізації

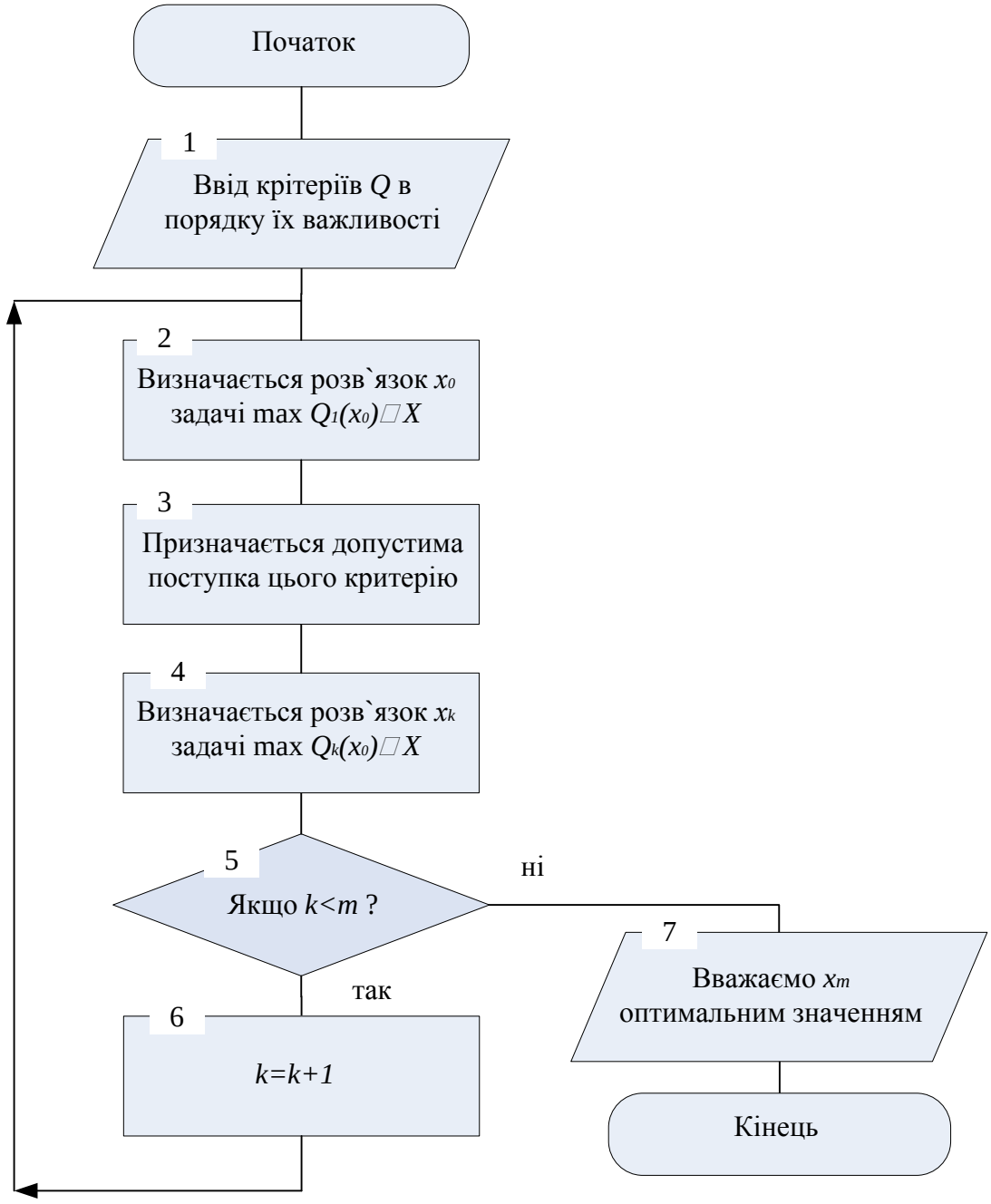

Рисунок 3.2 - Алгоритм розв’язку задачі багатокритеріальної оптимізації методом послідовних поступок

3.3 Інструкція користувача підсистеми

Підсистема для розв’язання задач багатокритеріальної оптимізації з використанням методу поступок включена в систему розв’язання оптимізаційних задач “OptimMEMS”. Відповідно, для завантаженя підсистеми необхідно запустити на виконання файл ProjectOptim.exe і на екран монітора система виведе

форму, приклад якої зображено на рисунку 3.3. У цій формі необхідно вибрати меню «БагатоКритОптим», а з випадаючого підменю «Метод поступок».

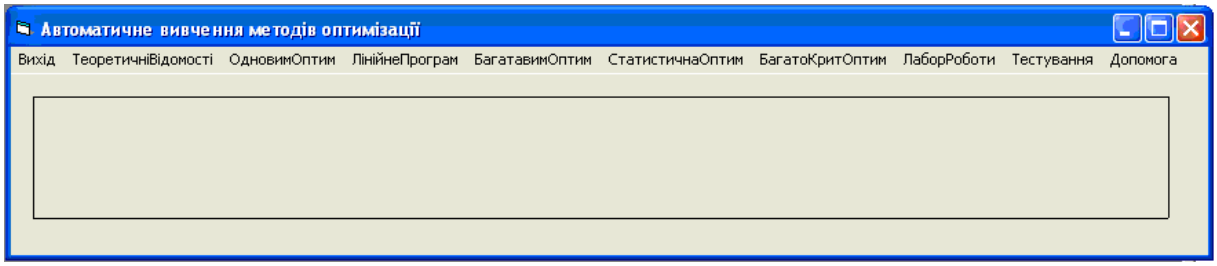

Рисунок 3.3 - Основна форма системи «OptimMEMS»

В результаті виконання попередньої послідовності команд, на екран монітора,система виведе форму підсистеми, приклад якої зображено на рисунку 3.2. Відповідна форма включає основні підменю для вводу/виводу інформації:

- Ввід вхідних даних.
- Розрахунок.
- Графіки Q (цільових функцій).
- Графіки X (значення змінних).
- Про програму (інформація про автора).

Кнопка «Ввід вхідних даних» призначена для завантаження форми призначеної для вводу вхідних даних. Наступне меню під назвою «Розрахунок» дозволяє завантажити форму, що призначена для вводу основних параметрів розв’язку задач багатокритеріальної оптимізації та виводу протоколу роботи програми. Підменю «Графіки Q» та «Графіки X» призначені для виводу результатів роботи підсистеми у графічному вигляді. Кнопки «Зберегти» призначена для збереження даних у графічному файлі.

3.4 Особливості технічного забезпечення підсистеми

Оскільки програмні засоби які використовувались для створення підсистеми розв’язку задач багатокритеріальної оптимізації методом поступок функціонують під операційною системою сімейства Windows, то й мінімальна конфігурація ЕОМ має включати:

- процесор не гірший за Pentium з частотою 700 Мгц на відповідній

материнській платі;

- 256 Мб оперативної пам’яті;
- вінчестер на 2 Гб;
- відеокарта VGA з об’ємом пам’яті 4 Мб;
- кольовий SVGA монітор 15’’, 640X480 точок, LR NI,MPR-II;
- принтер, для друку документів;
- клавіатура, маніпулятор «миша», і т.п. - стандартні.

Слід відмітити, що при такій конфігурації програма буде працювати повільно. Тому, з метою досягнення більшої швидкодії та комфортної роботи корстувача програми рекомендується наступна конфігурація :

- процесор Pentium II з частотою не менше 1000 Мгц ;
- 512 Мб оперативної пам’яті ;
- вінчестер на 8 Гб ;
- відеокарта SVGA з об’ємом пам’яті 4 Мб ;
- кольовий SVGA монітор 17’’, 1024X768 точок, LR NI,MPR-II;
- принтер, для друку документів;
- клавіатура, маніпулятор «миша», і т.п. - стандартні.

3.5 Тестування основних модулів підсистеми

При тестуванні підсистеми були використані вхідні дані з літератури.

Цільові функції:

$$Q_1 = 2 * x_1 + 3 * x_2 + x_3 + 2 * x_4 \to \max$$
$$Q_2 = 3 * x_1 - x_2 \to \max \qquad (3.1.)$$
$$Q_3 = x_1 - 4 * x_2 \to \max$$

обмеження:

$$x_1 + 2 * x_2 + x_3 = 6$$
$$2 * x_1 + x_2 + x_4 = 8$$
$$-x_1 + x_2 + x_5 = 4$$

Введемо відповідні значення у перше вікно підсистеми рисунок 3.4.

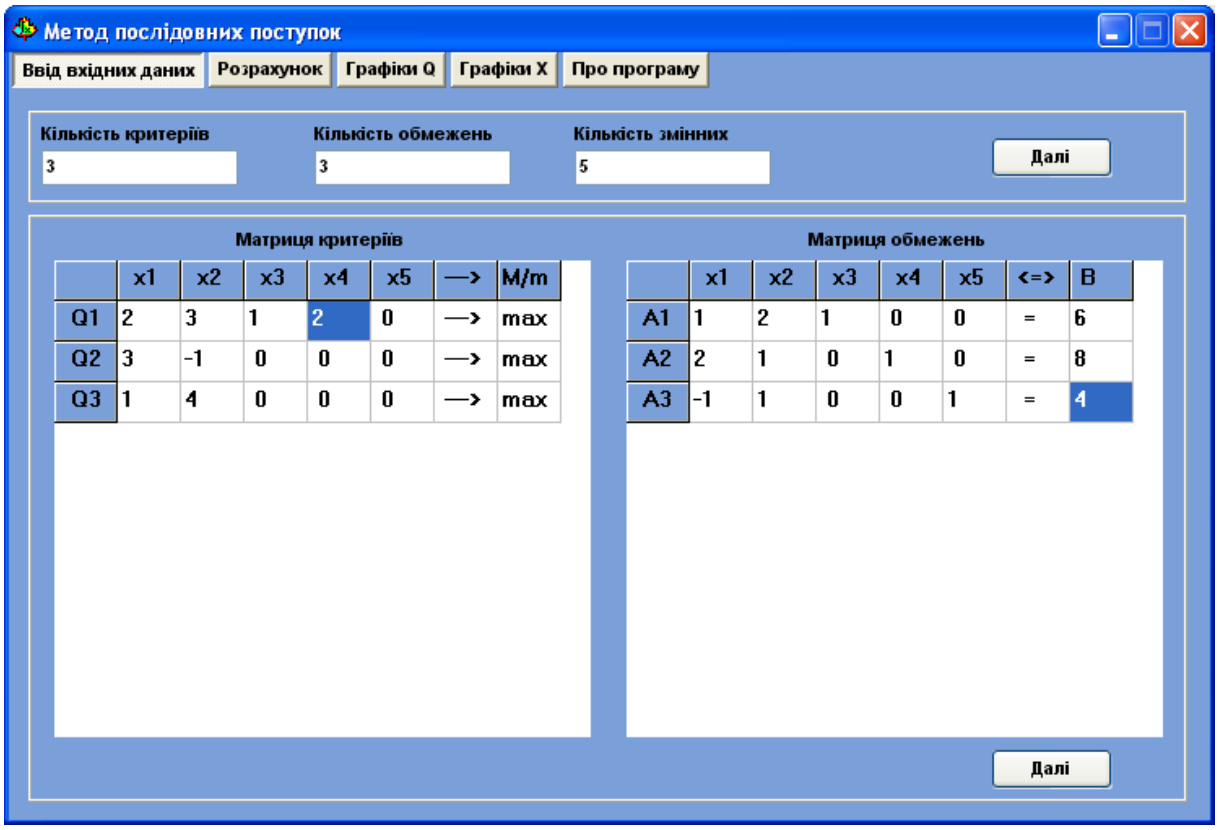

Рисунок 3.4 - Головне вікно підсистеми розв’язку задач багатокритеріальної оптимізації методом поступок

Після цього натискаємо кнопку «Далі», на екран монітора підсистема виведе форму, яка наведена на рисунку 3.5. Підсистема відобразить інформацію про вхідні дані задачі багатокритеріальної оптимізації (критерії, обмеження, значення поступок). Після вводу вхідних даних, згідно зазначеного вище алгоритму натиснувши кнопку «Наступний крок» отримуємо перші результати розв’язку задачі (3.1) рисунок 3.6. Призначаємо поступку і натискаємо кнопку «Добре». Після цього на екран монітора підсистема виведе форму приклад якої зображено на рисунку 3.7.

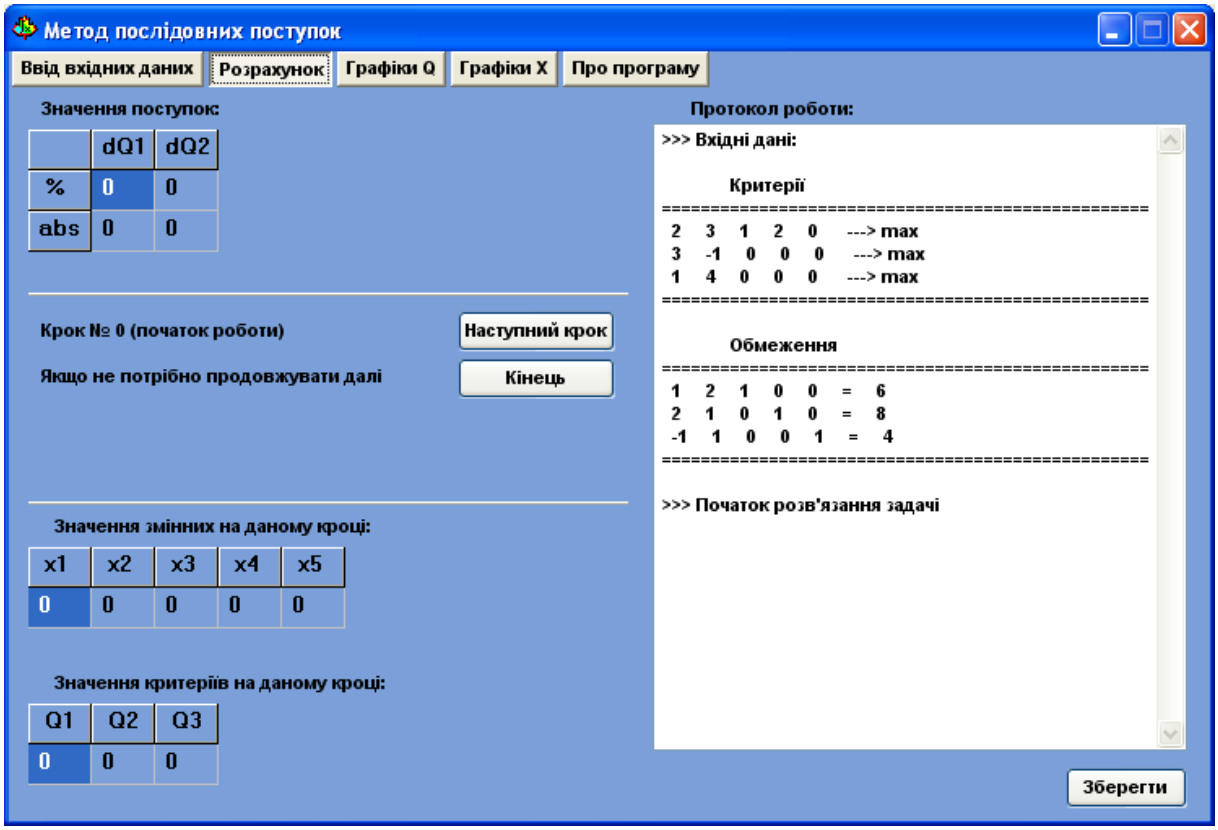

Рисунок 3.5 - Результати виконня введення вхідних даних тестового прикладу

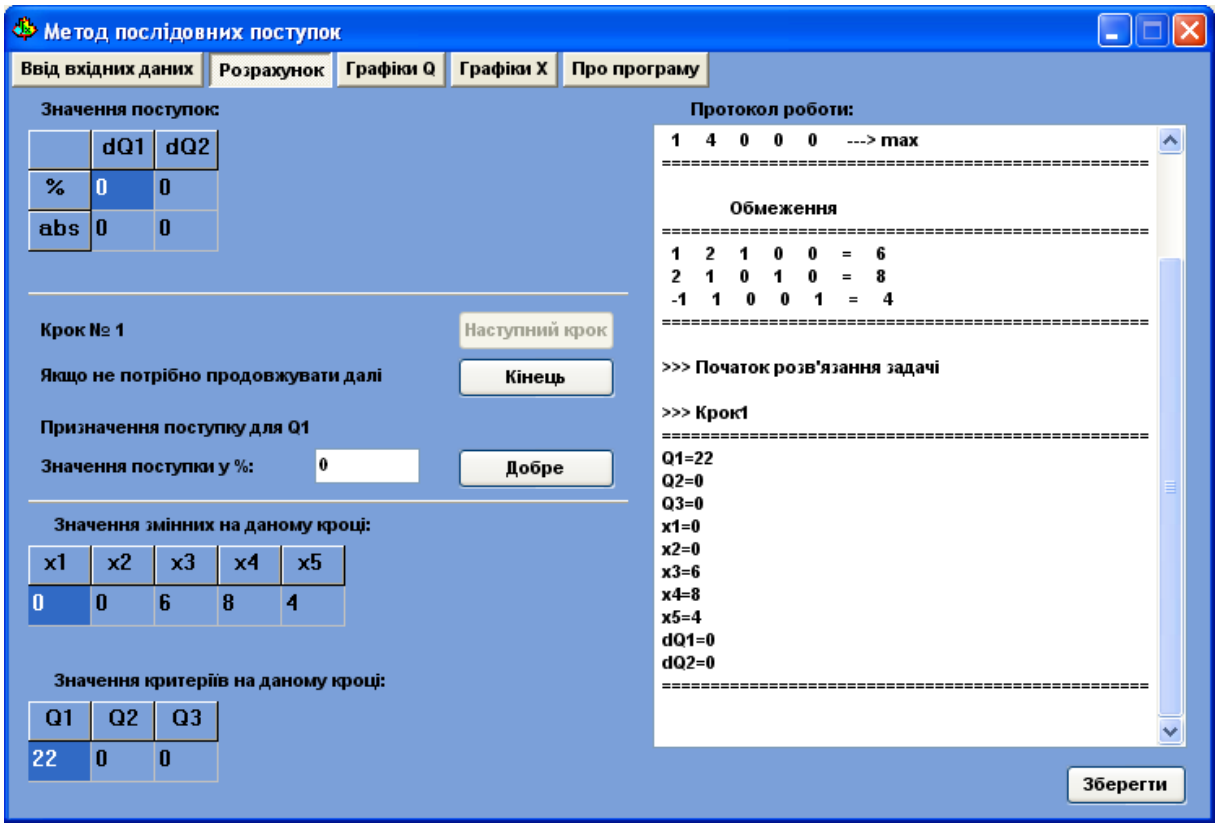

Рисунок 3.6 - Результат виконання першого кроку алгоритму

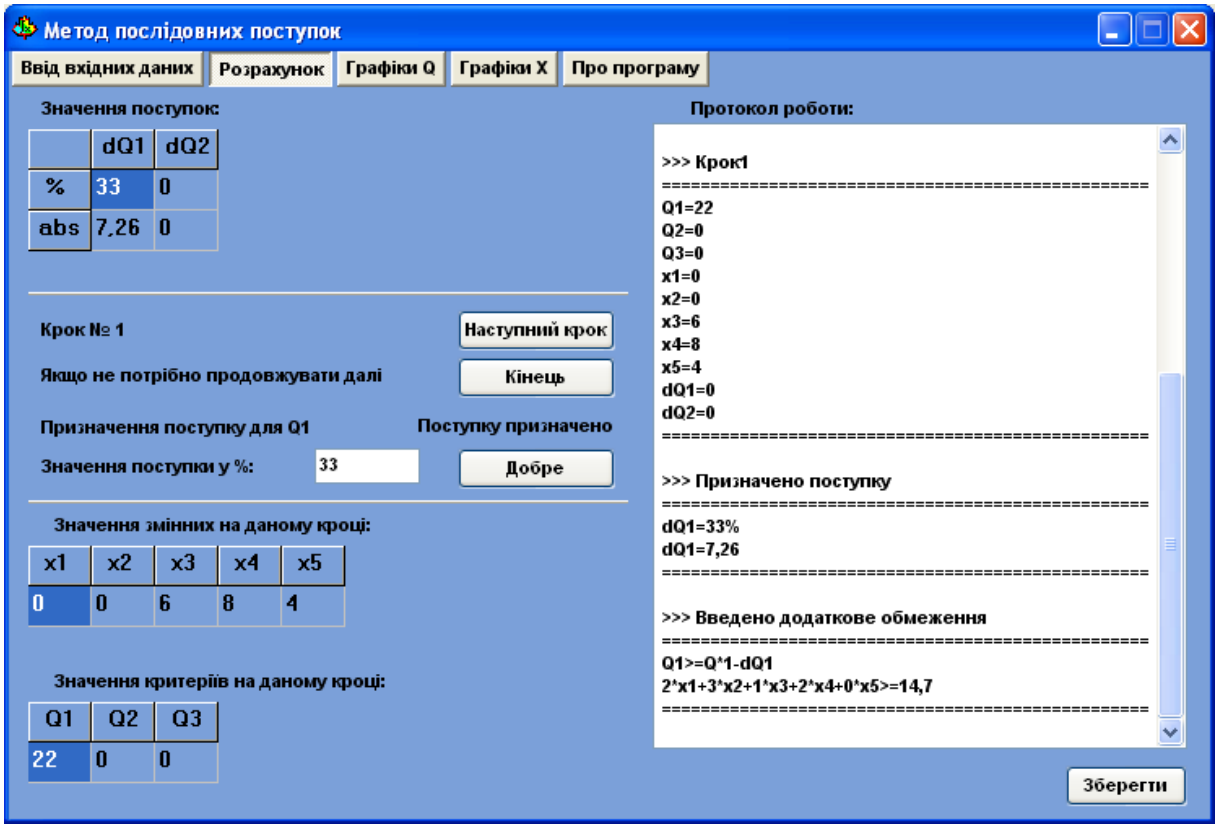

Рисунок 3.7 - Призначення поступок для розв’язку тестового прикладу задачі оптимізації

Послідовно призначаємо всі наступні поступки і доходимо до кінця розв’язку задачі багатокритеріальної оптимізації методом послідовних поступок (рис. 3.8).

Підсистема автоматизації розв’язання задач багатокритеріальної оптимізації дозволяє графічно відобразити результати роботи. На рисунку 3.9 наведено результати зміни значень цільових функцій, а на рисунку 3.10 зміна аргумента.

Графіки та протокол роботи програми можуть бути записані у файли (графічному та текстовому форматі відповідно) для подальшого опрацювання (рис. 3.11).

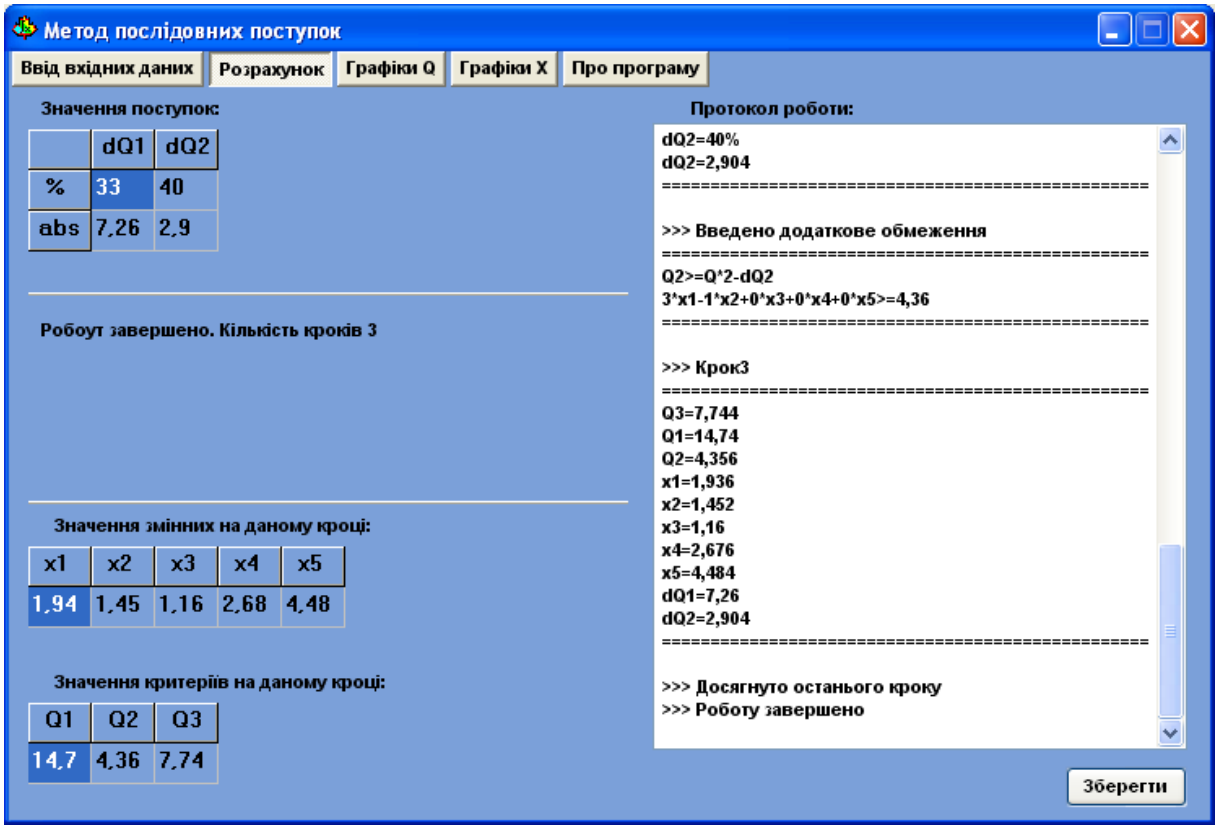

Рисунок 3.8 - Кінцеві результати роботи підсистеми

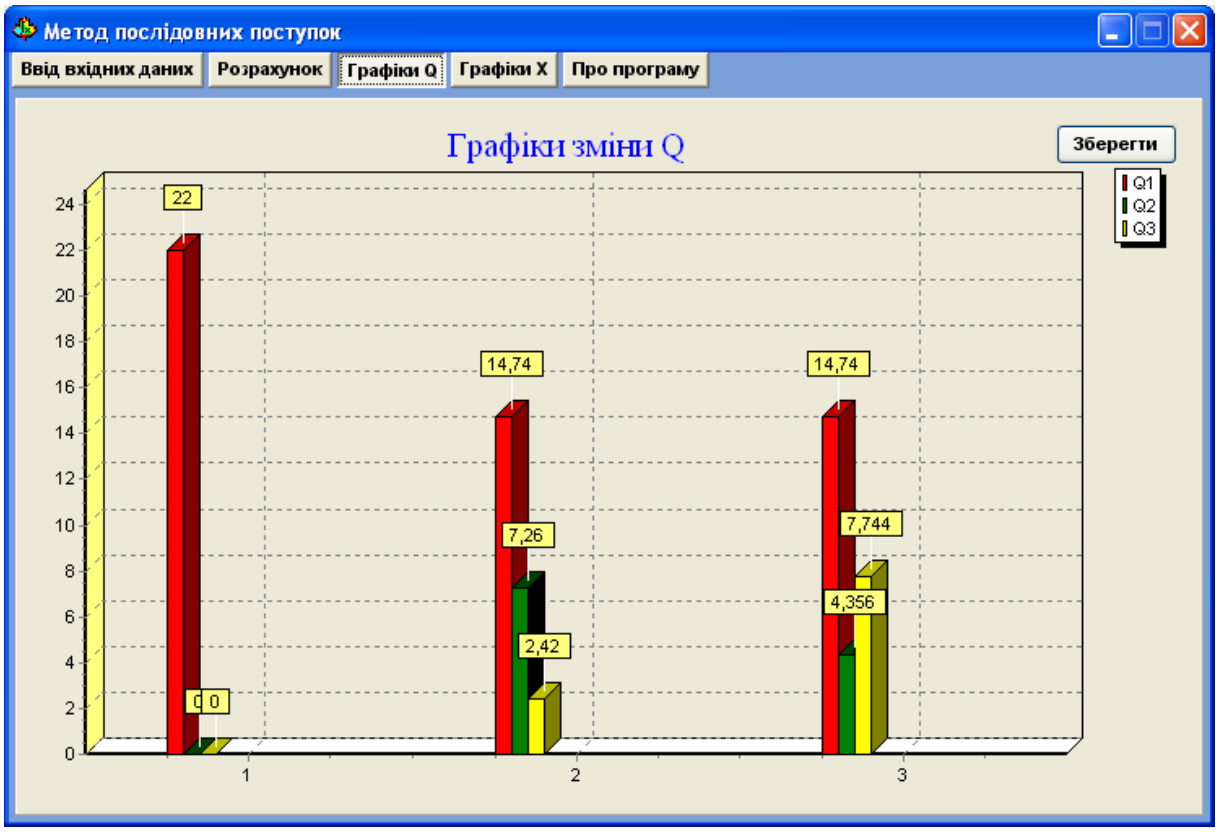

Рисунок 3.9 - Зміна значення цільових функцій Q1, Q2, Q3

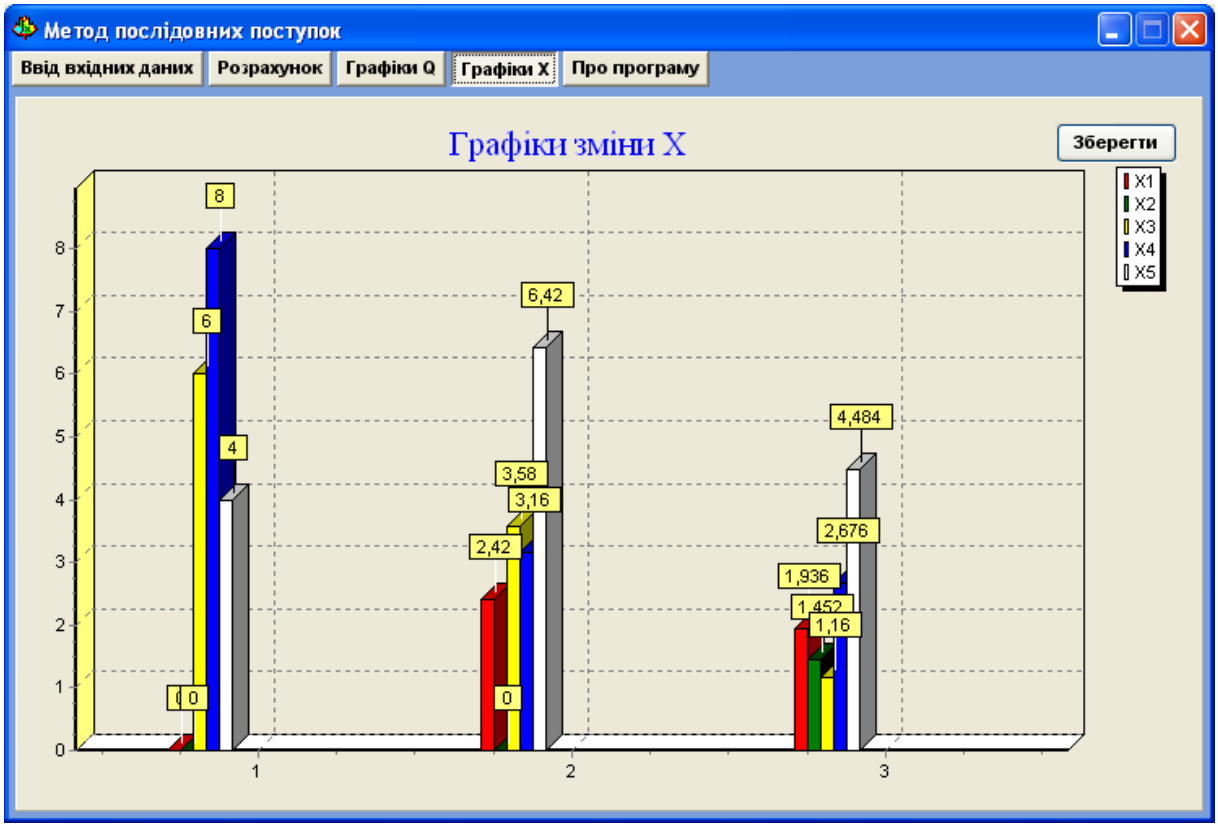

Рисунок 3.10 - Зміна значення аргументів X1, X2, X3, X4, X5

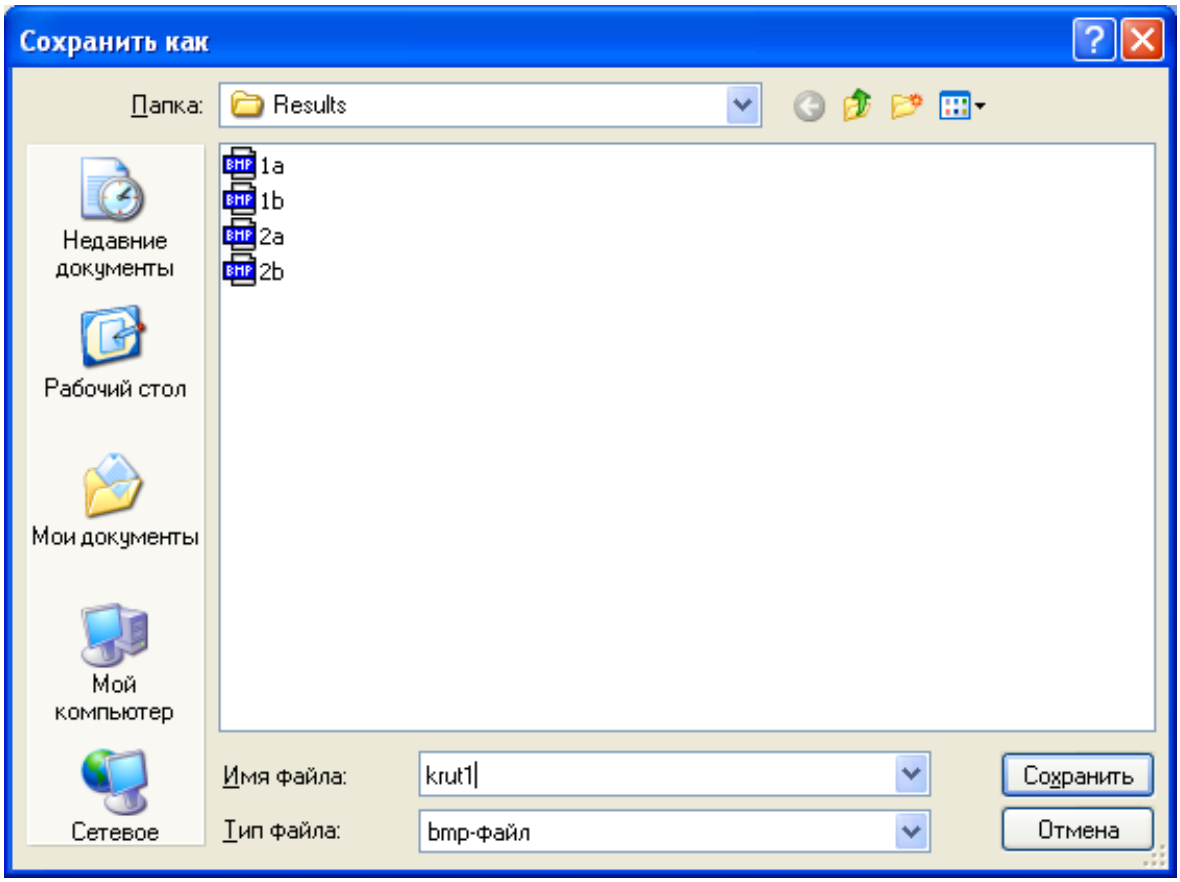

Рисунок 3.11 - Меню збереження графічної інформації у файл

Дана структура даних забезпечує високу ефективність представлення інформації, необхідної для роботи підсистеми та є зручним засобом інтеграції даних у середовище різних підсистем подібного характеру. Вихідний файл включає інформацію про процес розв’язання задачі (протокол роботи підсистеми), де наведені результати основних та допоміжних результатів розрахунків.

Отримані дані співпадають з теоретичними отриманими з літеранурних джерел.

Лістинг файлу з результатами виконання програми:

```
>>> Вхідні дані:
            Критерії
==================================================
 2    3    1    2    0     ---> max
 3    -1    0    0    0     ---> max
 1    4    0    0    0     ---> max
==================================================
            Обмеження
==================================================
 1    2    1    0    0    =    6
 2    1    0    1    0    =    8
 -1    1    0    0    1    =    4
==================================================
>>> Початок розв'язання задачі

>>> Крок1
==================================================
Q1=22
Q2=0
Q3=0
x1=0
x2=0
x3=6
```

x4=8

x5=4

dQ1=0

dQ2=0

==========

>>> Призначено поступку

==========

dQ1=33%

dQ1=7,26

==========

>>> Введено додаткове обмеження

==========

Q1>=Q*1-dQ1

2*x1+3*x2+1*x3+2*x4+0*x5>=14,7

==========

>>> Крок2

==========

Q2=7,26

Q1=14,74

Q3=2,42

x1=2,42

x2=0

x3=3,58

x4=3,16

x5=6,42

dQ1=7,26

dQ2=0

==========

>>> Призначено поступку

==========

dQ2=40%

dQ2=2,904

==========

>>> Введено додаткове обмеження

==========

Q2>=Q*2-dQ2

3*x1-1*x2+0*x3+0*x4+0*x5>=4,36

==========

>>> Крок3

==========

Q3=7,744

Q1=14,74

Q2=4,356

x1=1,936

x2=1,452

x3=1,16

x4=2,676

x5=4,484

dQ1=7,26

dQ2=2,904

==========

>>> Досягнуто останього кроку

>>> Роботу завершено

>>> Графік значень Q успішно записаний в файл E:\OptimMEMS\Results\2a.bmp

>>> Графік значень X успішно записаний в файл E:\OptimMEMS\Results\2b.bmp

>>> Протокол успішно записаний в файл E:\OptimMEMS\Results\2.txt

3.6 Висновок

Описано розлоблене програмне та технічне забезпечення підсистеми для розв’язання задач багатокритеріальної оптимізації.

Наведені результати розв’язку тестових задач багатокритеріальної оптимізації методом послідовних поступок.

Описані особливості інтерфейсу підсистеми розв’язку задач багатокритеріальної оптимізації та алгоритм їх рішення з допомогою розробленої підсистеми.

ВИСНОВКИ

Розроблено структуру підсистеми розв’язку задач багатокритеріальної оптимізації методом послідовних поступок, яка включає підсистему вводу вхідних даних, підсистему розрахунків, підсистему виведення даних та інтерфейс.

Побудова модель оптимізації структури компютерної мережі, яка може включати такі критерії як вартість, надійність, пропускна здатність та інтенсивність відмов з відповідними обмеженнями.

Описано розлоблене програмне та технічне забезпечення підсистеми для розв’язання задач багатокритеріальної оптимізації.

Наведені результати розв’язку тестових задач багатокритеріальної оптимізації методом послідовних поступок.

Описані особливості інтерфейсу підсистеми розв’язку задач багатокритеріальної оптимізації та алгоритм їх рішення з допомогою розробленої підсистеми.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вентцель Е.С. Исследование операций. М.:Советское радио, 1972
2. Кофман А., Фор Р. Займемся исследованием операций. М:Мир, 1966
3. Р.Л. Кини, Х.Райфа Принятие решений при многих критериях: предпочтения и замещения. М.:Радио и связь
4. Таха, Хэмди А. Введение в исследование операций – М.:Мир,2001
5. Р. Штойер. Многокритериальная оптимизация: теория, вычисления, приложения. М.:Наука, 1982
6. Многокритериальная оптимизация. Математические аспекты. М.:Наука, 1989
7. Акулич И. Л. Математическое программирование в примерах и задачах. - М.: Высшая школа, 1986.
8. Алексеев В. М.. Галеев Э. М.. Тихомиров В. М. Сборник задач по оптимизации. Теория, примеры, задачи. - М.: Наука, 1984.
9. Ашманов С. А., Тимохов А. В. Теория оптимизации в задачах и упражнениях. - М.: Наука, 1991.
10. Базара М., Шетти К. Нелинейное программирование. Теория и алгоритмы. - М.: Мир, 1982.
11. Банди Б. Методы оптимизации. Вводный курс. - М.: Радио и связь, 1988.
12. Банди Б. Основы линейного программирования. - М.: Радио и связь, 1989
13. С.А.Исаев. Решение многокритериальных задач. Інтернет-ресурс http://bspu.ab.ru/Docs/~saisa/ga/idea1.html.
14. Розділ „Математика\Optimization Toolbox”. Інтернет-ресурс http://www.matlab.ru/optimiz/index.asp.
15. Е.В. Никульчев. Разработка многокритериальных систем управления динамическими объектами. Інтернет-ресурс http://do.sssu.ru/ito2001/mater/mgapi.html.
16. Пападимитриу Х., Стайглиц К. Комбинаторная оптимизация. Алгоритмы и сложность. - М.: Мир, 1985. - 510 с.

17. Грень Е. Статистические игры и их применение. –М.: Статистика, 1975. – 176 с.

18. Де Гроот М. Оптимальные статистические решения. –М.: Мир, 1974. –492 с.

19. Сухарев А.Г. Оптимальный поиск экстремума. – М.: Изд МГУ, 1975.

20. Черноусько Ф.Л., Меликян. Игровые задачи управления и поиска.–М.: Наука, 1978

21. Гермейер Ю.Б. Введение в теорию исследования операций. М.: Наука, 1971.

22. Жилинскас А.Г. Глобальная оптимизация. Аксиоматика статистических моделей, алгоритмы, применение. –Вильнюс: Мокслас, 1986.

Додаток А

Лістинг програми MainUnit.pas - головного модуля

```
unit MainUnit;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, SimplexUnit, Grids, ComCtrls, ExtCtrls, TeEngine,
  Series, TeeProcs, Chart, jpeg;

type
  TForm1 = class(TForm)
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    TabSheet2: TTabSheet;
    TabSheet3: TTabSheet;
    TabSheet4: TTabSheet;
    TabSheet5: TTabSheet;
    NumberInputPanel: TPanel;
    LabeledEdit1: TLabeledEdit;
    LabeledEdit2: TLabeledEdit;
    LabeledEdit3: TLabeledEdit;
    Button3: TButton;
    MatrixInputPanel: TPanel;
    CrutStringGrid: TStringGrid;
    Label2: TLabel;
    ConsStringGrid: TStringGrid;
    Label3: TLabel;
    Button4: TButton;
```

Memo1: TMemo;

Button2: TButton;

Button1: TButton;

Label4: TLabel;

XStringGrid: TStringGrid;

Label5: TLabel;

Label6: TLabel;

QStringGrid: TStringGrid;

Label7: TLabel;

dQStringGrid: TStringGrid;

Label1: TLabel;

SaveDialog1: TSaveDialog;

Bevel1: TBevel;

Bevel2: TBevel;

dQPanel: TPanel;

Label8: TLabel;

Button5: TButton;

Edit1: TEdit;

Label9: TLabel;

Label11: TLabel;

Label10: TLabel;

Button6: TButton;

QChart: TChart;

Series1: TBarSeries;

Series2: TBarSeries;

Series3: TBarSeries;

Series4: TBarSeries;

Series5: TBarSeries;

Series6: TBarSeries;

Series7: TBarSeries;

Series8: TBarSeries;

Series9: TBarSeries;

Series10: TBarSeries;

XChart: TChart;

BarSeries1: TBarSeries;

BarSeries2: TBarSeries;

BarSeries3: TBarSeries;

BarSeries4: TBarSeries;

BarSeries5: TBarSeries;

BarSeries6: TBarSeries;

BarSeries7: TBarSeries;

BarSeries8: TBarSeries;

BarSeries9: TBarSeries;

BarSeries10: TBarSeries;

Series11: TBarSeries;

Series12: TBarSeries;

Series13: TBarSeries;

Series14: TBarSeries;

Series15: TBarSeries;

SaveDialog2: TSaveDialog;

SaveDialog3: TSaveDialog;

Button7: TButton;

Button8: TButton;

Label13: TLabel;

Label14: TLabel;

Label15: TLabel;

Label16: TLabel;

Label17: TLabel;

Label18: TLabel;

Label12: TLabel;

Label19: TLabel;

Label20: TLabel;

```
    Label21: TLabel;
    Label22: TLabel;
    Label23: TLabel;
    Timer1: TTimer;
    Image1: TImage;
    procedure Button1Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure CrutStringGridKeyPress(Sender: TObject; var Key: Char);
    procedure ConsStringGridKeyPress(Sender: TObject; var Key: Char);
    procedure Button4Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  function GetQ(nQ:word;R:TExtArray):extended;
    procedure Edit1KeyPress(Sender: TObject; var Key: Char);
    procedure Button6Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure Button8Click(Sender: TObject);
    procedure TabSheet5Show(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

  Simplex:TSimplex;

  CrutCount:word;
```

```
    ConsCount:word;
    ZminCount:word;

    Cr:array of TCruterij;
    Con:array of TConstrain;
    X:TExtArray;
    Q:TExtArray;
    dQ:array[1..2] of TExtArray;

    StepNumber:word;

    Ser:array of TBarSeries;

  implementation

  {$R *.dfm}
  {$R WindowsXP.res}

  procedure TForm1.Button1Click(Sender: TObject);
  begin
  if SaveDialog1.Execute then
    begin
    Memo1.Lines.Add('');
    Memo1.Lines.Add('>>> Протокол успішно записаний в файл
'+SaveDialog1.FileName);
    Memo1.Lines.Add('');
    Memo1.Lines.SaveToFile(SaveDialog1.FileName);
    end;
  end;

  procedure TForm1.Button3Click(Sender: TObject);
```

```
var i,j:word;
begin
CrutCount:=StrToInt(LabeledEdit1.Text);
ConsCount:=StrToInt(LabeledEdit2.Text);
ZminCount:=StrToInt(LabeledEdit3.Text);
NumberInputPanel.Enabled:=false;
MatrixInputPanel.Enabled:=true;

//=============================================================
=
CrutStringGrid.ColCount:=ZminCount+3;
CrutStringGrid.RowCount:=CrutCount+1;
for i:=1 to ZminCount do
  begin
  CrutStringGrid.Cells[i,0]:='  x'+IntToStr(i);
  end;
i:=ZminCount+1;
CrutStringGrid.Cells[i,0]:=' ---> ';
i:=i+1;
CrutStringGrid.Cells[i,0]:='M/m';
for i:=1 to CrutCount do
  begin
  CrutStringGrid.Cells[0,i]:='  Q'+IntToStr(i);
  end;
for i:=1 to CrutCount do
  begin
  for j:=1 to ZminCount do
    begin
    CrutStringGrid.Cells[j,i]:=' 0';
    end;
  end;
```

```
      j:=ZminCount+1;
      for i:=1 to CrutCount do
          CrutStringGrid.Cells[j,i]:=' --->';
      j:=j+1;
      for i:=1 to CrutCount do
          CrutStringGrid.Cells[j,i]:='max';

      //=========================================================
=
      ConsStringGrid.ColCount:=ZminCount+3;
      ConsStringGrid.RowCount:=ConsCount+1;
      for i:=1 to ZminCount do
        begin
        ConsStringGrid.Cells[i,0]:='  x'+IntToStr(i);
        end;
      i:=ZminCount+1;
      ConsStringGrid.Cells[i,0]:=' <=> ';
      i:=i+1;
      ConsStringGrid.Cells[i,0]:=' B';
      for i:=1 to ConsCount do
        begin
        ConsStringGrid.Cells[0,i]:='  A'+IntToStr(i);
        end;
      for i:=1 to ConsCount do
        begin
        for j:=1 to ZminCount do
          begin
          ConsStringGrid.Cells[j,i]:=' 0';
          end;
        end;
      j:=ZminCount+1;
```

```
      for i:=1 to ConsCount do
          ConsStringGrid.Cells[j,i]:='  =';
      j:=j+1;
      for i:=1 to ConsCount do
          ConsStringGrid.Cells[j,i]:=' 0';

      //============================================================
=
      xStringGrid.ColCount:=ZminCount;
      for i:=0 to (ZminCount-1) do
        begin
        xStringGrid.Cells[i,0]:='  x'+IntToStr(i+1);
        xStringGrid.Cells[i,1]:=' 0'
        end;

      //============================================================
=
      QStringGrid.ColCount:=CrutCount;
      for i:=0 to (CrutCount-1) do
        begin
        QStringGrid.Cells[i,0]:='  Q'+IntToStr(i+1);
        QStringGrid.Cells[i,1]:=' 0'
        end;

      //============================================================
=
      dQStringGrid.ColCount:=CrutCount;
      for i:=0 to (CrutCount) do
        begin
        dQStringGrid.Cells[i+1,0]:=' dQ'+IntToStr(i+1);
        dQStringGrid.Cells[i+1,1]:=' 0';
```

```
  dQStringGrid.Cells[i+1,2]:=' 0';
  end;
dQStringGrid.Cells[0,1]:='  %';
dQStringGrid.Cells[0,2]:=' abs';
end;

procedure TForm1.CrutStringGridKeyPress(Sender: TObject; var Key: Char);
begin
if not ((key='0') or (key='1') or (key='2') or (key='3') or
      (key='4') or (key='5') or (key='6') or (key='7') or
      (key='8') or (key='9') or (key='-') or (key='i') or
      (key='m') or (key='a') or (key='x') or (key='n') or
      (key=#13) or (key=#8))
        then key:=#0;
end;

procedure TForm1.ConsStringGridKeyPress(Sender: TObject; var Key: Char);
begin
if not ((key='0') or (key='1') or (key='2') or (key='3') or
      (key='4') or (key='5') or (key='6') or (key='7') or
      (key='8') or (key='9') or (key='-') or (key='=') or
      (key='>') or (key='<') or
      (key=#13) or (key=#8))
        then key:=#0;
end;

procedure TForm1.Button4Click(Sender: TObject);
var i,j:word;
   st:string;
begin
MatrixInputPanel.Enabled:=false;
```

```
SetLength(Cr,CrutCount);
SetLength(Con,ConsCount);

for i:=0 to (CrutCount-1) do
  begin
  SetLength(Cr[i].Cr,ZminCount);
  for j:=0 to (ZminCount-1) do
    begin
    Cr[i].Cr[j]:=StrToFloat(CrutStringGrid.Cells[j+1,i+1]);
    end;
  end;
for i:=0 to (CrutCount-1) do
  begin
  if CrutStringGrid.Cells[ZminCount+2,i+1]='min' then Cr[i].F:=false
    else Cr[i].F:=true;
  end;

for i:=0 to (ConsCount-1) do
  begin
  SetLength(Con[i].A,ZminCount);
  for j:=0 to (ZminCount-1) do
    begin
    Con[i].A[j]:=StrToFloat(ConsStringGrid.Cells[j+1,i+1]);
    end;
  end;
for i:=0 to (ConsCount-1) do
  begin
  Con[i].B:=StrToFloat(ConsStringGrid.Cells[ZminCount+2,i+1]);
  end;
for i:=0 to (ConsCount-1) do
```

```
        begin
        if ConsStringGrid.Cells[ZminCount+1,i+1]='<' then Con[i].Sign:=Less
          else if ConsStringGrid.Cells[ZminCount+1,i+1]='>' then Con[i].Sign:=Greater
            else Con[i].Sign:=Equal;
        end;

      Memo1.Lines.Add('>>> Вхідні дані:');
      Memo1.Lines.Add('');
      Memo1.Lines.Add('            Критерії');
      Memo1.Lines.Add('==========================================
======');
      for i:=0 to (CrutCount-1) do
        begin
        st:='  ';
        for j:=0 to (ZminCount-1) do
          begin
          st:=st+(FloatToStr(Cr[i].Cr[j])+'     ');
          end;
        st:=st+' ---> ';
        if Cr[i].F then st:=st+'max'
          else st:=st+'min';
        Memo1.Lines.Add(st);
        end;
      Memo1.Lines.Add('==========================================
======');
      Memo1.Lines.Add('');
      Memo1.Lines.Add('            Обмеження');
      Memo1.Lines.Add('==========================================
======');
      for i:=0 to (ConsCount-1) do
        begin
```

```
        st:='  ';
        for j:=0 to (ZminCount-1) do
          begin
          st:=st+(FloatToStr(Con[i].A[j])+'     ');
          end;
        if Con[i].Sign=Less then st:=st+'<     ';
        if Con[i].Sign=Greater then st:=st+'>     ';
        if Con[i].Sign=Equal then st:=st+'=     ';
        st:=st+FloatToStr(Con[i].B);
        Memo1.Lines.Add(st);
        end;
      Memo1.Lines.Add('==========================================
======');
      Memo1.Lines.Add('');
      Memo1.Lines.Add('>>> Початок розв"язання задачі');
      SetLength(dQ[1],CrutCount);
      SetLength(dQ[2],CrutCount);
      for i:=0 to (CrutCount-1) do
        begin
        dQ[1,i]:=0;
        dQ[2,i]:=0;
        end;
      SetLength(Q,CrutCount);
      StepNumber:=0;
      TabSheet2.Enabled:=true;
      PageControl1.ActivePage:=TabSheet2;

      SetLength(Ser,CrutCount);

      for i:=0 to (CrutCount-1) do
        begin
```

```
        QChart.Series[i].ShowInLegend:=true;
        end;
      for i:=0 to (ZminCount-1) do
        begin
        XChart.Series[i].ShowInLegend:=true;
        end;
      end;

      procedure TForm1.Button2Click(Sender: TObject);
      var i:word;
      begin
      StepNumber:=StepNumber+1;
      if (StepNumber=0) then Label1.Caption:='Крок № 0 (початок роботи)'
        else Label1.Caption:='Крок № '+IntToStr(StepNumber);
      Label11.Visible:=false;
      Memo1.Lines.Add('');
      Memo1.Lines.Add('>>> Крок'+IntToStr(StepNumber));
      Memo1.Lines.Add('==========================================
======');
      Simplex:=TSimplex.Create(Cr[StepNumber-1].Cr,Cr[StepNumber-1].F);
      for i:=0 to (ConsCount-1) do
        begin
        Simplex.AddCons(Con[i].B,Con[i].A,Con[i].Sign);
        end;
      if (Simplex.Solve=SIMPLEX_DONE) then
        begin

Memo1.Lines.Add('Q'+IntToStr(StepNumber)+'='+FloatToStr(Simplex.GetMin));
        Q[StepNumber-1]:=Simplex.GetMin;
        SetLength(X,0);
        X:=Simplex.GetSolution;
```

```
      for i:=1 to CrutCount do
        begin
        if i<>StepNumber then
          begin
          Memo1.Lines.Add('Q'+IntToStr(i)+'='+FloatToStr(GetQ(i,X)));
          Q[i-1]:=GetQ(i,X);
          end;
        QChart.Series[i-1].AddXY(StepNumber,Q[i-1]);
        QStringGrid.Cells[i-1,1]:=FloatToStrF(GetQ(i,X),ffGeneral,3,3);
        end;
      for i:=0 to (ZminCount-1) do
        begin
        XChart.Series[i].AddXY(StepNumber,X[i]);
        Memo1.Lines.Add('x'+IntToStr(i+1)+'='+FloatToStr(X[i]));
        xStringGrid.Cells[i,1]:=FloatToStrF(X[i],ffGeneral,3,3);
        end;
      for i:=1 to (CrutCount-1) do
        begin
        Memo1.Lines.Add('dQ'+IntToStr(i)+'='+FloatToStr(dQ[2,i-1]));
        dQStringGrid.Cells[i,1]:=FloatToStrF(dQ[1,i-1],ffGeneral,3,3);
        dQStringGrid.Cells[i,2]:=FloatToStrF(dQ[2,i-1],ffGeneral,3,3);
        end;
      end
    else
      begin
      if Simplex.Solve=(SIMPLEX_NO_SOLUTION) then
        Memo1.Lines.Add('>>> Неможиво утворити базис');
      exit;
      end;
    Memo1.Lines.Add('==========================================
======');
```

```
Memo1.Lines.Add('');
Simplex.Free;

if (StepNumber=CrutCount) then
  begin
  dQPanel.Visible:=false;
  Label8.Visible:=false;
  Label9.Visible:=false;
  Label10.Visible:=false;
  Label11.Visible:=false;
  Edit1.Visible:=false;
  Button5.Visible:=false;
  Button2.Visible:=false;
  Button6.Visible:=false;
  Label1.Caption:='Робоут завершено. Кількість кроків
'+IntToStr(StepNumber);
  Memo1.Lines.Add('>>> Досягнуто останього кроку');
  Memo1.Lines.Add('>>> Роботу завершено');
  end;

dQPanel.Visible:=true;
dQPanel.Enabled:=true;
Button2.Enabled:=false;
Label8.Caption:='Призначення поступку для Q'+IntToStr(StepNumber);

end;

function TForm1.GetQ(nQ:word;R:TExtArray):extended;
var i:word;
   s:extended;
begin
```

```
s:=0;
for i:=0 to (ZminCount-1) do
  begin
  s:=s+Cr[nq-1].Cr[i]*R[i];
  end;
Result:=s;
end;

procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
if not ((key='0') or (key='1') or (key='2') or (key='3') or
      (key='4') or (key='5') or (key='6') or (key='7') or
      (key='8') or (key='9') or (key=#13) or (key=#8))
        then key:=#0;
end;

procedure TForm1.Button6Click(Sender: TObject);
var i:word;
begin
Memo1.Lines.Add('>>> Припинено користувачем');
Memo1.Lines.Add('=========================================
======');
Memo1.Lines.Add('Остаточні результати');
Memo1.Lines.Add('=========================================
======');
for i:=1 to CrutCount do
    begin
    Memo1.Lines.Add('Q'+IntToStr(i)+'='+FloatToStr(GetQ(i,X)));
    end;
for i:=0 to (ZminCount-1) do
    begin
```

```
        Memo1.Lines.Add('x'+IntToStr(i+1)+'='+FloatToStr(X[i]));
        end;
    Memo1.Lines.Add('==========================================
======');
    end;

    procedure TForm1.Button5Click(Sender: TObject);
    var i:word;
       st:string;
    begin
    dQ[1,StepNumber-1]:=StrToInt(Edit1.Text);
    dQ[2,StepNumber-1]:=(Q[StepNumber-1]*dQ[1,StepNumber-1])/100;
    dQStringGrid.Cells[StepNumber,1]:=FloatToStrF(dQ[1,StepNumber-
1],ffGeneral,3,3);
    dQStringGrid.Cells[StepNumber,2]:=FloatToStrF(dQ[2,StepNumber-
1],ffGeneral,3,3);
    Button2.Enabled:=true;
    dQPanel.Enabled:=false;
    Label11.Visible:=true;
    Memo1.Lines.Add('>>> Призначено поступку');
    Memo1.Lines.Add('==========================================
======');
    Memo1.Lines.Add('dQ'+IntToStr(StepNumber)+'='+Edit1.Text+'%');
    Memo1.Lines.Add('dQ'+IntToStr(StepNumber)+'='+FloatToStr(dQ[2,StepNumber
-1]));
    Memo1.Lines.Add('==========================================
======');

    SetLength(Con,Length(Con)+1);
    ConsCount:=ConsCount+1;
    SetLength(Con[ConsCount-1].A,ZminCount+1);
```

```
      for i:=0 to (ZminCount-1) do
        begin
        Con[ConsCount-1].A[i]:=Cr[StepNumber-1].Cr[i];
        end;
      Con[ConsCount-1].A[ZminCount]:=1;

      Con[ConsCount-1].Sign:=Greater;
      Con[ConsCount-1].B:=Q[StepNumber-1]-dQ[2,StepNumber-1];

      for i:=StepNumber to (CrutCount-1) do
        begin
        SetLength(Cr[i].Cr,Length(Cr[i].Cr)+1);
        Cr[i].Cr[Length(Cr[i].Cr)-1]:=-50000;
        end;

      Memo1.Lines.Add('');
      Memo1.Lines.Add('>>> Введено додаткове обмеження');
      Memo1.Lines.Add('=========================================
======');
      Memo1.Lines.Add('Q'+IntToStr(StepNumber)+'>='+'Q*'+IntToStr(StepNumber)+
                '-'+'dQ'+IntToStr(StepNumber));
      st:='';
      st:=st+FloatToStrF(Cr[StepNumber-1].Cr[0],ffGeneral,3,3)+'*x1';
      for i:=1 to (ZminCount-1) do
        begin
        if (Cr[StepNumber-1].Cr[i]>=0) then st:=st+'+'+FloatToStrF(Cr[StepNumber-
1].Cr[i],ffGeneral,3,3)
           else st:=st+FloatToStrF(Cr[StepNumber-1].Cr[i],ffGeneral,3,3);
        st:=st+'*x'+IntToStr(i+1);
        end;
      st:=st+'>=';
```

```
      st:=st+FloatToStrF(Q[StepNumber-1]-dQ[2,StepNumber-1],ffGeneral,3,3);
      Memo1.Lines.Add(st);
      Memo1.Lines.Add('=========================================
======');
      end;

      procedure TForm1.Button7Click(Sender: TObject);
      begin
      if SaveDialog2.Execute then
        begin
        QChart.SaveToBitmapFile(SaveDialog2.FileName);
        Memo1.Lines.Add('');
        Memo1.Lines.Add('>>> Графік значень Q успішно записаний в файл
'+SaveDialog2.FileName);
        Memo1.Lines.Add('');
        end;
      end;

      procedure TForm1.Button8Click(Sender: TObject);
      begin
      if SaveDialog3.Execute then
        begin
        XChart.SaveToBitmapFile(SaveDialog3.FileName);
        Memo1.Lines.Add('');
        Memo1.Lines.Add('>>> Графік значень X успішно записаний в файл
'+SaveDialog3.FileName);
        Memo1.Lines.Add('');
        end;
      end;

      procedure TForm1.TabSheet5Show(Sender: TObject);
```

```
begin
Image1.Visible:=false;
Label12.Top:=Label12.Top+440;
Label13.Top:=Label13.Top+440;
Label14.Top:=Label14.Top+440;
Label15.Top:=Label15.Top+440;
Label16.Top:=Label16.Top+440;
Label17.Top:=Label17.Top+440;
Label18.Top:=Label18.Top+440;
Label19.Top:=Label19.Top+440;
Label20.Top:=Label20.Top+440;
Label21.Top:=Label21.Top+440;
Label22.Top:=Label22.Top+440;
Label23.Top:=Label23.Top+440;
Timer1.Enabled:=true;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
if (Label20.Top>20) then
  begin
  Label12.Top:=Label12.Top-2;
  Label13.Top:=Label13.Top-2;
  Label14.Top:=Label14.Top-2;
  Label15.Top:=Label15.Top-2;
  Label16.Top:=Label16.Top-2;
  Label17.Top:=Label17.Top-2;
  Label18.Top:=Label18.Top-2;
  Label19.Top:=Label19.Top-2;
  Label20.Top:=Label20.Top-2;
  Label21.Top:=Label21.Top-2;
```

```
    Label22.Top:=Label22.Top-2;
    Label23.Top:=Label23.Top-2;
    end
  else
    begin
    Timer1.Enabled:=false;
    Image1.Visible:=true;
    end;

end;

end.
```

Додаток Б

Лістинг програми SimplexUnit.pas – модулядля роботи симплекс методу

```
unit SimplexUnit;

interface

const
  SIMPLEX_DONE = 0;
  SIMPLEX_NO_SOLUTION = 1;
  SIMPLEX_NO_BOTTOM = 2;
  SIMPLEX_NEXT_STEP = 3;

type
  TOperation = (Equal,Less,Greater);
  TExtArray = array of extended;
  TConstrain = record
    A     :  TExtArray;
    B     :  extended;
    Sign  :  TOperation;
  end;

  TCruterij = record
    Cr     :  TExtArray;
    F      :  boolean;
  end;

  TSimplex = class
    M,N   :  integer;
    Cons  :  array of TConstrain;
    C     :  TExtArray;
```

```
  L     :  extended;
  Basis :  array of integer;
  Max   :  boolean;

  Constructor Create(_C:TExtArray; Maximize:boolean=false);
  Constructor CreateBasis(const Simplex:TSimplex);
  Constructor Copy(const Simplex:TSimplex);

  Procedure AddCons(_B:extended; _A:TExtArray; Sign:TOperation);

  Procedure SetAllLengths(Len:integer);
  Function  SimplexStep:integer;
  Function  CheckBasis:boolean;
  Procedure Normalize;
  Procedure MulString(Number:integer; Value:extended);
  Procedure AddString(Num1,Num2:integer; Value:extended);

  Function Solve:integer;
  Function GetMin:extended;
  Function GetSolution:TExtArray;

  Destructor Free;
end;

TIntSimplex = class(TSimplex)
  CurX       : TExtArray;
  CurL       : extended;
  CurFound   : boolean;
  Constructor Create(_C:TExtArray; Maximize:boolean=false);
  Procedure DelLastCons;
  Function  IntSolve:integer;
```

```
    Function  GetIntMin:extended;
    Function  IsInteger(value:extended):boolean;
    Function  GetIntSolution:TExtArray;
    Function  SearchCons(_B:extended;_A:TExtArray):integer;
  end;

implementation

uses Math;

procedure TSimplex.AddCons(_B: extended; _A: TExtArray; Sign: TOperation);
var
  j  :  integer;
begin
  if (Length(_A)>N) then SetAllLengths(Length(_A));
  inc(M);
  SetLength(Cons,M);
  if ((_B=0) and (Sign=Less)) then Sign:=Equal;
  Cons[M-1].B:=_B;
  Cons[M-1].Sign:=Sign;
  SetLength(Cons[M-1].A,N);
  for j:=0 to Length(_A)-1 do Cons[M-1].A[j]:=_A[j];
  if Length(_A)<N then for j:=Length(_A) to N-1 do Cons[M-1].A[j]:=0;
end;

procedure TSimplex.AddString(Num1, Num2: integer; Value: extended);
var
  j   :  integer;
begin
  for j:=0 to N-1 do
Cons[Num1].A[j]:=Cons[Num1].A[j]+Cons[Num2].A[j]*Value;
```

```
  Cons[Num1].B:=Cons[Num1].B+Cons[Num2].B*Value;
end;

function TSimplex.CheckBasis: boolean;
var
  i,j,k   : integer;
  f       : boolean;
begin
  SetLength(Basis,M);
  for i:=0 to M-1 do Basis[i]:=-1;
  for j:=0 to N-1 do begin
    f:=true;
    k:=-1;
    i:=0;
    while (f and (i<M)) do begin
      if ((Cons[i].A[j]<>0) and (Cons[i].A[j]<>1)) then f:=false;
      if (Cons[i].A[j]=1) then begin
        if (k=-1) then k:=i
        else f:=false;
      end;
      inc(i);
    end;
    if (f and (k<>-1)) then Basis[k]:=j;
  end;
  f:=true;
  for i:=0 to M-1 do f:=f and (Basis[i]<>-1);
  Result:=f;
end;

constructor TSimplex.Create(_C: TExtArray; Maximize:boolean);
var
```

```
  j   :  integer;
begin
  N:=Length(_C);
  M:=0;
  SetLength(C,N);
  Max:=Maximize;
  if (Maximize) then for j:=0 to N-1 do C[j]:=-_C[j]
  else for j:=0 to N-1 do C[j]:=_C[j];
  Max:=Maximize;
end;

constructor TSimplex.Copy(const Simplex: TSimplex);
var
  i,j     :  integer;
begin
  M:=Simplex.M;
  N:=Simplex.N;
  SetLength(Cons,M);
  SetLength(Basis,M);
  SetLength(C,N);
  Max:=Simplex.Max;
  for i:=0 to M-1 do begin
    SetLength(Cons[i].A,N);
    Basis[i]:=-1;
    for j:=0 to N-1 do Cons[i].A[j]:=Simplex.Cons[i].A[j];
    Cons[i].B:=Simplex.Cons[i].B;
    Cons[i].Sign:=Simplex.Cons[i].Sign;
  end;
  for i:=0 to Simplex.N-1 do C[i]:=Simplex.C[i];
end;
```

```
constructor TSimplex.CreateBasis(const Simplex: TSimplex);
var
  i,j     :  integer;
begin
  M:=Simplex.M;
  N:=Simplex.N;
  SetLength(Cons,M);
  SetLength(Basis,M);
  for i:=0 to M-1 do begin
    SetLength(Cons[i].A,N);
    for j:=0 to N-1 do Cons[i].A[j]:=Simplex.Cons[i].A[j];
    Cons[i].B:=Simplex.Cons[i].B;
    Cons[i].Sign:=equal;
  end;
  for i:=0 to M-1 do begin
    if (Simplex.Basis[i]<>-1) then Basis[i]:=Simplex.Basis[i]
    else begin
      SetAllLengths(N+1);
      for j:=0 to M-1 do Cons[j].A[N-1]:=0;
      Cons[i].A[N-1]:=1;
    end;
  end;
  SetLength(C,N);
  for i:=0 to Simplex.N-1 do C[i]:=0;
  for i:=Simplex.N to N-1 do C[i]:=1;
end;

destructor TSimplex.Free;
begin
   SetLength(C,0);
   SetLength(Basis,0);
```

```
  SetLength(Cons,0);
  M:=0;
  N:=0;
end;

function TSimplex.GetMin: extended;
var
  i   : integer;
begin
  L:=0;
  for i:=0 to M-1 do if (Basis[i]<N) then begin
    L:=L+C[Basis[i]]*Cons[i].B;
  end;
  if (Max) then Result:=-L
  else Result:=L;
end;

function TSimplex.GetSolution: TExtArray;
var
 Solution   : TExtArray;
 i,j        : integer;
begin
 SetLength(Solution,N);
 for j:=0 to N-1 do begin
   Solution[j]:=0;
   i:=0;
   while ((i<M) and (Basis[i]<>j)) do inc(i);
   if ((Basis[i]=j) and (i<M)) then Solution[j]:=Cons[i].B;
 end;
 Result:=Solution;
end;
```

```
procedure TSimplex.MulString(Number: integer; Value: extended);
var
 j   :  integer;
begin
  for j:=0 to N-1 do Cons[Number].A[j]:=Cons[Number].A[j]*Value;
  Cons[Number].B:=Cons[Number].B*Value;
end;

procedure TSimplex.Normalize;
var
  i   : integer;
begin
  for i:=0 to M-1 do if (Cons[i].Sign<>Equal) then begin
    SetAllLengths(N+1);
    if (Cons[i].Sign=Greater) then Cons[i].A[N-1]:=-1
    else Cons[i].A[N-1]:=1;
  end;
end;

procedure TSimplex.SetAllLengths(Len: integer);
var
  i, j  :  integer;
  OldN  :  integer;
begin
  OldN:=N;
  N:=Len;
  SetLength(C,N);
  for i:=0 to M-1 do SetLength(Cons[i].A,N);
  if (OldN<N) then begin
    for j:=OldN to N-1 do begin
```

```
      C[j]:=0;
      for i:=0 to M-1 do Cons[i].A[j]:=0;
    end;
  end;
end;

function TSimplex.SimplexStep: integer;
var
  Delta   :  TExtArray;
  i,j     :  integer;
  MaxVal  :  extended;
  MaxNum  :  integer;
  AB      :  extended;
  ABLine  :  integer;
begin
  SetLength(Delta,N);
  MaxVal:=0.1e-12;
  MaxNum:=N;
  for j:=0 to N-1 do begin
    Delta[j]:=0;
    for i:=0 to M-1 do Delta[j]:=Delta[j]+Cons[i].A[j]*C[Basis[i]];
    Delta[j]:=Delta[j]-C[j];
    if (Delta[j]>MaxVal) then begin
      MaxVal:=Delta[j];
      MaxNum:=j;
    end;
  end;
  if (MaxNum=N) then Result:=SIMPLEX_DONE
  else begin
    AB:=0;
    ABLine:=M;
```

```
        for i:=0 to M-1 do if (Cons[i].A[maxnum]<>0) then begin
          if ((Cons[i].B/Cons[i].A[maxnum]>=0) and
((Cons[i].B/Cons[i].A[maxnum]<AB) or (ABLine=M))) then begin
            AB:=Cons[i].B/Cons[i].A[maxnum];
            ABLine:=i;
          end;
        end;
        if (ABLine=M) then Result:=SIMPLEX_NO_BOTTOM
        else begin
          for i:=0 to M-1 do if (i<>ABline) then AddString(i,ABline,-
Cons[i].A[maxnum]/Cons[ABline].A[maxnum]);
          MulString(ABline,1/Cons[ABline].A[maxnum]);
          Basis[ABline]:=MaxNum;
          Result:=SIMPLEX_NEXT_STEP;
          L:=0;
          for i:=0 to M-1 do begin
            L:=L+C[Basis[i]]*Cons[i].B;
          end;
        end;
      end;
    end;

    function TSimplex.Solve: integer;
    var
      OldN      : integer;
      i,j       : integer;
      Simplex   : TSimplex;
      f         : boolean;
      Step      : integer;
    begin
      OldN:=N;
```

```
  Normalize;
  f:=false;
  if (not CheckBasis) then begin
    Simplex:=TSimplex.CreateBasis(self);
    Simplex.Solve;
    f:=Simplex.GetMin<>0;
    if (not f) then for i:=0 to M-1 do begin
      for j:=0 to N-1 do Cons[i].A[j]:=Simplex.Cons[i].A[j];
      Cons[i].B:=Simplex.Cons[i].B;
      Basis[i]:=Simplex.Basis[i];
    end;
    Simplex.Free;
  end;
  if (f) then Step:=SIMPLEX_NO_SOLUTION
  else repeat
    Step:=SimplexStep;
  until (Step<>SIMPLEX_NEXT_STEP);
  SetAllLengths(OldN);
  Result:=Step;
end;

constructor TIntSimplex.Create(_C:TExtArray; Maximize:boolean=false);
begin
  CurFound:=false;
  inherited;
end;

procedure TIntSimplex.DelLastCons();
```

```
begin
  dec(M);
  SetLength(Cons,M);
end;

function TIntSimplex.GetIntMin: extended;
begin
  Result:=CurL;
end;

function TIntSimplex.GetIntSolution: TExtArray;
begin
  Result:=CurX;
end;

function TIntSimplex.IsInteger(Value:extended):boolean;
begin
  Result:=((Value=floor(Value)) or (Value=ceil(Value)));
end;

function TIntSimplex.IntSolve: integer;
var
  i        :  integer;
  OldN     :  integer;
  FractCol  :  integer;
  TmpX     :  TExtArray;
  TmpCons   :  TExtArray;
  Simplex   :  TSimplex;
```

```
  NewValue : extended;
  OldCons  : integer;
  OldSign  : TOperation;
begin
  OldSign:=Equal;
  SetLength(TmpX,1);
  Simplex:=TSimplex.Copy(self);
  if (Simplex.Solve=SIMPLEX_DONE) then begin
   if (not CurFound or ((Simplex.GetMin<CurL) and not Max) or
((Simplex.GetMin>CurL) and Max)) then begin
    TmpX:=Simplex.GetSolution;
    i:=0;
    while ((i<N) and IsInteger(TmpX[i])) do inc(i);
    FractCol:=i;
    if (FractCol<>N) then begin
     OldN:=N;
     SetLength(TmpCons,OldN);
     for i:=0 to N-1 do TmpCons[i]:=0;
     TmpCons[FractCol]:=1;
     NewValue:=floor(TmpX[FractCol]);
     OldCons:=SearchCons(NewValue,TmpCons);
     if (OldCons=-1) then AddCons(NewValue,TmpCons,less)
     else begin
      OldSign:=Cons[OldCons].Sign;
      Cons[OldCons].Sign:=Equal;
     end;
     IntSolve;
```

```
      if (OldCons=-1) then DelLastCons
      else Cons[OldCons].Sign:=OldSign;


      NewValue:=ceil(TmpX[FractCol]);
      OldCons:=SearchCons(NewValue,TmpCons);
      if (OldCons=-1) then AddCons(NewValue,TmpCons,greater)
      else begin
        OldSign:=Cons[OldCons].Sign;
        Cons[OldCons].Sign:=Equal;
      end;
      IntSolve;
      if (OldCons=-1) then DelLastCons
      else Cons[OldCons].Sign:=OldSign;
      SetAllLengths(OldN);
    end
    else begin
      CurX:=Simplex.GetSolution;
      CurL:=Simplex.GetMin;
      CurFound:=true;
    end;
   end;
  end;
  Simplex.Free;
  if (CurFound) then Result:=SIMPLEX_DONE
  else Result:=SIMPLEX_NO_SOLUTION;
end;
```

```
function TIntSimplex.SearchCons(_B: extended; _A: TExtArray): integer;
var
 i,j    :  integer;
 f      :  boolean;
begin
 i:=0;
 f:=false;
 while ((i<M) and not f) do begin
   f:=(_B=Cons[i].B);
   if (f) then for j:=0 to N-1 do f:=f and (_A[j]=Cons[i].A[j]);
   if (not f) then inc(i);
 end;
 if (i=M) then Result:=-1
 else Result:=i;
end;

end.
```