Київський національний університет імені Тараса Шевченка
Факультет комп'ютерних наук та кібернетики
Кафедра дослідження операцій

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА
за спеціальністю 113 «Прикладна математика»
на тему:

«Оптимальне резервування при кількох обмеженнях»

Виконавець:
Студент групи
ПМ-2 ОКР Магістр
Артюхов Роман Вячеславович

Науковий керівник:
професор, доктор
фізико-математичних наук
Мацак Іван Каленикович

Роботу розглянуто й допущено до захисту на засіданні кафедри дослідження операцій від «__»_____2021 року, протокол №____.

Завідувач кафедри проф. Іксанов О.М. _____
(підпис)

Київ - 2021

ЗМІСТ

ВСТУП

Одним з ефективних методів підвищення надійності складних систем є резервування. Проте, завжди виникають питання. Що резервувати в першу чергу? Який резерв необхідний? Як найраціональніше розподілити ресурси підвищення надійності між підсистемами? Останнє питання є найбільш актуальним.

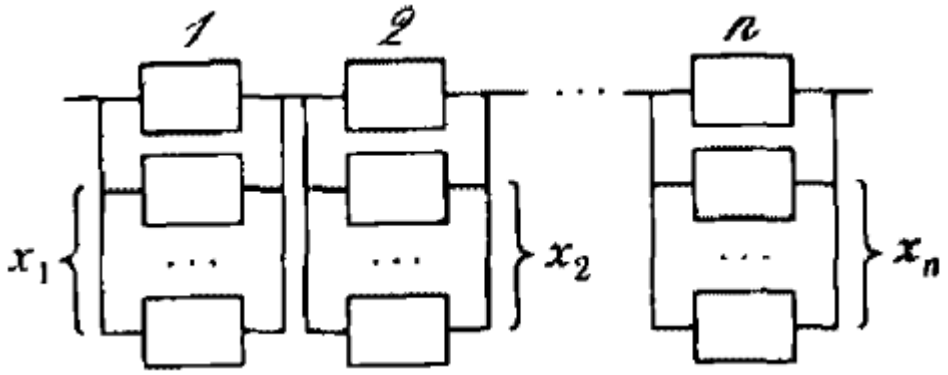
Можливі дві основні постановки задачі оптимального резервування.

1. Шляхом резервування досягти максимально можливого значення обраного показника надійності системи при заданих обмеженнях на загальні витрати, пов’язані з введенням резервних елементів.
2. Шляхом резервування досягнути необхідного значення показника надійності системи при мінімально можливих витратах на резервні елементи.

РОЗДІЛ І

1.1 Математичне формулювання задачі оптимального резервування при одному обмеженні

Нехай існує система, яка складається з n незалежних елементів. Надійність кожного і-го елемента може бути підвищена за рахунок використання резервних елементів і-го типу (мал. 1). Позначимо повне число резервних елементів в і-й резервній групі через x_i.



мал. 1

Роглянемо показники надійності типу часу безвідмовної роботи або коефіцієнту готовності, які можливо уявити у вигляді добутку відповідних вірогідностей для окремих підсистем. Позначимо цей загальний показник надійності через $R_i(x_i)$, де x_i — загальна кількість елементів і-ї резервної групи. Функція $R_i(x_i)$залежить від того, що являє собою підсистема, яка ступінь завантаженості резерву і від інших факторів.

Нехай відома також функція вартості цих елементів $C_i(x_i)$. Зазвичай вважають, що ця функція в усіх прикладних задачах лінійно залежить від кількості елементів:

$$C_i(x_i)=\sum_{i=1}^{n} c_i x_i,$$

c_i— вартість одного елементу і-го типу.

Іноді в якості обмежувального фактора розглядається не вартість, а вага або об'єм резервних елементів, особливо якщо проектувальник має справу з такими об'єктами, як космічні кораблі або підводні човни. Можливий розгляд і декількох обмежувальних факторів разом.

Задача (1) може бути записана у вигляді:

$$\max \left\{ \prod_{i=1}^{n} R_i(x_i) \middle| \sum_{i=1}^{n} C_i(x_i) \leq C_{\text{доп}} \right\}, \; 1 \leq i \leq n,$$

де $C_{\text{доп}}$ — є допустиме обмеження на сумарну вартість резервних елементів.

Задача (2) — зворотна — формується у вигляді:

$$\min \left\{ \sum_{i=1}^{n} C_i(x_i) \middle| \prod_{i=1}^{n} R_i(x_i) \geq R_{\text{треб}} \right\}, \; 1 \leq i \leq n,$$

де $R_{\text{треб}}$ — необхідне значення показника надійності системи.

1.2 Математичне формулювання задачі оптимального резервування при декількох обмеженнях

Зазвичай на практиці розглядається задача типу:

$$\max \left\{ \prod_{i=1}^{n} R_i(x_i) \middle| \sum_{i=1}^{n} C_i^{(1)}(x_i) C_{\text{доп}}^{(1)} \leq, \ldots, \sum_{i=1}^{n} C_i^{(m)}(x_i) \leq C_{\text{доп}}^{(m)} \right\}, \; 1 \leq i \leq n,$$

де верхній індекс показує тип (номер) обмежувального фактору. Як вже згадувалось, такими обмежувальними факторами можуть бути вартість, вага, габарити і тд.

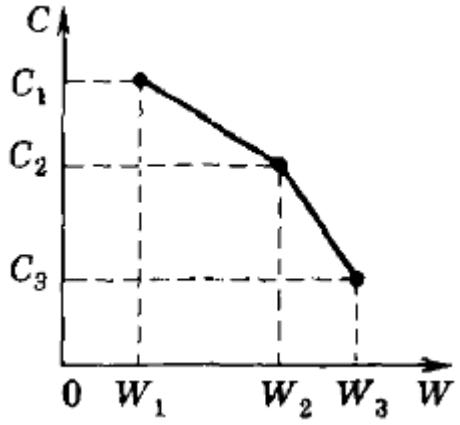
Обернена задача при декількох обмеженнях не мають настільки просте формулювання, як у випадку єдиного обмеження. Зазвичай

розглядається рішення, при якому досягається необхідне значення показника надійності системи, а інші фактори знаходяться в “прийнятній зоні”. Наприклад, якщо обмежувальними факторами є вартість і вага резервних елементів, а самі ці два фактори суперечливі, тобто зменшення ваги призводить до збільшення вартості, то одне й те ж саме значення показника надійності системи може бути досягнуто лише при дуже великій сумарній вазі резерву або при дуже великій його вартості. Наприклад, можна взяти більше важких, але дешевих елементів, програвши у вазі системи. Фінальний вибір залежить від характеру системи і її призначення. Так, для наземної системи може бути більш критичною вартість, а для підводного човна — сумарний об’єм або вага резервних елементів.

В таких випадках виникає необхідність пошуку компромісного рішення, тобто задача зводиться до багатокритерійної оптимізації: знаходження множини Парето. Формально цю задачу можна записати у вигляді:

$$\text{MIN}\left\{\sum_{i=1}^{n} C_i^{(1)}(x_i),\ldots,\sum_{i=1}^{n} C_i^{(m)}(x_i)\,\middle|\,\prod_{i=1}^{n} R_i(x_i)\geq R_{\text{треб}}\right\},\ 1\leq i\leq n.$$

Будемо позначати через MIN прописними буквами оптимізацію за Парето. Необхідний рівень показника надійності може бути досягнутий декількома шляхами. Наприклад, декілька різних векторів $X^1=(x_1^{(1)},.,x_n^{(1)})$, $X^2=(x_1^{(2)},.,x_n^{(2)}),\ldots,X^K=(x_1^{(K)},.,x_n^{(K)})$ забезпечують виконання умови $\prod_{i=1}^{n} R_i(x_i)\geq R_{\text{треб}}$, даючи при цьому різноманітні набори $\sum_{i=1}^{n} C_i^{(1)}(x_i),\ldots,\sum_{i=1}^{n} C_i^{(m)}(x_i)$ значень обмежуючих факторів. Всі ці



рішення і утворюють множину Парето. В двовимірному випадку, наприклад, для вартості C і ваги W множина Парето утворює деяку оболочку — мал. 2. мал. 2

Кожне з рішень характеризується своїм унікальним набором значень C і W. Фактично на цьому математична частина закінчується. Наступає момент, коли вступає в силу приймаюча рішення персона.

1.3 Метод множників Лагранжа

Перші спроби вирішення задачі оптимального резервування опирались на метод невизначених множників Лагранжа. В загальному, використання цього методу є некоректним, оскільки, по-перше, всі змінні в задачі оптимального резервування приймають дискретні значення, а, по-друге, обмеження має вид нерівностей, а не суворих рівностей. Проте, цей метод важливий для ряда випадків, оскільки дозволяє швидко знаходити оціночні значення.

Продемонструємо цей метод на прикладі високонадійної послідовної системи, підвищення надійності елементів якої можливо з використанням гарячого резервування (мал. 1). Цільову функцію R(X) для такої системи можна записати приближено так:

$$R(X) = \prod_{i=1}^{n} R_i(x_i) \approx 1 - \sum_{i=1}^{n} Q_i(x_i),$$

де $X=(x_1,...,x_n)$ — набір резервних елементів всіх типів, $R_i(x_i)$ — показник надійності і-ї резервної групи, а $Q_i(x_i) = 1 - R_i(x_i)$. Показники надійності $R_i(x_i)$ може в даному випадку слугувати вірогідність безперебійної роботи або коефіцієнт готовності, тобто показник, який виражається у вигляді добутку відповідних показників для послідовної системи.

Замінимо задачу максимізації функції R(X) задачею мінімізації функції Q(X) = 1 - R(X). У випадку гарячого резерву з x_i резервними елементами $Q_i(x_i) = q_i^{x_i+1}$.

Поставимо обернену задачу оптимального резервування: знайти вектор складу резервних елементів X = (x_1,...,x_n) доставляючий

$$\min \left\{ \sum_{i=1}^{n} c_i x_i \Big| \sum_{i=1}^{n} q^{x_1+1} = Q^0 \right\},$$

де Q^0 є необхідне значення.

Для вирішення задачі методом невизначених множників Лагранжа необхідно записати функцію Лагранжа $\underline{\lambda}(X)= \sum_{i=1}^{n} c_i x_i + \underline{\lambda} \sum_{i=1}^{n} q^{x_1+1}$,

де $\underline{\lambda}$ — невизначений множник Лагранжа, а потім записати систему (n+1)-их рівнянь:

$$\frac{d}{dx_i}\left(\sum_{i=1}^{n} c_i x_i + \underline{\lambda} \sum_{i=1}^{n} q^{x_1+1}\right) = 0, \text{ i=1,2,...,n} \quad (1)$$

$$\sum_{i=1}^{n} q^{x_1+1} = Q^0. \quad (2)$$

Рівняння вище є рівнянням зв’язку.

В результаті диференціювання отримуємо для перших n значень

$c_i + \underline{\lambda}(\ln q_i) q_i^{x_i+1} = 0$, звідки слідує

$$q_i^{x_i+1} = - \frac{c_i}{\underline{\lambda}_i \ln q_i}. \quad (3)$$

Позначивши $\alpha_i = -c_i/\ln q_i$, після підстановки (3) в рівняння зв’язку (2) знаходимо:

$$\underline{\lambda} = -\frac{1}{Q^0}\sum_{i=1}^{n}\alpha_i. \quad (4)$$

Рішення отримується шляхом підстановки (4) в (3):

$$q_i^{x_i+1} = Q^0 \frac{\alpha_i}{\sum_{i=1}^{n}\alpha_i},$$

звідки фінально отримуємо:

$$x_i = \frac{1}{lnq_i}\ln(Q^0 \frac{\alpha_i}{\sum_{i=1}^{n}\alpha_i}).$$

Зазначимо, що в інших припущеннях про характер резерва отримати будь-які формули в замкнутому вигляді не вдасться.

Метод множників Лагранжа був розглянутий нами, оскільки він допомагає зрозуміти важливий принцип на якому базується метод найшвидшого спуску, широко використовуваний в інженерній практиці.

Можна пояснити суть методу невизначених множників, розглянувши процес численного знаходження вирішення задачі оптимального резервування. З (1) слідує, що $\frac{d}{dx_i}\sum_{i=1}^{n} q^{x_i+1} / \frac{d}{dx_i}\sum_{i=1}^{n} c_i x_i = \underline{\lambda} =$ const (5) для всіх i=1,2,...,n. Рішення полягає в одночасному збільшенні всіх x_i таким чином, щоб при цьому виконувалась умова (5). Цей процес продовжується до тих пір, поки не буде виконана умова (2).

РОЗДІЛ ІІ

2.1 Домінування

Розглянемо систему, яка складається з k послідовних з'єднаних підсистем. Система вважається працюючою тоді і тільки тоді, коли працює кожна з її підсистем. Передбачається, що кожен елемент і-го типу характеризується j типами різних затрат, тобто величина c_{ij} — витрати j-го типу на і-й елемент. Наприклад, першим типом витрат може бути вага, другим — об'єм, третім — вартість. Для кожного типу витрат визначені лінійні обмеження наступного вигляду:

$$\sum_{i=1}^{k} c_{ij} n_i \le c_j, \text{ j=1,2,...,r} \quad (6)$$

Так, наприклад, може вимагатися, щоб повна вага системи не перевищувала деяку задану величину C_1, повний об'єм — величину C_2, а повна вартість в доларах — величинаC_3.

Кожен елемент і-го типу характеризується вірогідністю безвідмовної роботи p_i незалежно від того, працюють чи не працюють інші елементи системи. Таким чином, надійність системи P(n), де n = ($n_{1,}$,..., $n_{k,}$) визначається як P(n) = $\prod_{i=1}^{k}(1-q_i^{n_i})$ (7), де q_i= 1 - p_i.

Наша задача полягає в знаходженні вектора n, компонентами якого є позитивні цілі числа, щоб максимізувати функцію P(n), при виконанні умов (6).

Нехай c_j(n) = $\sum_{i=1}^{k} c_{ij} n_i$ — сумарні витрати j-го типу на систему в цілому, якщо резервована система характеризується вектором n. Далі

будемо говорити, що n^1 домінує над n^2, якщо $c_j(n^1) \leq c_j(n^2)$, j = 1,...,r, в той час як $P(n^1) \geq P(n^2)$. Якщо при цьому принаймні одна з нерівностей є суворою, то будемо казати, що n^1 суворо домінує над n^2. Послідовність S, яка складається з векторів n^h, h = 1,2,..., які задовольняють умови (6), буде називатися домінуючою послідовністю, якщо ні один з векторів n^h не домінується суворо ніяким іншим вектором.

Для вирішення задачі нам необхідно розглянути лише члени домінуючої послідовності S.

Щоб побудувати домінуючу послідовність системи, яка складається лише з двох підсистем 1 і 2, складемо таблицю з двома входженнями: в клітці таблиці, стоячій на перетині рядка n_1 і стовпчика n_2, міститься вектор $\left[c_1(n_1, n_2),\ c_2(n_1, n_2),\ldots,\ c_r(n_1, n_2)Q(n_1, n_2)\right]$, де $c_j(n_1, n_2) = c_{1j}n_1 + c_{2j}n_2$, j=1,...,r та $Q(n_1, n_2) = 1-(1-q_1^{n_1})(1-q_2^{n_2})$

Цей вектор містить інформацію про ненадійність і про витрати на системи, маючих місце у випадку, якщо в системі використано n_1 елементів типу 1 і n_2 елементів типу 2. В таблицю включені лише такі вектори, які задовольняють умови (6). Після цього виключаємо з таблиці всі домінуючі вектори, тобто такі вектори, для яких в таблиці існує принаймні один домінуючий вектор. Ті вектори, які залишилися після вказаної операції виключення утворюють домінуючу послідовність.

Далі покажемо, що домінуюча послідовність для системи, яка складається з s підсистем, може бути побудована на основі домінуючої послідовності для частини тієї ж самої системи, яка складається з s-1 пыдсистем. Тим самим по індукції доводиться існування домінуючої послідовності для системи, яка складається з випадкової кількості

підсистем. Процес полягає у наступному: спочатку будується домінуюча послідовність для підсистем 1 і 2, потім, оперуючи результивною домінуючою послідовністю для цих підсистем і характеристиками підсистеми 3, будується домінуюча послідовність для частини системи, яка складається з підсистем 1, 2 і 3 і так далі до тих пір, поки не буде побудована домінуюча послідовність для всієї системи в цілому.

Побудуємо таблицю, в які рядок n_s відповідає n_s елементам типу s, а h-й стовпчик відповідає векторові n^h, який є h-м членом домінуючої послідовності для перших s-1 підсистем. На перетині стовпчика h і рядка n_s стоїть вектор $\left[c_1(n^h, n_s),\ c_2(n^h, n_s),\ldots,\ c_r(n^h, n_s)Q(n^h, n_s)\right]$, тобто вектор ненадійності і витрат на систему, якщо остання характеризується вектором (n^h, n_s). Зазначимо, що і в загальному випадку $c_j(n^h, n_s)=c_j(n^h)+c_{sj}n_s$, j=1,...,r і

$$Q(n^h, n_s)=1-\left[1 - Q(n^h)\right](1-q_s^{n_s}).$$

У таблицю включаються лише вектори, які задовольняють обмежуючі умови, при чому виключаються всі суворо домінуючі вектори. Решта векторів в таблиці утворюють, як ми доведемо в Теоремі 1, домінуючу послідовність для підсистем 1,2,...,s.

Теорема 1. Вектори, які залишаються суворо недомінуючими в описаній вище таблиці, утворюють домінуючу послідовність для системи із s підсистем.

Доведення. Нам необхідно довести, що два твердження: 1) вектори, які отримані при допомозі вказаного процесу, включають у себе всі суворо недомінуючі вектори і 2) кожен з векторів, отриманих з використанням цього процесу, є суворо недомінуючими.

Перше твердження доведемо по індукції. Спочатку помітимо, що для системи, яка складається з єдиної підсистеми, всі вектори є суворо недомінуючими. Припустимо тепер, що вектори, отримані за допомогою нашого процесу для системи з j підсистем j=1,2,...,s-1, включають всі суворо недомінуючі вектори, які задовольняють умову (6). Тоді за індукцією вектор $(n_1,...,n_{s-1})$ домінується деякими недомінуючими векторами $(n_1^*,...,n_{s-1}^*)$, отриманими в результаті того ж процесу. Таким чином, за означенням:

$$Q(n_1,...,n_{s-1}) \geq Q(n_1^*,...,n_{s-1}^*),$$

$$C_j(n_1,...,n_{s-1}) \geq C_j(n_1^*,...,n_{s-1}^*),$$

$$j=1,...,r.$$

Звідси випливає, що

$$Q_n = 1\text{-}P(n_1,...,n_{s-1})P(n_s) \geq 1\text{-}P(n_1^*,...,n_{s-1}^*)P(n_s^*)=Q(n^*),$$

де $n_s^*=n_s$ та

$$C_j(n\ )=C_j(n_1,...,n_{s-1})+C_j(n_s) \geq C_j(n_1^*,...,n_{s-1}^*)+C_j(n_s^*)=C_j(n^*),$$

$$j=1,...,r,$$

тобто, що вектор n домінується вектором n^*. З іншого боку, вектор n^*, який належить таблиці, сам домінується вектором, отриманим за допомогою нашого процесу. Тому, доведено, що будь-який вектор, який задовольняє умову (6), домінується деяким вектором, отриманим на основі описаного вище процесу. Отже, доведення першого твердження завершено.

Для доведення другого твердження припустимо, що n^0 є деякий вектор, отриманий за допомогою нашого процесу. Якщо n^0 суворо

домінується будь-яким вектором, який задовольняє умову (6), він повинен в той самий час суворо домінуватися деякими недомінуючими векторами, які також задовольняють умову (6). Але щойно було доведено, що всі недомінуючі вектори, які задовольняють умову (6), отримані в процесі застосування нашого процесу. Таким чином, вектор n^0 суворо домінується, наприклад, вектором n^1, також отриманим нашим процесом. У результаті отримано протиріччя, оскільки ніякий вектор, отриманий за допомогою описаного раніше процесу, не може домінувати будь-який інший вектор, отриманий цим самим процесом. Цим самим доведено друге твердження.

2.2 Наближення

На практиці, при використанні описаного процесу домінуючої послідовності, можна зробити наступне припущення. Замість використання виразу $Q(n_1,n_2)=1-(1-q_1^{n_1})(1-q_2^{n_2})=q_1^{n_1}+q_2^{n_2}-q_1^{n_1}q_2^{n_2}$, можна, нехтуючи добутком в останній рівності, використовувати вираз

$$Q(n_1,n_2)\approx q_1^{n_1}+q_2^{n_2}. \quad (7)$$

Аналогічним чином для системи, яка складається з s підсистем, можна наближено записати:

$$Q(n\ ,n_s)\approx Q(n\ )+q_s, \text{ де } n = (n_1,...,n_{s-1})$$

В [3] показано, що використання даного наближення для випадку r=1 призводить до помилки в досяжній надійності системи P, не перевищуючи величину Q^2 (тут Q = 1-P). Для випадку r>1 доведення аналогічне тому, яке приведено в [3].

В усіх застосуваннях описаної процедури оптимального розподілення резервних елементів будемо в подальшому використовувати наближений вираз (7).

Ще одне наближення дозволяє зменшити довжину домінуючої послідовності. При порівнянні пари векторів в таблиці можна внести до розгляду допустиму погрішність ε_j за вартістю j-го типу, а також допустиму погрішність ε_q за надійністю. Тепер, якщо будь-які два вектори в таблиці відрізняються один від одного по витратам j-го типу на величину ε_j чи менше, то по цьому типу витрат вони вважаються ідентичними. (Це ж саме відноситься і до векторів, які відрізняються один від одного за ненадійністю величиною ε_q чи менше). У результаті довжина кожної домінуючої послідовності зменшується. Деякі задачі, які практично не можуть бути вирішені через величезних за своєю довжиною домінуючих послідовностей, іноді вдається наближено вирішити, ввівши допустимі погрішності за одним чи більше факторами. Спочатку варто спробувати вирішити необхідну задачу точними методами. Потім, якщо домінуючі послідовності виявляються занадто довгими для того, щоб отримати рішення без відповідних труднощів обчислювального характеру, вводиться незначна допустима погрішність ненадійності. Якщо і після цього домінуюча послідовність залишається занадто довга, то можна або збільшити допустиму погрішність ε_q, або ввести додаткові допустимі погрішності ε_j за деякими типами витрат. Подібне збільшення допустимих погрішностей або збільшення їх кількості продовжується до тих пір, доки не буде досягнуте шукане рішення.

2.3 Початкове значення n_i

Як буде показано нижче, розміри домінуючих послідовностей визначають масштаби задачі, яка може бути вирішена на обчислювальній машині, а також час, який необхідний для отримання рішення. Тому вкрай важливо намагатися зробити домінуючі послідовності якомога коротшими.

Одним із способів зменшення довжини домінуючих послідовностей є використання найбільш початкових значень n_i, які тільки можливо знайти.

Метод знаходження таких найбільших початкових значень заключається в наступному:

1. Будемо додавати по одному елементу кожного типу до тих пір, доки нарешті, при додаванні чергового елементу не відбудеться порушення хоча б одного з обмежень.
2. Вирахуємо значення надійності P для побудови таким чином системи.
3. З виразу P $\leq$ 1 - $q_i^{n_i^0}$, визначимоn_i^0 — мінімальну кількість елементів і-го типу, необхідних для досягнення надійності, рівній P чи більше. Зрозуміло, що шукане вирішення задачі оптимального резервування буде досягатися для величин n_i, які принаймні не менше отриманих величин n_i^0.
4. Таким чином, в якості початкових значень n_i можуть бути взяті величини n_i^0.

Вигідність використання початкових значень може бути видна з наступних прикладів, для яких були проведені чисельні розрахунки. Так, для системи, яка складається з 10 підсистем, при трьох обмеженнях використання описаного способу призвело до зменшення довжини домінуючої послідовності від початку обчислення до моменту порушення одного з обмежень з 364 до 62 членів. Для системи з 20 підсистем при трьох обмеженнях довжина домінуючої послідовності для етапу вирішення, охоплюючого 10 підсистем, виявивилась рівною 559 членам в той час, як використання початкових значень дозволило прийти до рішення

при результативній довжині домінуючої послідовності, яка рівна всього 69 членам.

Іншим методом для знаходження початкових величин n_i є використання допустимих погрішностей для знаходження наближеного рішення. Після отримання наближеного рішення, варто використовувати наведені вище п. 2, 3 і 4.

РОЗДІЛ ІІІ

3.1. Опис програми

Метод динамічного програмування було реалізовано мовою програмування Python 3 з використанням бібліотека NumPy.

Програма складається з 4-х основних функцій та тіла програми.

Функція обчислення витрат:

```
def expences(c_ij, n_i): #функція, що обчислює витрати при задних c_ij, n_i
  exp = np.zeros(r)
  for i in range(0,r):
    exp[i] = np.sum(c_ij[i]*n_i)
  return exp
```

Функція обчислення надійності:

```
def reliability(q_i, n_i): #функція, що обчислює надійність при задних q_i, n_i
  P_0 = 1
  for i in range(0,k):
    P_0 = P_0*(1-q_i[i]**n_i[i])
  return P_0
```

Функція домінуючих послідовностей:

```
def get_dominated_sequence(CCH, QCH, L):

  NC_L = np.arange(s_i[L], H_i[L]+1)
  QRH = q_i[L]**NC_L

  CRH = NC_L*c_ij[0][L]
  CRH = np.concatenate(([CRH], [NC_L*c_ij[1][L]]))
  for j in range(2, r):
    CRH = np.concatenate((CRH, [NC_L*c_ij[j][L]]))

  M = len(QCH) # кількість стовпців
  N = len(QRH) # кількість рядків

  Table_Values = np.zeros((N,M, len(c_j)+1))
  for i in range(0, N):
    for j in range(0, M):
      for l in range(0, len(c_j)):
        Table_Values[i,j,l] = CCH[l][j] + CRH[l][i]
      Table_Values[i,j,len(c_j)] = QCH[j]+QRH[i]
```

```
    flag = np.ones((N,M)) # масив індикаторів чи належить даний вектор таблиці до домінуючої послідовності
    N_values = np.zeros((N,M, L+1))

    for i in range(0, N):
      for j in range(0, M):
        flag[i][j] = all(Table_Values[i,j,:r] <= c_j)
        N_values[i,j] = np.concatenate((NC_prev[j], NC_L[i]), axis=None)
    N_values = N_values.reshape(N*M, L+1)

    flag = flag.reshape(N*M)
    Values = Table_Values.reshape(N*M, len(c_j)+1)
    # print("Етап ", L-1, "")
    # print("Таблиця значень:", Table_Values)

    i = 0
    for i in range(0, N*M):
      if flag[i]:
        for j in range(i+1, N*M):
          if flag[j]:
            # print(Values[i], Values[i]<=Values[j])
            if (sum(Values[i]<=Values[j]) == r+1): flag[j] = False
            if (sum(Values[i]<=Values[j]) == 0):
              flag[i] = False
              break
    Values_1 = Values[flag == 1]
    CCH_new = np.transpose(Values_1[:, :-1])
    QCH_new = np.transpose(Values_1[:, -1])

    # print("Домінуючі послідовності:", Values_1)
    print("Етап ", L-1, " ok")
    return(CCH_new, QCH_new, N_values[flag == 1])
```

Вхідні дані:

```
[ ] k = 10
    r = 4
    c_j = np.random.randint(low = k*5, high = k*10, size=r)
    c_ij = np.random.rand(r,k)*5
    p_i =  np.random.uniform(low=0.6, high=np.nextafter(1,2), size=k)

    print(c_j)
    print(c_ij)
    print(p_i)
```

```
[86 97 60 67]
[[3.79300363e+00 4.02167216e+00 3.53739893e+00 3.40221046e-01
  2.80415594e+00 4.09927737e+00 1.03554906e+00 3.54314898e+00
  2.28399194e-03 1.28864462e+00]
 [3.46220340e+00 4.40956007e+00 2.57712987e+00 8.49488608e-01
  2.96611738e+00 4.53423637e+00 2.92252392e+00 3.51346331e+00
  6.28553773e-01 4.02619272e-01]
 [3.07330707e+00 3.15778435e+00 1.59480750e+00 4.47614195e+00
  1.69811084e+00 2.29612272e+00 1.23448136e+00 1.77310724e+00
  4.26398409e+00 2.21729439e+00]
 [4.24752787e+00 3.68885244e+00 3.63896289e+00 3.44617890e+00
  7.55666869e-02 2.04025884e-01 4.14012011e+00 2.47691722e+00
  2.64912475e+00 4.06726263e+00]]
[0.64183819 0.87434199 0.97663551 0.80189918 0.65667135 0.94880715
 0.75662471 0.86627167 0.62829438 0.77622346]
```

Функція обрахунку мінімальних початкових значень:

```
import time
start_time = time.time()
q_i = (1 - p_i) # надійність для одиночного елемента підсистеми i
s_i = np.ones(k) # початкове значення для кількості елементів i-го типу n_i

i = 0
while i < k:
  if all(expences(c_ij, s_i) < c_j):
    print("Значення s_i", s_i, "Витрати ", (expences(c_ij, s_i)))
    if (i < k-1):
      s_i[i] += 1
      i += 1
    else:
      s_i[i] += 1
      i = 0
    continue
  s_i[i-1] -= 1
  break

P_0 = reliability(q_i, s_i)
print("Обмеження порушено при значенні s_i: ", s_i, "Надійність:", P_0)

s_i = np.zeros(k)


for i in range(0,k):
  while((1 - q_i[i] ** s_i[i] <= P_0)):
    s_i[i] += 1

s_i = s_i.astype(int)
print("Мінімальні початкові значення s_i = ", s_i)
```

Обчислення максимальної допустимої кількості елементів з урахуванням мінімальних початкових значень:

```
# Максимальна кількість елементів для кожної підсистеми за урахуванням мінімальних початкових значень
H_i = np.full(k, min(c_j))
for i in range(0,k):
  for j in range(0, r):
    H_i_current = int((c_j[j] - np.sum(np.delete(s_i, i)*np.delete(c_ij[j], i)))/c_ij[j][i])
    H_i[i] = min(int(H_i[i]), H_i_current)
H_i = H_i.astype(int)
print("Максимальна кількість елементів для кожної підсистеми за урахуванням мінімальних початкових значень: ", H_i)
```

Тіло програми:

```
NC_0 = np.arange(s_i[0], H_i[0]+1)
NC_prev = NC_0.copy()
# print(NC_prev)
QCH = q_i[0]**NC_0

CCH = NC_0*c_ij[0][0]
CCH = np.concatenate(([CCH], [NC_0*c_ij[1][0]]))
for j in range(2, r):
  # print([CCH])
  # print([NC_0*c_ij[j][0]])
  CCH = np.concatenate((CCH, [NC_0*c_ij[j][0]]))
for L in range(1, k):
  CCH, QCH, NC_prev = get_dominated_sequence(CCH, QCH, L)


print( "Оптимальні значенні n_i: ", NC_prev[QCH == min(QCH)])
print("Час роботи ", (time.time() - start_time), "секунд")
```

3.2 Приклад роботи програми

Розглянемо роботу описаної в минулому розділі програми на прикладі задачі з чотирьох підсистем з наступними показниками:

Номер підсистеми	1	2	3	4	Обмеження
c_{i1}	1.5	2.8	3.6	5.0	40.0
c_{i2}	1.0	1.0	1.0	1.0	25.0
Надійність	0.3	0.2	0.15	0.25	

Тобто необхідно знайти значення n1, n2, n3, n4 такі, що надійність системи

$$P(n_1, n_2, n_3, n_4) = \left(1 - 0.3^{n_1}\right)\left(1 - 0.2^{n_2}\right)\left(1 - 0.15^{n_3}\right)\left(1 - 0.25^{n_4}\right)$$

є максимальною за умови обмежень:

$$1.5n_1 + 2.8n_2 + 3.6n_3 + 5.0n_4 \leq 40.0$$
$$1.0n_1 + 1.0n_2 + 1.0n_3 + 1.0n_4 \leq 25.0 .$$

Зпочатку знайдемо мінімальні початкові значення за допомогою обчислення пробного значення надійності до моменту порушення обмежень:

```
Значення s_i [1. 1. 1. 1.] Витрати  [12.9  4. ]
Значення s_i [2. 1. 1. 1.] Витрати  [14.4  5. ]
Значення s_i [2. 2. 1. 1.] Витрати  [17.2  6. ]
Значення s_i [2. 2. 2. 1.] Витрати  [20.8  7. ]
Значення s_i [2. 2. 2. 2.] Витрати  [25.8  8. ]
Значення s_i [3. 2. 2. 2.] Витрати  [27.3  9. ]
Значення s_i [3. 3. 2. 2.] Витрати  [30.1 10. ]
Значення s_i [3. 3. 3. 2.] Витрати  [33.7 11. ]
Значення s_i [3. 3. 3. 3.] Витрати  [38.7 12. ]
Обмеження порушено при значенні s_i:  [3. 3. 3. 3.] Надійність: 0.9469277960625
Мінімальні початкові значення s_i =  [3 2 2 3]
```

Мінімальні початкові значення:

$$n_1 = 3,\ n_2 = 2,\ n_3 = 2,\ n_4 = 3$$

Підставляючи мінімальні початкові значення в обмеження знаходимо максимальну кількість допустимих елементів для кожної підсистеми:

$$H_1 = 8,\ H_2 = 4,\ H_3 = 4,\ H_4 = 4$$

Надалі попарно проводиться обчислення домінуючих послідовностей.

Етап 0 – підсистеми 1 і 2.

```
Етап  0
Таблиця значень: [[[1.010000e+01 5.000000e+00 6.700000e-02]
  [1.160000e+01 6.000000e+00 4.810000e-02]
  [1.310000e+01 7.000000e+00 4.243000e-02]
  [1.460000e+01 8.000000e+00 4.072900e-02]
  [1.610000e+01 9.000000e+00 4.021870e-02]
  [1.760000e+01 1.000000e+01 4.006561e-02]]

 [[1.290000e+01 6.000000e+00 3.500000e-02]
  [1.440000e+01 7.000000e+00 1.610000e-02]
  [1.590000e+01 8.000000e+00 1.043000e-02]
  [1.740000e+01 9.000000e+00 8.729000e-03]
  [1.890000e+01 1.000000e+01 8.218700e-03]
  [2.040000e+01 1.100000e+01 8.065610e-03]]

 [[1.570000e+01 7.000000e+00 2.860000e-02]
  [1.720000e+01 8.000000e+00 9.700000e-03]
  [1.870000e+01 9.000000e+00 4.030000e-03]
  [2.020000e+01 1.000000e+01 2.329000e-03]
  [2.170000e+01 1.100000e+01 1.818700e-03]
  [2.320000e+01 1.200000e+01 1.665610e-03]]]
Домінуючі послідовності: [[1.01000e+01 5.00000e+00 6.70000e-02]
 [1.16000e+01 6.00000e+00 4.81000e-02]
 [1.29000e+01 6.00000e+00 3.50000e-02]
 [1.44000e+01 7.00000e+00 1.61000e-02]
 [1.59000e+01 8.00000e+00 1.04300e-02]
 [1.74000e+01 9.00000e+00 8.72900e-03]
 [1.72000e+01 8.00000e+00 9.70000e-03]
 [1.87000e+01 9.00000e+00 4.03000e-03]
 [2.02000e+01 1.00000e+01 2.32900e-03]
 [2.17000e+01 1.10000e+01 1.81870e-03]
 [2.32000e+01 1.20000e+01 1.66561e-03]]
```

Етап 1 – домінуючі послідовності з Етапу 0 (підсистеми 1,2) та підсистема 3:

```
Етап  1
Таблиця значень: [[[1.730000e+01 7.000000e+00 8.950000e-02]
  [1.880000e+01 8.000000e+00 7.060000e-02]
  [2.010000e+01 8.000000e+00 5.750000e-02]
  [2.160000e+01 9.000000e+00 3.860000e-02]
  [2.310000e+01 1.000000e+01 3.293000e-02]
  [2.460000e+01 1.100000e+01 3.122900e-02]
  [2.440000e+01 1.000000e+01 3.220000e-02]
  [2.590000e+01 1.100000e+01 2.653000e-02]
  [2.740000e+01 1.200000e+01 2.482900e-02]
  [2.890000e+01 1.300000e+01 2.431870e-02]
  [3.040000e+01 1.400000e+01 2.416561e-02]]

 [[2.090000e+01 8.000000e+00 7.037500e-02]
  [2.240000e+01 9.000000e+00 5.147500e-02]
  [2.370000e+01 9.000000e+00 3.837500e-02]
  [2.520000e+01 1.000000e+01 1.947500e-02]
  [2.670000e+01 1.100000e+01 1.380500e-02]
  [2.820000e+01 1.200000e+01 1.210400e-02]
  [2.800000e+01 1.100000e+01 1.307500e-02]
  [2.950000e+01 1.200000e+01 7.405000e-03]
  [3.100000e+01 1.300000e+01 5.704000e-03]
  [3.250000e+01 1.400000e+01 5.193700e-03]
  [3.400000e+01 1.500000e+01 5.040610e-03]]

 [[2.450000e+01 9.000000e+00 6.750625e-02]
  [2.600000e+01 1.000000e+01 4.860625e-02]
  [2.730000e+01 1.000000e+01 3.550625e-02]
  [2.880000e+01 1.100000e+01 1.660625e-02]
  [3.030000e+01 1.200000e+01 1.093625e-02]
  [3.180000e+01 1.300000e+01 9.235250e-03]
  [3.160000e+01 1.200000e+01 1.020625e-02]
  [3.310000e+01 1.300000e+01 4.536250e-03]
  [3.460000e+01 1.400000e+01 2.835250e-03]
  [3.610000e+01 1.500000e+01 2.324950e-03]
  [3.760000e+01 1.600000e+01 2.171860e-03]]]
```

```
Домінуючі послідовності: [[1.73000e+01 7.00000e+00 8.95000e-02]
 [1.88000e+01 8.00000e+00 7.06000e-02]
 [2.01000e+01 8.00000e+00 5.75000e-02]
 [2.16000e+01 9.00000e+00 3.86000e-02]
 [2.31000e+01 1.00000e+01 3.29300e-02]
 [2.46000e+01 1.10000e+01 3.12290e-02]
 [2.44000e+01 1.00000e+01 3.22000e-02]
 [2.37000e+01 9.00000e+00 3.83750e-02]
 [2.52000e+01 1.00000e+01 1.94750e-02]
 [2.67000e+01 1.10000e+01 1.38050e-02]
 [2.82000e+01 1.20000e+01 1.21040e-02]
 [2.80000e+01 1.10000e+01 1.30750e-02]
 [2.95000e+01 1.20000e+01 7.40500e-03]
 [3.10000e+01 1.30000e+01 5.70400e-03]
 [3.25000e+01 1.40000e+01 5.19370e-03]
 [3.31000e+01 1.30000e+01 4.53625e-03]
 [3.46000e+01 1.40000e+01 2.83525e-03]
 [3.61000e+01 1.50000e+01 2.32495e-03]
 [3.76000e+01 1.60000e+01 2.17186e-03]]
```

Етап 2 – домінуючі послідовності з Етапу 1 (підсистеми 1,2, 3) та підсистема 4:

```
Етап  2
Таблиця значень: [[[3.230000e+01 1.000000e+01 1.051250e-01]
  [3.380000e+01 1.100000e+01 8.622500e-02]
  [3.510000e+01 1.100000e+01 7.312500e-02]
  [3.660000e+01 1.200000e+01 5.422500e-02]
  [3.810000e+01 1.300000e+01 4.855500e-02]
  [3.960000e+01 1.400000e+01 4.685400e-02]
  [3.940000e+01 1.300000e+01 4.782500e-02]
  [3.870000e+01 1.200000e+01 5.400000e-02]
  [4.020000e+01 1.300000e+01 3.510000e-02]
  [4.170000e+01 1.400000e+01 2.943000e-02]
  [4.320000e+01 1.500000e+01 2.772900e-02]
  [4.300000e+01 1.400000e+01 2.870000e-02]
  [4.450000e+01 1.500000e+01 2.303000e-02]
  [4.600000e+01 1.600000e+01 2.132900e-02]
  [4.750000e+01 1.700000e+01 2.081870e-02]
  [4.810000e+01 1.600000e+01 2.016125e-02]
  [4.960000e+01 1.700000e+01 1.846025e-02]
  [5.110000e+01 1.800000e+01 1.794995e-02]
  [5.260000e+01 1.900000e+01 1.779686e-02]]
```

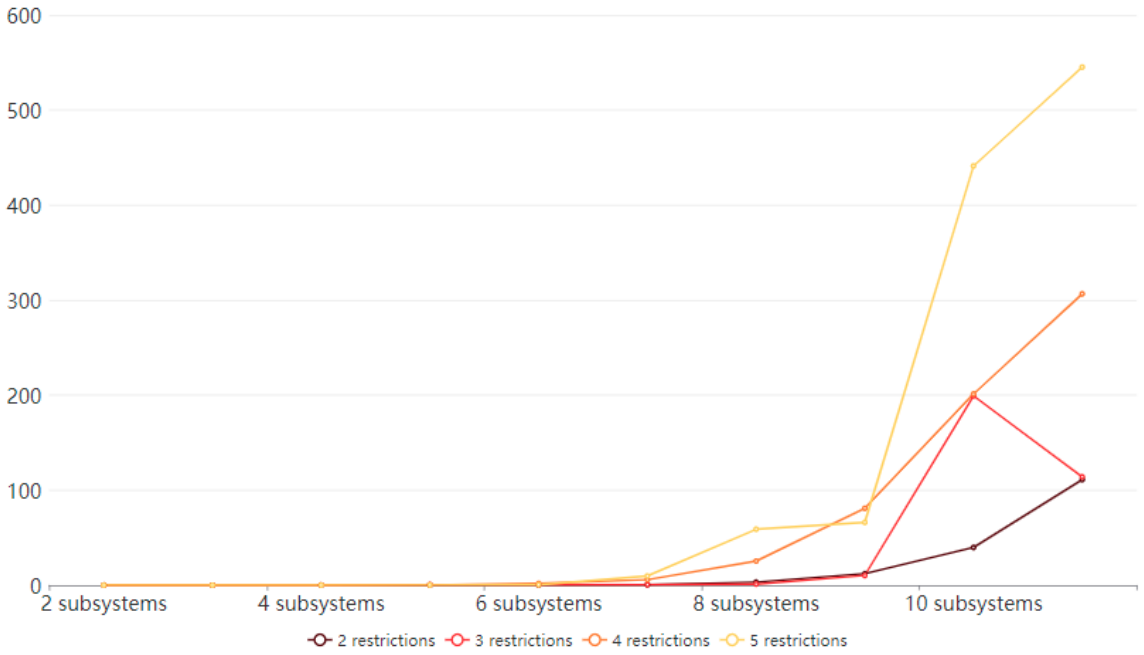
```
[[3.730000e+01 1.100000e+01 9.340625e-02]
 [3.880000e+01 1.200000e+01 7.450625e-02]
 [4.010000e+01 1.200000e+01 6.140625e-02]
 [4.160000e+01 1.300000e+01 4.250625e-02]
 [4.310000e+01 1.400000e+01 3.683625e-02]
 [4.460000e+01 1.500000e+01 3.513525e-02]
 [4.440000e+01 1.400000e+01 3.610625e-02]
 [4.370000e+01 1.300000e+01 4.228125e-02]
 [4.520000e+01 1.400000e+01 2.338125e-02]
 [4.670000e+01 1.500000e+01 1.771125e-02]
 [4.820000e+01 1.600000e+01 1.601025e-02]
 [4.800000e+01 1.500000e+01 1.698125e-02]
 [4.950000e+01 1.600000e+01 1.131125e-02]
 [5.100000e+01 1.700000e+01 9.610250e-03]
 [5.250000e+01 1.800000e+01 9.099950e-03]
 [5.310000e+01 1.700000e+01 8.442500e-03]
 [5.460000e+01 1.800000e+01 6.741500e-03]
 [5.610000e+01 1.900000e+01 6.231200e-03]
 [5.760000e+01 2.000000e+01 6.078110e-03]]]
Домінуючі послідовності: [[32.3      10.        0.105125]
 [33.8      11.        0.086225]
 [35.1      11.        0.073125]
 [36.6      12.        0.054225]
 [38.1      13.        0.048555]
 [39.6      14.        0.046854]
 [39.4      13.        0.047825]
 [38.7      12.        0.054   ]]
```

З домінуючих послідовностей, які є результатом 3 етапу було обрано вектор с найменшою ненадійністю. Значення

$$n_1 = 6,\ n_2 = 3,\ n_3 = 2,\ n_4 = 3,$$

які відповідають отриманому вектору і є оптимальним розв’язком системи.

3.3 Інфографіка



	2 restrictions	3 restrictions	4 restrictions	5 restrictions
2 subsyste	0.0170464515686	0.0189192295074	0.0185480117797	0.013760328292850
3 subsyste	0.0144991874694	0.0255448818206	0.0211343765258	0.014120340347290
4 subsyste	0.0228464603424	0.0643186569213	0.0313155651092	0.035264492034910
5 subsyste	0.0582196712493	0.5100204944610	0.1240880489349	0.153140306472780
6 subsyste	0.1519246101379	0.3453330993652	1.8287565708160	0.405272722244260
7 subsyste	0.5727279186248	0.4710164070129	5.7509396076202	9.694065809249900
8 subsyste	3.2397716045380	1.1937813758850	25.364343643188	58.917860984802000
9 subsyste	12.077820062637	10.236672639847	80.944291353226	66.0799343585970
10 subsyst	39.712050914764	199.26919364929	201.28753256798	441.296555757520
11 subsyst	111.16367459297	114.032809257510	306.70427203178	545.352169752120

*Дані вказані у секундах.

ВИСНОВОК

В даній дипломній роботі було реалізовано метод динамічного програмування для оптимального резервування при кількох обмеженнях і на прикладі була перевірена робота методу. Також, було опрацьовано літературу на тему оптимального резервування. Після тестування роботи методу можна відзначити, що при збільшенні кількості підсистем і обмежень збільшуються і час роботи програми, і вимоги до апаратної частини обладнання.

Варто зазначити, що реалізований метод може бути також використаний і для задач, які не пов’язані з задачами оптимального резервування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. І. О. Ушаков. Курс теории надежности систем — 2008, 8р.
2. Р. Барлоу, Ф. Прошан (переклад І. О. Ушаков). Математическая теория надежности — 1969, 6р.
3. Kettele J. D., Jr., 1962, Least-Cost allocation of reliability investment, Operations Res., v. 10, №2, 249-265 с.