УДК 004.45 519.1 **DOI:10.30837/rt.2024.4.219.02**

Д.Ю. ГОЛУБНИЧИЙ, канд. техн. наук, О.С. ГОЛОВЧЕНКО

ЗАСТОСУВАННЯ АЛГОРИТМІВ РАНГОВОГО ПІДХОДУ ПРИ ПЛАНУВАННІ РОЗПОДІЛУ ЗАДАЧ В БАГАТОПРОЦЕСОРНИХ СИСТЕМАХ

Вступ

Класична комбінаторна задача про ранець є прикладом задачі цілочисельного лінійного програмування (ЦЛП) з булевими змінними (БЗ). Вона належить до широко відомих завдань дискретної оптимізації. Це завдання вперше було сформульовано Д. Данцигом у роботі і з того часу активно досліджується. Популярність класичної комбінаторної задачі про ранець викликана великою кількістю її додатків, оскільки багато які з реальних завдань описуються в рамках даної моделі. Однією із сфер застосування є розподіл завдань у багатопроцесорних системах та інформаційно-комунікаційних мережах.

Вирішення завдання оптимального планування розподілу задач у мережі з використанням моделі ЦЛП з БЗ полягає у знаходженні оптимального, відмовостійкого та ефективного способу розподілу ресурсів мережі. Даний підхід дозволяє враховувати як обмеженість ресурсів, так і необхідність забезпечення резервування для підвищення надійності роботи інформаційно-комунікаційних систем та мереж.

Зовнішня простота математичної моделі є оманливою, оскільки обчислювальну складність задачі про ранець характеризує результат про її NP-труднощі вже в одновимірному однокритеріальному випадку. Незважаючи на це, доведена NP-повнота задачі та її широка застосовуваність відкривають можливості для вивчення різних підходів до її вирішення, включаючи приблизні методи, націлені на пошук рішень, близьких до оптимальних.

Мета роботи – експериментальне порівняння властивостей наближених алгоритмів вирішення задачі цілочисельного лінійного програмування з булевими змінними, заснованих на ранговому підході, з метою їх подальшого застосування при розподілі задач в багатопроцесорних системах.

1. Основна частина

1.1. Формалізація планування розподілу задач в багатопроцесорних системах

Використання математичної моделі ЦЛП з БЗ для рішення задачі оптимального планування розподілу задач у мережі з відмовостійким функціонуванням має такі особливості [1 – 5]:

1. Бінарні змінні: у ЦЛП з БЗ рішення приймається у вигляді "0" або "1", що означає, чи включена певна задача або ресурс в планування. Це підходить для задач, де є вибір між кількома можливими альтернативами, наприклад, вибір маршруту для передачі даних чи визначення, чи використовувати певний вузол як резервний;

2. Моделювання резервування: БЗ дозволяють точно визначити в моделі резервування ресурсів у мережі;

3. Обмеження відмовостійкості: за допомогою використання лінійних обмежень в ЦЛП моделі враховуються вимоги до відмовостійкості, що дозволяє впроваджувати політику надмірності, яка необхідна для забезпечення стійкості до відмов;

4. Оптимізація використання ресурсів: у моделі ЦЛП задаються обмеження на ресурси, такі як пропускна здатність або кількість доступних вузлів. Це допомагає знайти оптимальний розподіл задач, щоб ефективно використовувати доступні ресурси та забезпечити балансування навантаження в мережі;

5. Моделювання розподілених задач: використання моделі ЦЛП з БЗ дозволяє оптимально визначити розподіл завдань по мережі, враховуючи фактори, такі як час виконання, затримка, а також доступність ресурсів. Використання БЗ полегшує опис сценаріїв, коли задача або призначена для виконання в конкретному вузлі, або не виконується;

6. Забезпечення QoS: модель ЦЛП можна розширити для включення вимог до якості обслуговування (QoS), використовуючи додаткові обмеження;

7. Комбінаторна складність: використання БЗ робить модель комбінаторною, що означає, що задача належить до класу NP-складних. Це значить, що розмірність задачі швидко зростає зі збільшенням кількості змінних, тому для великих мереж потрібні спеціалізовані методи рішення, такі як евристики або наближені алгоритми;

8. Відповідність цільовій функції: цільова функція в ЦЛП з БЗ може бути сформульована таким чином, щоб мінімізувати загальні витрати, час виконання або максимізувати мулевих стійкість мережі. Використання булевих змінних дозволяє включати до моделі додаткові критерії, що відображають вимоги щодо надійності та ефективності;

9. Аналіз надмірності**:** за допомогою БЗ можна моделювати надмірність у мережі, щоб визначити, які вузли чи канали можуть бути відключені без втрати працездатності. Це дозволяє знаходити мінімальну кількість необхідних резервних елементів;

10. Гнучкість у визначенні обмежень: БЗ полегшують включення різних додаткових обмежень до моделі, таких як вимоги до розміщення певних завдань в конкретних вузлах, забезпечення безпеки або визначення критичних елементів мережі, що повинні бути постійно активними.

Для оцінки процесу управління розподілом задач у мережі використовуються показники оцінки ефективності [6]:

- коефіцієнт функціональної потужності мережі:

$$E_\nu(\vartheta_\mu^i) = \sum_{i=1}^{n_\nu}\sum_{\mu=1}^{m_i}\beta_{\mu i}\,, \tag{1}$$

де nν – загальна кількість ПМ у мережі в стані S_ν;

m_i – загальна кількість завдань, що здатний вирішувати і-й ПМ у стані S_ν;

$\beta_{\mu i}$ – вага μ-ї задачі і-го ПМ, що характеризує її важливість (пріоритет) для інформаційно-комунікаційних систем;

- сумарний час доступу до даних у мережі, який обумовлений розміщенням сегментів бази даних (БД) мережі на її фізичній структурі:

$$T = \sum_{i=1}^{m}\sum_{j=1}^{n} K_j p_i \tau_{ij} x_{ij}\,, \tag{2}$$

де $\tau_{ij} = \dfrac{V_i^{(b)}}{\lambda_{ij}}$ – середній час вибору 1 кілобайта інформації з і-го фрагмента у вузлі j;

λ_{ij} – коефіцієнт, який враховує швидкість вибору та обробки 1 мегабайта даних при звертанні до і-го фрагмента БД у j-му вузлі;

$V_i^{(b)}$ – ємність зовнішньої пам’яті, яка необхідна для розміщення і-го фрагмента БД;

p_i – характеристика частоти використання і-го фрагмента БД при функціонуванні системи управління базою даних (СУБД);

K_j – коефіцієнт, який показує швидкість доступу до даних на j-му вузлі мережі;

$T_в$ – середній час відновлення мережі $T_в$,

$$x_{ij} = \begin{cases} 1, \text{ якщо фрагмент БД з номером } i \text{ розміщений у } j-\text{му вузлі мережі;} \\ 0, \text{ у протилежному випадку;} \end{cases}; \tag{3}$$

Нехай маємо M_i $i = \overline{(1,n)}$ – управляючих процесорних модулів (ПМ), які служать для управління об’єктами O_ε. Відмови системи зв’язку і відмови ПМ вважаємо незалежними. Під станом s(t) на момент часу t маємо на увазі набір станів усіх модулів у цей момент, тобто $s(t)=\sigma_1,\ldots,\sigma_n$, де $\sigma_i \in \{0,1\}$ та

$$\sigma_i = \begin{cases} 1, \text{якщо} \quad M_i - \text{відмовивший} \quad \text{модуль} \quad (\text{о} - \text{ПМ}); \\ 0, \text{якщо} \quad M_i - \text{працездатний} \quad \text{модуль} \quad (\text{р} - \text{ПМ}). \end{cases} \tag{4}$$

Початковий стан системи s(t = 0) = s0 = 00…0. Нехай Λn – множина усіх станів системи; D = {M1,… Mn – множина усіх модулів системи; Ων = {U1, … ,UL}– множина усіх задач, які вирішуються у мережі в стані Sν; $\Omega_i^{\nu} = \{U_1^i, \ldots, U_\mu^i, \ldots, U_m^i\}$ – підмножина задач, що працездатний модуль Mi здатний вирішувати, коли система знаходиться в стані Sν. Для стану s0 задана множина Ω0 = {Uj}, 0 = {Uj}, і початковий розподіл задач між усіма модулями, тобто підмножини $\Omega_i^{\nu} = \{U_1^i, \ldots, U_\mu^i, \ldots, U_m^i\}$. Нехай Dfν = {Mi}ν, Drν = {Mi}ν – множини відповідно ПМ, які відмовили, і працездатних ПМ, що відповідають стану Sν; Afν – множина усіх власних задач модулів, що відмовили, для стану Sν; Arν = Ω0\Afν – множина усіх власних задач р-Пм для стану Sν. Кожна задача $U_\mu^i \in \Omega_i^{\nu}$ характеризується ступенем важливості, що визначає мету функціонування системи управління і задається ваговим коефіцієнтом {βμi}. Таким чином, результат розподілу задач визначається корисним ефектом E, який оцінюється сумарною ваговою характеристикою множин задач, які вирішуються в мережі у даному стані Sν, який називають функціональною потужністю мережі (Eν).

Математична модель задачі перерозподілу визначається як:

$$E_\nu = \sum_{i\backslash=1}^{n_\nu} \sum_{\mu=1}^{m_i} \beta_{\mu i} X_{\mu i} \to \max, \tag{5}$$

при обмеженнях:

$$\sum_{i\backslash=1}^{n_\nu} \sum_{\mu=1}^{m_i} \Delta T_{\mu i} X_{\mu i} \le \Delta T_\nu^{\text{доп}}; \tag{6}$$

$$\sum_{i\backslash=1}^{n_\nu} \sum_{\mu=1}^{m_i} V_{\mu i} X_{\mu i} \le V_\nu^{\text{доп}}; \tag{7}$$

$$\sum_{i\backslash=1}^{n_\nu} \sum_{\mu=1}^{m_i} t_{\mu i} X_{\mu i} \le T_\nu^{\text{доп}}; \tag{8}$$

$$\sum_{\mu=1}^{m_i} X_{\mu i} \le 1, \tag{9}$$

де

$$X_{\mu i} = \begin{cases} 1, \text{якщо} \quad \mu - \text{задача} \quad \text{вирішується} \quad \text{в} \quad i - \text{му} \quad \text{ПМ}; \\ 0, \quad \text{у} \quad \text{іншому випадку}. \end{cases} \tag{10}$$

$\Delta T_\nu^{\text{доп}}$, $V_\nu^{\text{доп}}$, $T_\nu^{\text{доп}}$ – допустимі граничні значення відповідно до сумарного середнього часу обслуговування задачі в і-му ПМ, ємності пам’яті, яка необхідна для рішення задачі, та часу перерозподілу інформації у всій мережі в цілому.

Обмеження (9) визначає, що задача може бути призначена для рішення тільки на один вузол мережі. Таким чином, задача перерозподілу (5) – (9) зводиться до багаторазового рішення задачі ЦЛП із БЗ в умовах деградації мережі, яку необхідно вирішувати в масштабі реального часу.

1.2. Математична постановка задачі цілочисельного лінійного програмування з булевими змінними

Розглянемо сутність задач ЦЛП з БЗ на прикладі задачі про рюкзак. Загальна постановка цієї задачі формулюється таким чином. Необхідно знайти двійковий вектор $\vec{x}$, при якому досягається максимум функції:

$$f(\vec{x}) = \sum_{j=1}^{n} c_j \times x_j, \tag{11}$$

при виконанні умов:

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i, \tag{12}$$

$$x_j \in \{0,1\}, \qquad i = \overline{(1,m)}; \qquad j = \overline{(1,n)}. \tag{13}$$

Для спрощення викладу математичної моделі розглянемо одномірну задачу, тобто максимізуємо функціонал

$$f(\vec{x}) = c_1 x_1 + c_2 x_2 + ... + c_n x_n, \tag{14}$$

при обмеженнях:

$$\sum_{j=1}^{n} a_{1j} x_j \le b, \tag{15}$$

$$c_1 \ge c_2 \ge ... \ge c_n; \qquad a_{ij} > 0; \qquad c_j > 0 \qquad j = \overline{(1,n)}. \tag{16}$$

Сутність рангового підходу до рішення задач ЦЛП з БЗ описана а [3, 4]. Тут приведемо лише деякі основні положення:

- всі можливі вектори рішень можуть бути розподілені по рангах, де кількість одиниць у векторі відповідає рангу;
- вектори будуються послідовно від найменших рангів до найбільших;
- перед побудовою векторів елементи мають бути відсортовані в порядку зменшення їхньої ваги функціоналу;
- всі вектори можуть бути представлені у вигляді графу, де кожна вершина позначає всі можливі вектори, останнім елементом яких є елемент вершини (рис. 1).

1.3. Узагальнена процедура А0

Основою реалізації рангового підходу є реалізація загальної процедури А0, описаної у [5]. Процедура А0 дозволяє на основі обраного правила відсікання безперспективних шляхів вирішити задачу (11) – (16), дозволяє формувати множини локальних екстремумів і виділяти серед них глобальний, використовуючи конструкцію графа DΔ.

Програмно реалізація процедури А0 зводиться до виконання вкладених циклів, де цикл найвищого рівня виконує перебір рангів r, цикл другого рівня виконує перебір вершин (елементів вектора), чий порядковий номер дорівнює або вищий за поточний ранг. Цикл останнього, третього рівня виконує перебір всіх векторів попереднього рангу, останній елемент яких має порядковий номер, нижчий за порядковий номер елементу, визначений другим циклом, і викликає функцію побудови нового вектора. Новий вектор формується додаванням нового елементу до вектора попереднього рангу.

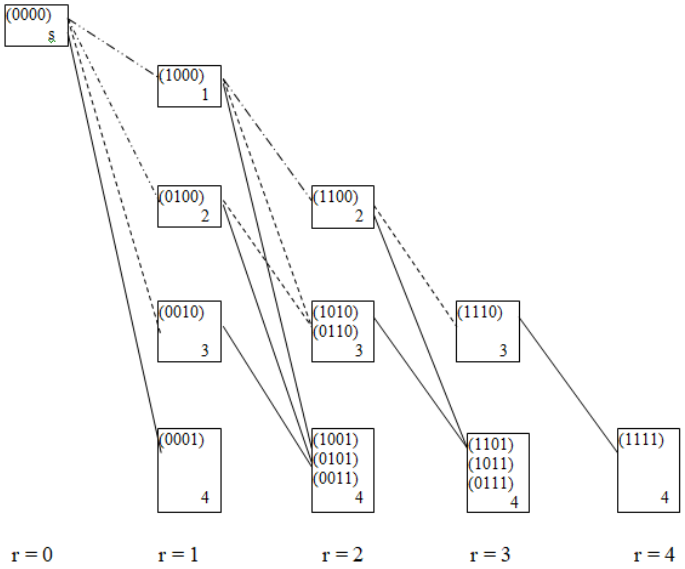

Рис. 1. Геометрична інтерпретація графа DΔ

Для реалізації структури, в якій зберігаються зазначені вектори, було створено динамічний масив посилань на списки, розмірність якого визначається при визначенні розмірності задачі, і дорівнює їй. В списках зберігаються посилання на об'єкти, які являють собою вектори та їхні вагові параметри. Структура цих об'єктів приведена у [7]. Таким чином, всі вектори одного рангу містяться в одному спільному списку. Такий розподіл дещо змінює логіку перебору, оскільки комірки (тут і надалі коміркою позначатимемо всі вектори однієї вершини) не мають фізичних кордонів, а тільки логічні.

1.4. Стратегії відсікання безперспективних шляхів в графі DΔ

В роботі [5] приведено декілька стратегій відсікання безперспективних шляхів, тобто стратегій, що дозволяють зменшити кількість векторів, які необхідно проаналізувати, відкидаючи ті, які не можуть привести до найкращого рішення.

Далі будуть розглянуті три стратегії відсікання комбінації, по дві з яких формують два наближені алгоритми пошуку найкращих векторів.

1.4.1. Стратегія відсікання L1

Наближена стратегія відсікання безперспективних шляхів L1 базується на виборі з кожної комірки тільки одного вектора з найбільшою вагою за функціоналом для побудови нового вектора. При цьому вага обмеження враховується тільки в розумінні, що:

- забороняється обирати вектор, вага обмеження якого при додаванні нового елементу перевищить максимально допустиму вагу обмеження;
- якщо існують два чи більше векторів з однаковою вагою функціоналу, то слід розглядати той, вага обмеження якого є меншою.

При цьому слід зазначити, що довгий час вважалося правилом, що оскільки елементи векторів перед початком роботи відсортовуються в порядку зменшення ваги функціоналу, а при побудові векторів нові елементи вибираються послідовно, то перший вектор у комірці завжди матиме найбільшу вагу функціоналу, а решта векторів в комірці будуть відсортовані в порядку зменшення ваги функціоналу. Насправді це правило відповідає дійсності тільки за умови розгляду всіх можливих векторів рішення аналогічно до повного перебору. Але навіть у цьому випадку можна спостерігати стрибок значення ваги функціоналу ближче до середини комірки.

Розуміння того, що згадане правило є недійсним, підводить до питання, як визначати вектор з найбільшою вагою функціоналу. Наразі розглянуто дві основні можливості визначення вектора з найбільшою вагою функціонала:

- за допомогою сортування;
- за допомогою простого порівняння.

Кожний із згаданих підходів має свої переваги та недоліки. Так, для сортування було обрано стандартну функцію сортування мови C++ stable_sort(), в основі якої лежить алгоритм сортування злиттям (merge sort). Значною перевагою використання сортування є те, що сортування необхідно провести тільки один раз, після чого, якщо найважчий за функціоналом вектор не зможе бути обраний, наступний буде обрано без зайвих дій. Недоліком такого підходу є додаткові витрати пам'яті, а також те, що якщо перший елемент задовольняє умовам обмеження (12) – (16), то сортування виконує велику кількість зайвої роботи.

Програмно даний підхід реалізовано таким чином:

- вектори однієї комірки переписуються до одномірного вектора для того, щоб не змінювати вихідну структуру даних і не впливати таким чином на подальші обчислення;
- виконуються два сортування, перше сортує вектори за вагою обмеження у порядку зростання, друге – за вагою функціоналу у порядку спадання. Важливо, що друге сортування залишає вектори у тій же послідовності, в якій вони надійшли, за умови, що їхні функціонали однакові.

Другий підхід засновано на ідеї, що в більшості випадків необхідний вектор буде знайдений за кілька перших спроб. Програмно він реалізований так, що:

- при першому проході відбувається перебір всіх векторів комірки і порівнюється за вагою функціоналу. Як тільки перший вектор обрано, для всіх наступних векторів, що мають таку ж вагу функціоналу, відбувається додаткова перевірка ваги обмеження. Тобто обирається тільки такий вектор, вага функціоналу якого або більша за вагу поточного обраного, або така ж, але вага обмеження менша;
- якщо обраний вектор не може бути використаний для побудови вектора наступного рангу через отримувану вагу обмеження, то відбувається ще один цикл перебору, причому цього разу додається додаткове обмеження, яке не дозволяє обирати вектори тієї ж або більшої (для третього проходу і далі) ваги функціоналу ніж у попереднього обраного вектора.

В цілому метод з сортуванням вигідно використовувати за умови, що перші вектори з вищою вагою функціоналу будуть відкинуті і буде необхідно перебирати решту векторів комірки. Метод лінійного перебору є економнішим за сортування з точки зору використання додаткової пам'яті і кількості операцій, але підходить здебільшого у тих випадках, коли відповідний вектор знаходиться у першій трійці за вагою функціоналу.

1.4.2. Стратегія відсікання L2

Опис стратегії відсікання L2 у формалізованому вигляді описано в [5]. У спрощеному вигляді дану стратегію можна описати так, що для побудови нового вектора обирається тільки один вектор з кожної комірки попереднього рангу з меншим порядковим номером останнього елементу, ніж у додаваного елементу, який має найменшу у комірці вагу обмеження. Логічно, що якщо цей вектор не може бути обраний для побудови нового вектора, то інші вектори даної комірки не розглядаються.

Оскільки за вагою обмеження вектори всередині комірки ніяк не структуровані, то для вибору відповідного вектора є необхідність у додаткових механізмах відбору. Такими механізмами можуть стати вже використовувані у стратегії L1 сортування та лінійного пошуку.

Програмно підхід заснований на сортуванні схожим чином, до того як його реалізовано для L1:

- вектори однієї комірки переписуються до одномірного вектора для того, щоб не змінювати вихідну структуру даних і не впливати таким чином на подальші обчислення;
- виконуються два сортування, перше сортує вектори за вагою функціоналу у порядку спадання, друге – за вагою обмеження у порядку зростання. Важливо, що друге сортування залишає вектори у тій же послідовності, в якій вони надійшли за умови, що їхні обмеження однакові.

Таким чином, найпершим же вектором у відсортованій послідовності буде найлегший за вагою обмеження і одночасно найважчий за функціоналом серед тих векторів, які мають однакову з ним вагу обмеження. Недоліком такого підходу є те, що решта векторів у відсортованій послідовності не потрібні, але операції сортування для них виконані.

Другий підхід, тобто лінійний пошук програмно реалізований так, що фактично відбувається тільки один прохід по комірці, при якому відбувається перебір всіх векторів комірки і порівнюється за вагою обмеження. Як тільки перший вектор обрано, для всіх наступних векторів, що мають таку ж вагу обмеження, відбувається додаткова перевірка ваги функціоналу. Тобто обирається тільки такий вектор, вага обмеження якого або менша за вагу поточного обраного, або така ж, але вага функціоналу більша.

В цілому метод з сортуванням не є вигідним для використання даною стратегією, оскільки виконує зайві дії завжди, використовуючи при цьому додаткове місце у пам'яті пристрою.

1.4.3. Стратегія відсікання L3

Стратегія L3 базується на ідеї завчасного відсікання множин безперспективних шляхів наступним чином. Для кожного елементу розраховується його потенційна додатна вага функціоналу, яка дорівнює сумарній вазі функціоналу усіх елементів, що мають вищий порядковий номер. При побудові векторів, починаючи з першого рангу, розраховується чи перевищує його вага функціоналу разом з потенційною вагою функціонала його останнього елементу вагу функціоналу еталонного вектора (зазвичай першого вектора рангу). Якщо це так, то на основі даного вектора можуть бути сформовані вектори більш високого рангу. Якщо така вага не перевищує еталонну, то слід розуміти, що розгляд будь-якого вектора, який може бути побудований на основі поточного, є надмірним, оскільки не дасть кращого результату. Отже такий вектор може бути виключений зі списку. Такий підхід економить пам'ять, але, що важливіше, він скорочує кількість векторів, які мають бути сформовані і проаналізовані.

Формальний опис цієї стратегії наведено у [5]. Програмна реалізація L3 зводиться до того, що для кожного елементу у початковому масиві вираховується його потенційна вага функціоналу як сума ваг функціоналів всіх наступних елементів. Причому вигідно розраховувати її, починаючи з останнього елементу, таким чином для кожного елементу виконується тільки одна операція додавання.

При створенні нового вектора виконується одна додаткова перевірка, а саме – перевіряється чи перевищує вага функціоналу нового вектора разом із потенційною вагу функціоналу першого вектора у списку і, відповідно, якщо список пустий, то дана перевірка не виконується. Недоліком цієї стратегії є те, що згідно з нею можна відкинути тільки один вектор і вектори, що будуються на його основі, за один раз. Тому було запропоновано модифіковану версію даної стратегії.

Модифікована стратегія базується та тих же принципах, що і звичайна L3, проте відрізняється у деяких окремих правилах відсікання.

На першому ранзі розглядаються вектори, що складаються з одного елементу. Оскільки вони фактично повторюють вхідний масив, то вони є одразу відсортованими за вагою функціоналу. Це означає, що якщо один з векторів за прогнозованою вагою функціоналу не перевищує ваги функціоналу першого елементу, то його та всі наступні вектори і всі, засновані на них, можна не розглядати.

На другому ранзі:

- якщо вектор містить перший елемент, то він завжди, за будь яких умов, є важчим за функціонал, за усі вектори, які побудовані після нього, не зважаючи на те, чи містять ці вектори перший елемент. Наприклад, вектор, що містить елементи з номерами 1 та 3, завжди буде важчим за вектори (2,3), (1,4), (2,4) і так далі. Тому, якщо на другому ранзі вектор, що має в своєму складі перший елемент, відкидається за L3, то інші вектори рангу можна не

розглядати, оскільки ані їх фактична вартість, ані потенційна не може перевищити такі у даного вектора;

- якщо вектор починається не з першого елементу, то його відкидання згідно зі стратегією L3 має призводити до кінця розгляду векторів поточної комірки. Наприклад, якщо вектор (2,5) був відкинутий, то жодний з векторів (3,5), (4,5) не зможе дати кращий результат. Отже комірка може бути закрита.

На третьому і більш високих рангах:

- якщо відкидається вектор, що містить перший елемент, то використовується та ж логіка, як і на другому ранзі при відкиданні вектора без першого елементу. Тобто переривається побудова векторів комірки;
- якщо відкидається вектор, що не містить першого елементу, то може бути відкинутий тільки сам цей вектор.

Таким чином, може бути відкинута значна кількість безперспективних векторів без необхідності формування та перевірки самих цих векторів.

1.5. Алгоритми A1 та A2

На основі зазначених стратегій відсікання та узагальненої процедури A0 будуються алгоритми пошуку найкращих рішень задач ЦЛП з БЗ.

В даній роботі розглядаються два алгоритми пошуку наближеного рішення A1 та A2, які засновано, відповідно, на комбінаціях A0, L1, L3 та A0, L2, L3. У формальному вигляді ці алгоритми представлено у [8]. В даному дослідженні аналізуватиметься вплив програмних реалізацій даних стратегій на наступні метрики роботи алгоритмів A1 та A2:

- похибка (різниця між знайденою відповіддю та точною);
- кількість проаналізованих векторів.

1.6. Експериментальне дослідження алгоритмів рангового підходу до вирішення задачі цілочисельного лінійного програмування з булевими змінними

1.6.1. Умови проведення експерименту

Було складено вісім варіантів експериментальної програми, по чотири реалізації кожного з алгоритмів. А саме:

- алгоритм A1 в реалізаціях зі стратегіями:
    - L1 (з сортуванням), L3 (без модифікацій);
    - L1 (без сортування), L3 (без модифікацій);
    - L1 (з сортуванням), L3 (модифікована);
    - L1 (без сортування), L3 (модифікована);
- алгоритм A2 в реалізаціях зі стратегіями:
    - L2 (з сортуванням), L3 (без модифікацій);
    - L2 (без сортування), L3 (без модифікацій);
    - L2 (з сортуванням), L3 (модифікована);
    - L2 (без сортування), L3 (модифікована).

Зазначені варіанти програми також у табл. 1.

Таблиця 1

Варіанти реалізації тестової програми

A1 / A2			
L3(без модифікацій)		L3(модифікована)	
L1(з сортуванням)	L1(без сортування)	L1(з сортуванням)	L1(без сортування)

Умови проведення експерименту: Ноутбук Acer Aspire 3, cpu Intel Core i3. Доступна кількість оперативної пам’яті 2,2 Гб. Розмірність задачі 150. Кількість тестів – 100, для кожної реалізації алгоритмів.

1.6.2. Результати дослідження

Результати дослідження за такими метриками, як похибка обчислення відповіді та кількість переглянутих векторів, наведено у зведених графіках.

На рисунках зібрано по чотири графіки тих експериментів, де використовувалася одна і та ж стратегія відсікання L1 або L2, або, простими словами, – графіки для всіх реалізацій A1 та A2. На рис. 2 приведено результати експериментів, щодо похибки обчислення при різних реалізаціях стратегій відсікання для алгоритму A1.

Як видно з рис. 2, здебільшого при використанні стратегії L1, тобто, в алгоритмі A1 незалежно від того, чи модифікована L3, похибка обчислення коливається від 0 до 7,6 відсотка. При цьому величина похибки практично не залежить від реалізації стратегій.

На рис. 3 наведено результати експериментів щодо похибки обчислення при різних реалізаціях стратегій відсікання для алгоритму A2.

На рис. 3 видно, що незалежно від реалізації стратегій відсікання похибка розрахунків за алгоритмом A2 не перевищує 6,3 %. Однак можна помітити тенденцію, що у приблизно половині тестів реалізація стратегії L2 через сортування виявилася точнішою, хоча і незначно (в межах 1,5 %). В решті тестів величина похибки не змінюється в залежності від програмної реалізації стратегій.

Другою розглядуваною метрикою є кількість проаналізованих векторів. На рис. 4 приведено результати експериментів щодо кількості проаналізованих векторів при різних реалізаціях стратегій відсікання для алгоритму A1.

З рис. 4 видно цікаву тенденцію. Будь-яка з реалізацій пропонує значне зменшення кількості проаналізованих векторів порівняно з класичним випадком для цієї задачі, тобто 2^{150}, пропонує розглядати максимум 173000 векторів. Проте набагато цікавішим є те, що при тому, що вплив на точність практично відсутній, запропонована модифікація стратегії відсікання L3 зменшує кількість проаналізованих векторів в середньому на 40000.

На рис. 5 наведено результати експериментів щодо кількості проаналізованих векторів при різних реалізаціях стратегій відсікання для алгоритму A2.

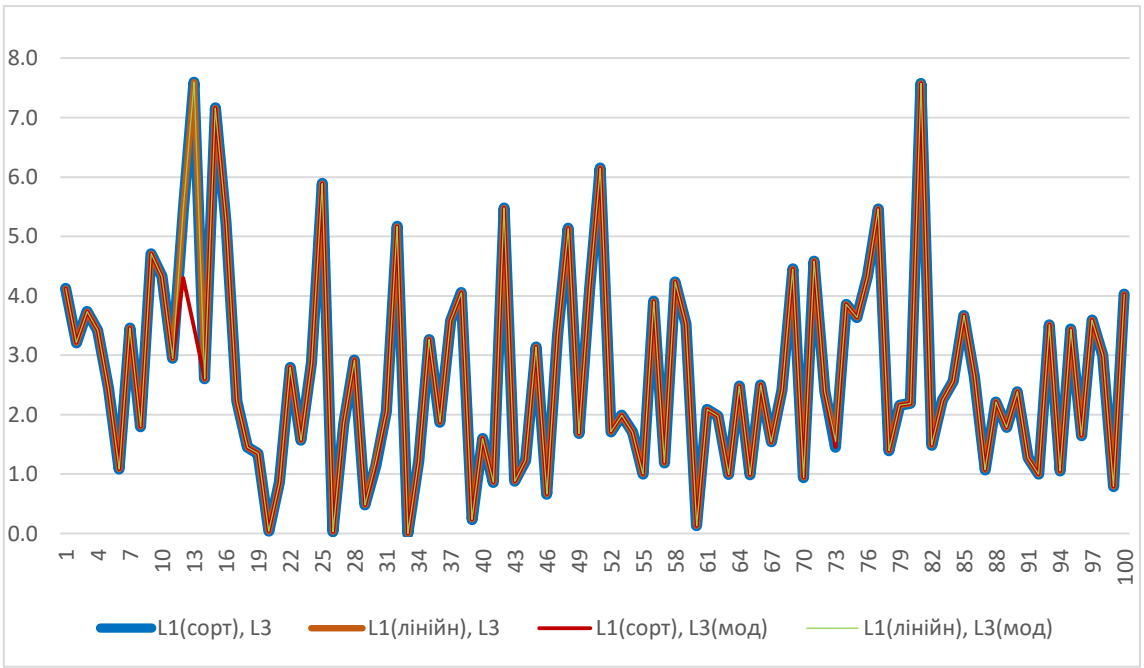

Рис. 2. Експериментальна оцінка похибки обчислення алгоритму A1
при різній реалізації стратегій відсікання

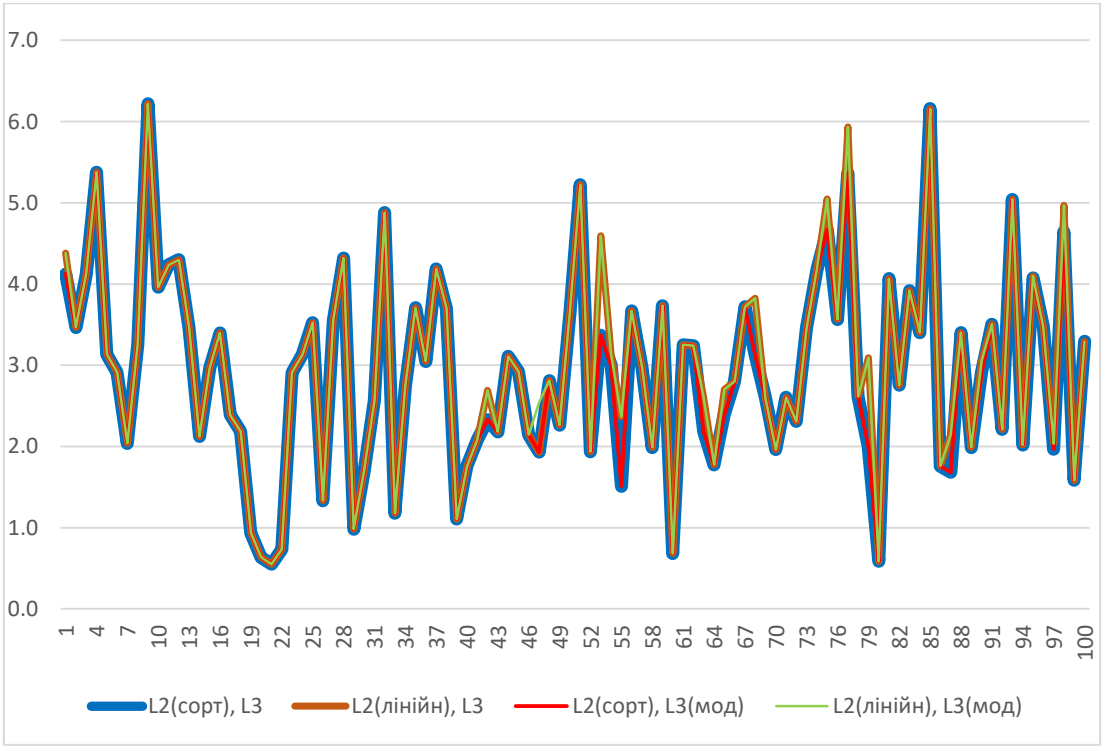

Рис. 3. Експериментальна оцінка похибки обчислення алгоритму А2
при різній реалізації стратегій відсікання

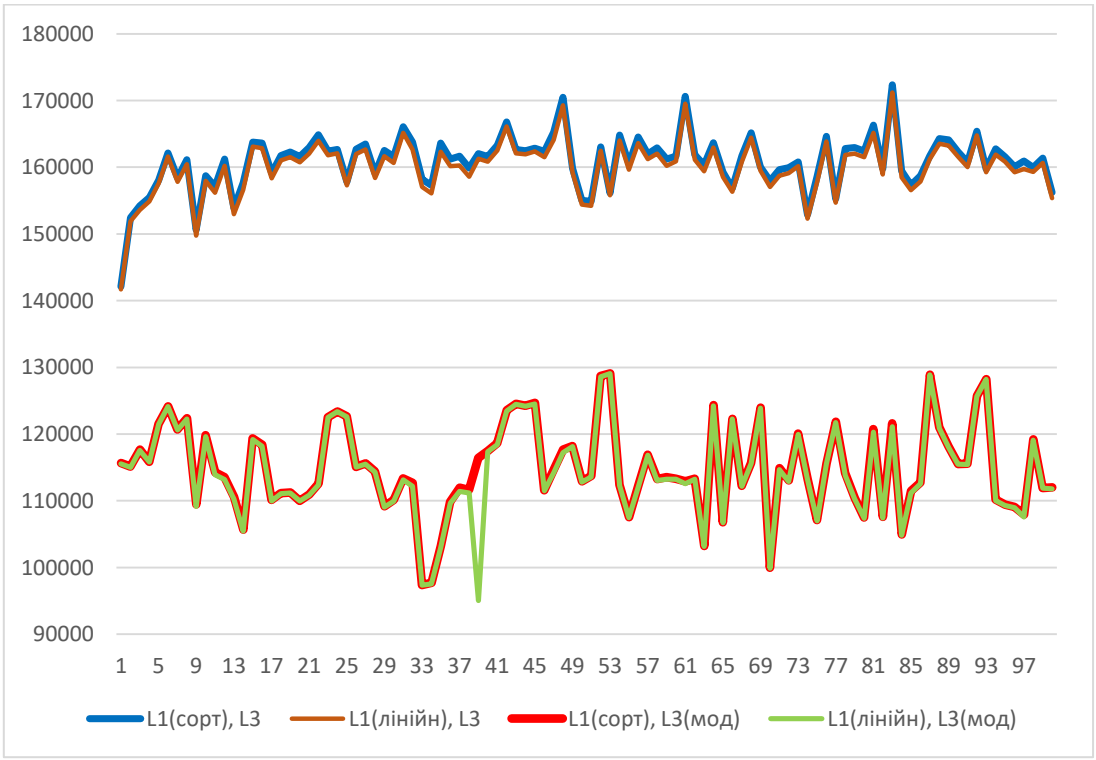

Рис. 4. Експериментальна оцінка кількості проаналізованих векторів алгоритму А1
при різній реалізації стратегій відсікання

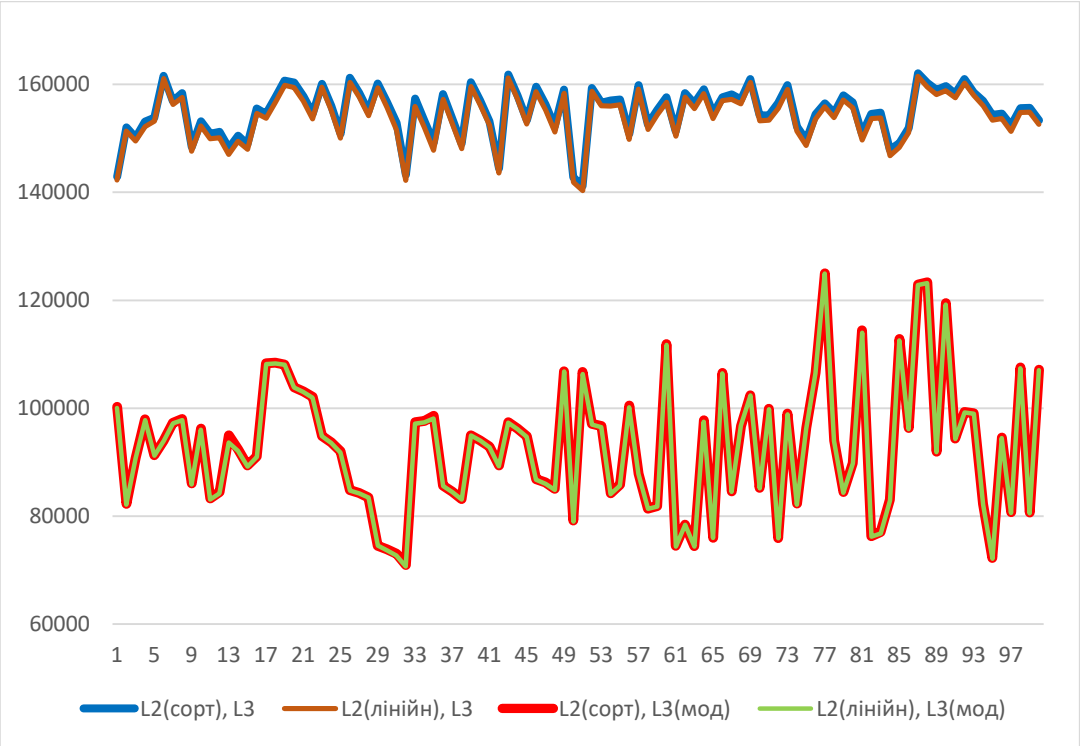

Рис. 5. Експериментальна оцінка кількості проаналізованих векторів алгоритму А2 при різній реалізації стратегій відсікання

З рис. 4 та 5 можна зробити висновок, що алгоритм А2 в середньому розглядає меншу кількість векторів ніж алгоритм А1. При цьому в обох алгоритмах використання модифікованої версії стратегії відсікання безперспективних наборів дає суттєвий виграш у кількості переглянутих векторів близько до 40000.

На обох рисунках можна помітити, що вплив реалізації стратегій L1 та L2 незначний (у межах 1000 векторів).

Висновки

1. Встановлено, що задача планування розподілу ресурсів у мережі (багатопроцесорній системі) може бути зведена до задачі цілочисельного лінійного програмування з булевими змінними (ЦЛП з БЗ). Такий підхід дозволяє враховувати як обмеження на ресурси, так і потребу в резервуванні, що підвищує надійність роботи інформаційно-комунікаційних систем. Особливістю даної задачі є необхідність її багаторазового вирішення в умовах деградації мережі в режимі реального часу.

2. Відомо, що задача цілочисельного лінійного програмування з булевими змінними належить до класу NP-повних задач. Це означає, що для неї не існує поліноміального алгоритму, що гарантував би рішення за визначений час. Тривалість обчислень залежить від розмірності задачі, яка визначає кількість можливих комбінацій рішень (векторів), які необхідно переглянути. У зв'язку з цим актуальними є методи пошуку наближених рішень, які дозволяють зменшити обчислювальні витрати, при цьому забезпечуючи прийнятний рівень точності.

3. Аналіз реалізацій запропонованих алгоритмів та стратегій відсікання безперспективних варіантів рішень дозволяє зробити такі висновки:

• використання алгоритму А1 дає похибку від 0 до 7,6 %, причому цей показник практично не залежить від модифікації L3;

• алгоритм А2 забезпечує похибку не більше 6,3 %, причому використання стратегії L2 реалізованої через сортування у деяких випадках дає більш точні результати (похибка менша на приблизно 1,5 %);

• всі запропоновані реалізації значно зменшують кількість переглянутих векторів порівняно з класичним випадком. Так, для задачі, яка при повному переборі вимагала б перегляду 2^{150} векторів, розглядається максимум 173000 векторів. При цьому модифікація стратегії відсікання L3 дозволяє скоротити кількість переглянутих векторів у середньому ще на 40000;

• виявлено, що алгоритм А2 в цілому розглядає менше векторів, ніж алгоритм А1. Використання модифікованої стратегії L3 у обох алгоритмах забезпечує додаткове скорочення кількості проаналізованих векторів;

• виявлено, що вплив програмної реалізації стратегій відсікання безперспективних шляхів на кількість проаналізованих векторів незначний і коливається в межах 1000 векторів.

Таким чином, результати дослідження підтверджують ефективність запропонованих стратегій у зменшенні обчислювальної складності задачі (в контексті зменшення кількості варіантів рішень, що потребують розгляду) при збереженні прийнятного рівня точності (похибка не перевищує 10 %). Рекомендується використання алгоритму А2, реалізованого з використанням стратегії L1 через лінійний пошук, та модифікованої стратегії L3.

Список літератури:

1. Laabadi S., Naimi M., El Amri H. and Achchab B. The 0/1 Multidimensional Knapsack Problem and Its Variants: A Survey of Practical Models and Heuristic Approaches // American Journal of Operations Research. 2018. No 8. P. 395–439.

2. Listrovoy S. V., Tretiak V. F., & Listrovaya A. S. Parallel algorithms of calculation process optimization for the boolean programming problems // Engineering Simulation. 1999. No 5. P. 569–579.

3. Третяк В., Голубничий Д., Коломійцев О., Мегельбей Г., Возний О., & Філіпенков О. Математична модель рангового підходу // Зб. наук. пр. ΛΟΓΟΣ,. 2020. P. 116–122. https://doi.org/10.36074/25.12.2020.v1.40

4. Коломійцев О., Осієвський С., Третяк В., Закіров З., Романюк А., Нікітченко В., Логвиненко Є., Лисиця А. Задачі дискретної оптимізації та їх постановка // InterConf. 2021. №75. С. 285–302. https://doi.org/10.51582/interconf.19-20.09.2021.033

5. Голубничий Д., Коломійцев О., Осієвський С., Третяк В., Рибальченко А., Любченко О., Головченко О. Метод відсікання безперспективних варіантів для задач цілочисельного лінійного програмування з булевими змінними з використанням рангового підходу // Наук. зб. «InterConf+». 2024. №41(185). С. 526–555. https://doi.org/10.51582/interconf.19-20.01.2024.064

6. Коломійцев О., Голубничий Д., Рибальченко А., Третяк В., Осієвський С., Возний О., Балабуха О., Качуровський Г., Грічанюк О., Галашевський Г., Сокова Т., Любченко О. Використання медодів рангового підходу в моделі транзакційної системи з реплікацією фрагментів бази даних для розгортання у хмарному середовищі // Scientific Collection «InterConf+». 2023. №38(175). С.326–341. https://doi.org/10.51582/interconf.19-20.10.2023.030

7. Голубничий Д., Головченко О. Features of the software model of the contracted graph in the (0,1)-knapsack problem // Proceedings of the Seventh International Scientific and Technical Conference "Computer and Information Systems and Technologies" Kharkiv : NURE, 2024. С. 25–26.

8. Пономаренко В.С. Цілочисельне програмування в економіці / В.С. Пономаренко, Д.Ю. Голубничий, В.Ф. Третяк. Харків : ХНЕУ, 2005. 204 с.

Надійшла до редколегії 15.11.2024

Відомості про авторів:

Голубничий Дмитро Юрійович – канд. техн. наук, доцент, Харківський національний університет радіоелектроніки, доцент кафедри електронних обчислювальних машин, факультет комп'ютерної інженерії та управління; Україна; e-mail: dmytro.holubnychyi1@nure.ua; ORCID: https://orcid.org/0000-0002-6873-7004

Головченко Олександр Сергійович – Харківський національний університет радіоелектроніки, магістрант кафедри електронних обчислювальних машин, факультет комп'ютерної інженерії та управління; Україна; e-mail: oleksandr.holovchenko@nure.ua; ORCID: https://orcid.org/0009-0002-7582-1746