Про підходи до прискорення перебору об’єктів в задачах обробки інформації

Олександр Гайша[1], *Сергій Лєнков*[2], *Олена Гайша*[3]

[13]Міжнародний класичний університет імені Пилипа Орлика
вул. Котельна, 2, Миколаїв, Україна, 54003
[2]Військовий інститут Київського національного університету імені Тараса Шевченка
Юлії Здановської, 81, Київ, Україна, 03189
[1]oleksandrgaysha1982@gmail.com, https://orcid.org/0000-0003-3711-547
[2]lenkov_s@ukr.net, https://orcid.org/0000-0001-7689-239
[3]haishaolena@gmail.com, https://orcid.org/0009-0000-4543-912

Received 03.02.2025, accepted 29.03.2025
https://doi.org/10.32347/st.2025.3.1207

Анотація. У роботі докладно розглянуто поняття процесу перебору. Він здійснюється з об'єктами x, які є елементами деякої множини X. Метою перебору є виконання деякої активної дії y = f(x) над елементами, що перебираються. Ця дія може, звичайно, застосовуватися і до кожного елемента, але у переважній більшості практичних завдань дія застосовується тільки до тих елементів, які задовольняють певній умові перебору P(x).

Найбільш простим способом перебору є повний перебір або «брут-форс». При цьому до кожного без винятку елементу множини X застосовується перевірка умови P(x) і, залежно від результату, здійснюється виконання дії y = f(x). Очевидно, що повний перебір є певним еталонним процесом у тому сенсі, що саме в порівнянні з ним зручно оцінювати ефективність того чи іншого алгоритму обмеженого перебору. Алгоритми обмеженого перебору доцільно застосовувати у переважній більшості практичних випадків, якщо тільки повний перебір не займає малопомітний час порядку часток секунди. Однак, у деяких випадках доцільно оптимізувати (обмежувати) перебір навіть за таких малих часів виконання всього процесу – зокрема, якщо такі короткі процеси перебору повторюються багато разів на одиницю часу, складаючись сумарно у деякий, уже досить значний час.

Метод гілок і меж є найвідомішим алгоритмом обмеженого перебору і дозволяє, поступово просуваючись від одного варіанта до іншого, відкидати цілі набори елементів, завідомо не відповідних умові P(x). Однак, даний метод не дозволяє робити надійні попередні оцінки ефективності обходу всього дерева відповідних елементів, а його застосовність для кожної конкретної задачі стає зрозумілою тільки в процесі перебору.

Гайша Олександр
к.т.н., доцент, доцент кафедри інженерних технологій

Сергій Лєнков
д.т.н. професор, головний науковий співробітник

Гайша Олена
ст. викладач кафедри інженерних технологій

У цій роботі пропонується метод обмеження, заснований на поданні умови перебору P(x) у кон’юнктивній або диз’юнктивній нормальній формі з подальшою перевіркою цієї умови за схемою скорочених логічних обчислень з використанням лише однієї з умов. Також для перевірки цієї умови пропонується заздалегідь проводити обчислення деякої додаткової інформації, спираючись на яку під час перебору, можна отримувати кінцевий результат швидше. В якості такої інформації можна прийняти поділ на класи (кластери) всіх об'єктів, що перебираються, і виконання переборів тільки в

межах кластерів, що скорочує загальну кількість елементів, що перебираються.

Пропонований підхід докладно пояснюється на прикладі рішення актуальної задачі з області комп'ютерної графіки, яка полягає у пошуку всіх правильних n-кутників (на прикладі правильних трикутників) на множині N точок тривимірного простору, які задані своїми координатами. Наведено словесний опис повного перебору, необхідного для вирішення такого завдання, а також докладний опис алгоритму обмеженого перебору, який складений за пропонованою схемою (скорочені обчислення за КНФ і попередня кластеризація всіх об'єктів, що перебираються або, точніше, пов'язаної з ними інформації – відстаней між усіма парами).

У роботі наведено числові розрахунки, що дозволяють провести теоретичну оцінку ефективності запропонованого рішення. Встановлено, що при ідеально рівномірному розподілі всіх об'єктів, що перебираються по K класам, відбувається значне зменшення кількості операцій порівняння відстаней при пошуку рівносторонніх трикутників (при числі K = 50 – теоретичне прискорення становить до 3 разів). Прискорення, що одержується, досліджено практично на модельній програмній реалізації, написаній мовою C#, в результаті чого показано, що реальне прискорення є досить близьким до ідеального, а сам запропонований підхід, відповідно, є ефективним.

Ключові слова: перебір, прискорення перебору, кластеризація, декомпозиція умов перебору.

ВСТУП

На сьогоднішній день існує значна кількість предметних областей, де застосовуються алгоритми перебору. Насамперед, з обчислювальної точки зору найскладніші перебори зустрічаються у завданнях пошуку інформації у базах даних (БД) [1], що розглянемо докладніше.

По-перше, такі завдання характеризуються значними обсягами даних, серед яких потрібно проводити перебір. Якщо йдеться про програму, запущену в оперативну пам'ять і яка не використовує базу даних для зберігання та вилучення інформації, то, очевидно, оброблювані дані будуть розміщені в оперативній пам'яті ЕОМ, яка на сьогоднішній день для персональних комп'ютерів (далі – ПК) становить величини порядку десятка гігабайт. Для продуктивних серверів обсяги оперативної пам'яті можуть досягати сотень гігабайт, що є досить великою величиною, яка вимагає застосування спеціальних апаратних рішень (проти звичайної архітектури ПК). Однак, дані, які розміщуються в базах даних, навіть при використанні звичайних ПК легко можуть досягати зазначеного розміру в 100 Гб, а при розміщенні на високопродуктивних серверах обсяги даних, що зберігаються, обчислюються сотнями терабайт, що в цілому на кілька порядків більше обсягів даних, які можна розміщувати в оперативній пам'яті ЕОМ. Таким чином, виконання операцій на даних, розміщених у БД, потенційно може потребувати на кілька порядків більше дій, ніж застосування того самого алгоритму в програмах, що не використовують підключення до БД [2].

Другим аспектом застосування алгоритмів пошуку (а також і сортування) на даних, розміщених саме в БД, є необхідність урахування того, що доступ до даних, розміщених на диску (а файли БД природно зберігаються в дисковій пам'яті, а не оперативній), завжди здійснюється значно повільніше, ніж до даних в оперативній пам'яті ЕОМ. Навіть сучасні швидкі SSD накопичувачі мають значно меншу швидкість доступу до даних, ніж пам'ять класу DDR5 або навіть DDR4. Також, відомо, що дискові накопичувачі оптимізовані для потокової передачі/прийому даних, а оперативна пам'ять дозволяє швидко отримувати доступ до невеликих розрізнених блоків даних. Це означає, що при інтенсивних процесах додавання/видалення даних з БД її файли будуть сильно фрагментовані, що ще гірше уповільнить роботу з нею. Таким чином, задачі оптимізації алгоритмів, що працюють із базою даних, слід приділяти велику увагу.

Ще одним аспектом організації пошуків (тобто переборів) саме в базах даних є особливості процедури отримання даних при дуже великих запитах. Використання стандартної процедури вибірки чергового

елемента, що задовольняє умові раніше ініційованого пошуку (наприклад, в системі MySQL спочатку один раз виконується команда mysqli_query(), а потім багато разів викликаються команди на кшталт mysqli_fetch_array()), при великій кількості елементів, що задовольняють умовам пошуку, може настільки загальмувати роботу з СУБД, що весь комп'ютер-сервер може перейти в заблокований стан [3].

Усі наведені моменти свідчать, що оптимізація процесу перебору є особливо актуальною при роботі з великими даними, які очевидно розміщуються в базах даних. В такому випадку перехід від повного до часткового (обмеженого) перебору є завданням, обов'язковим до виконання, звичайно за наявності відповідних можливостей (існування методів, їх реалізації і існування можливостей їх адекватного застосування).

Вся наведена аргументація стосувалася переборів у БД, однак і при розміщенні даних у пам'яті ЕОМ задача скорочення кількості елементів, що перебираються, може бути надзвичайно актуальною за наявності деяких особливостей всього процесу.

По-перше, самі елементи, що перебираються, в одних задачах можуть бути досить простими, як, наприклад, при задачі пошуку в числовому одновимірному масиві, але в інших випадках можуть являти собою складні, комплексні і значні за обсягом структури даних. У першому випадку навіть великі перебори інтегрально будуть задачами зі значним, але не гігантським обсягом обчислень. У другому ж загальна складність всього перебору значною мірою залежатиме від складності об'єктів, що перебираються. Таким чином, при переборі елементів, що належать до складно влаштованих структур даних, питання про обмеження перебору також стає дуже актуальним.

По-друге, хоча об'єкти, що перебираються, і можуть бути простими за своєю суттю (як наприклад, при переборі всіх елементів цілісного статичного масиву), проте умова, що перевіряється при переборі всіх елементів масиву, може бути обчислювально складною. Очевидно, що і тут актуальне (по можливості) скорочення множини елементів, що реально перебираються. Також, якщо перебір виконується не з метою пошуку, а застосування якоїсь операції до всіх без винятку елементів базової множини (у вказаному щойно випадку-прикладі – до всіх елементів масиву), то за умови, що ця операція є досить «важкою», обмеження перебору також повинно бути обов'язковим етапом всього процесу. Слід зазначити, що під «обмеженням перебору» тут потрібно розуміти не зменшення кількості об'єктів, що перебираються, але скорочення, по можливості, числа виконуваних при цьому операцій.

Наприклад, порахувавши і використавши результат тривалого перетворення f(ai) для поточного елемента масиву ai слід не видаляти його, а помістити в тимчасове сховище {f(ai)}. Якщо один із наступних елементів масиву випадково виявиться рівним ai, то вираховувати f(ai) не потрібно, а слід просто взяти готове значення з тимчасового сховища. Очевидно, що можливі інші шляхи скорочення обсягу виконуваних операцій, які в цілому можна віднести до алгоритмів обмеженого перебору, але які конкретно кроки в даному напрямку можна зробити - залежить від прикладного завдання.

Нарешті, можлива комбінація негативних факторів, коли:

- перебір необхідно здійснити на безлічі дуже складних за своєю структурою об'єктів;
- потрібна складна перевірка умови перебору
- сама операція, що застосовується до об'єктів, що відбираються при переборі, також дуже складна.

Слід пам'ятати, що складність тут скрізь йдеться у обчислювальному сенсі, тобто «складний»=«такий, що вимагає великих обсягів обчислень».

Досі бралися до уваги завдання, що відносяться великою мірою до галузі Big Data, де обсяги оброблюваних даних обчислюються мінімум гігабайтами, а часто

навіть терабайтами і більше. При цьому одне таке завдання, будучи запущеним на певній ЕОМ (або кластерній архітектурі), вирішується протягом тривалого
часу. Такі завдання можуть стосуватися фундаментальних наукових досліджень, важливих інноваційних технічних розрахунків, макроекономічних досліджень, тощо. Одним словом, вони є глобальними за своєю суттю (стратегічні). У той самий час існує дуже багато прикладних завдань середньої складності, вирішення яких потрібно отримувати майже миттєво, наприклад, прогнозування біржових коливань курсів валют і цінних паперів, прогноз погоди на найближчі години, розпізнавання особи людини, яку видно в даний момент через камеру зовнішнього спостереження, тощо. У цьому випадку задача хоча і не вимагає гігантських обсягів обчислень, проте існує жорсткий ліміт на загальний час її виконання, і крім того, отримувати рішення різних варіантів такої задачі потрібно досить часто, тобто. практично у потоці. Очевидно, що виграш у 0,1 сек при розпізнаванні однієї особи є несуттєвим, однак, якщо система повинна розпізнавати потік людей, що йдуть по жвавій вулиці, ситуація кардинально змінюється і навіть таке мале поліпшення в результаті призводить до сумарної відчутної економії та збільшення продуктивності системи.

Таким чином, обмеження перебору слід застосовувати у таких випадках:

- пошук/обробка інформації у джерелах Big Data, таких, як, наприклад, мережеві БД із загальним обсягом порядку терабайтів і вище;
- перебираються складні за своєю структурою об'єкти;
- здійснюється перевірка складної умови перебору;
- виконується складне перетворення елементів при переборі;
- сам собою перебір не дуже складний і може бути виконаний за цілком розумний час (наприклад, порядку секунд або хвилин), але у відповідній галузі такі завдання перебору виникають безперервно і постійно (потоком).

Як видно, спектр завдань, в яких потрібно застосування алгоритмів обмеження перебору, досить широкий, тому можна переходити до більш щільного обговорення цих методів.

АНАЛІЗ ОСТАННІХ ДОСЛІДЖЕНЬ ТА ПОСТАНОВКА ЗАДАЧІ

Насамперед слід зазначити, що сама галузь, пов'язана з алгоритмами взагалі та алгоритмами перебору зокрема, з'явилася порівняно недавно. Так, основи теорії алгоритмів почали розроблятися у 1930-х роках (перші роботи були створені в 1930-х і 1940-х роках Куртом Геделем, Аланом Тюрінгом, Емілем Постом, Алонсо Черчем та А.А. Марковим) [4]. У той самий час (30-40-ті рр.) закладаються і основи теорії оптимізації на роботах фон Неймана, Данцига, Канторовича та інших [5].

Робота з множинами об'єктів поступово стала типовим процесом при обробці однорідних масивів інформації, хоча різні варіанти цього процесу, звичайно, можуть відрізнятися значною мірою особливостями технічної реалізації для різних прикладних завдань. Природно з'явилися й різноманітні способи переходу від одного об'єкта до іншого та перевірки їх якостей чи виконання деяких процесів. Узагальнено такі методи називають алгоритмами перебору [6]. Найпростішим є повний перебір чи перебір з допомогою грубої сили (brute-force).

Для проведення будь-якого перебору потрібно провести кілька підготовчих процедур [7], а саме:

а) встановити деякі правила встановлення порядку у безлічі елементів, що підлягають перебору (адже, як відомо, математичне
поняття «множина» є невпорядкованим набором об'єктів). У найпростішому, але дуже поширеному випадку, йдеться про просту нумерацію, тобто всі елементи потрібно лише пронумерувати, зокрема призначити перший і останній з них. У більш складних випадках порядок може встановлюватися на підставі інших ознак (наприклад, від більшого елемента до меншого, за кольорами веселки, частотою вживання на практиці тощо);

б) також необхідно створити спосіб переходу від поточного аналізованого при переборі
елемента до наступного, що слідує безпосередньо за ним;
в) розробити і реалізувати суть активної дії y = f(x), яку потрібно проводити над об'єктами xi, що перебираються (наприклад, формувати з них нову множину відібраних об'єктів, або просто видавати їх на екран, або застосовувати
до кожного якесь перетворення і перезаписувати його, і т.п.);
г) зафіксувати умову перебору P(x), тобто встановити ряд вимог, яким повинен задовольняти поточний об'єкт, що розглядається, щоб застосовувати до нього активну дію, описану в попередньому пункті.

Зазначена послідовність дій на псевдокоді, наближеному до C/C++, матиме такий вигляд:

```
x = First;
while (x! = Last)
{
if(P(x) == true)
{
y = f(x);
use(y);
}
x++;
}
```

Слід зазначити, що в доступній літературі зустрічається чимало варіантів алгоритмічних дій, які включають слово «перебір». У цьому можуть описувати досить далекі друг від друга предметні області.

Так, наприклад, термін «метод перебору» в [8] визначається як найпростіший з методів пошуку значень дійсно-значних функцій за будь-яким із критеріїв порівняння (на максимум, на мінімум, на певну константу). Стосовно екстремальних завдань перебір є прикладом прямого методу умовної одновимірної пасивної оптимізації. Даний метод також називають методом рівномірного пошуку або перебором по сітці.

Вочевидь, у реальних задачах перебору є можливість зменшити (причому часто – досить істотно) кількість виконуваних дій. Такі перебори ми називатимемо обмеженими, а сам процес зменшення числа дій, порівняно з повним перебором, називатимемо обмеженням перебору. При цьому можна розділяти два варіанти запровадження обмежень:

- зменшити кількість об'єктів, що перебираються, зберігаючи складність процедур P(x) і y = f(x), чого можна досягти в багатьох реальних застосунках, враховуючи їх особливості. Також для скорочення перебору в деяких випадках слід провести додаткові дії, що не входять до процесу повного перебору, однак їхня загальна складність має бути меншою, ніж складність перебору тих об'єктів, які можна відкинути в результаті проведення цих додаткових дій;
- спростити процедуру P(x) або y = f(x), або обидві відразу, зберігаючи множину об'єктів, що перебираються.

Очевидно, що можливі різні комбінації двох запропонованих шляхів скорочення кількості операцій під час перебору. Наприклад, можливе спрощення умови P(x) і, як наслідок, зменшення числа об'єктів, що перебираються. Або можливе розбиття процедури обробки y = f(x) на етапи, за результатом першого з яких можна припиняти подальшу обробку деяких елементів, фактично скорочуючи кількість елементів, що перебираються (якщо розуміти під перебором застосування всієї процедури y = f(x), а не тільки її частини).

Найбільш відомим методом, який обмежує перебір, є метод гілок та меж [9]. Слід зазначити, що конкретний зміст виконуваних у цьому методі дій залежить від предметної області і саме тому цей метод і навіть групу методів ще називають:

- перебір із відходом назад чи з поверненням;
- бектрекінг (англ. backtracking);
- пошук у глибину;
- метод спроб і помилок, і т.д.

Метод гілок та меж (англ. branch and bound) – загальний алгоритмічний метод для знаходження оптимальних рішень різних завдань оптимізації, особливо дискретної та комбінаторної. Є розвитком методу

повного перебору, але на відміну від останнього тут виконується відсів підмножин допустимих рішень, які завідомо не задовольняють умову P(x).

Метод гілок і меж вперше запропонований в 1960 Алісою Ленд і Елісон Дойг [10] для вирішення завдань цілочисельного програмування.

Загальна ідея методу може бути описана на прикладі пошуку мінімуму функції f(x) на множині допустимих значень змінної x. Функція f та змінна x можуть бути довільної

природи. Для методу гілок та меж необхідні дві процедури:

- розгалуження;
- знаходження оцінок (меж).

Метод використовується для вирішення деяких NP-повних завдань, у тому числі задачі комівояжера та задачі про ранець.

Метод гілок та меж використовує перевірку умови P(x) на основі інформації, яка обчислюється безпосередньо під час перебору, що погіршує його продуктивність. Теоретично можна відсікати невідповідні варіанти ще до початку процедури перебору для чого слід вираховувати якусь додаткову інформацію, яка безпосередньо і не потрібна для початкової задачі самого перебору. У даній статті розвиватиметься саме такий підхід, коли скорочення операцій при самому переборі досягається з допомогою виконання надлишкових додаткових операцій перед стартом процедури перебору. Це дозволяє розраховувати цю додаткову вхідну інформацію заздалегідь, а, можливо, і на іншій ЕОМ. При цьому сам перебір займатиме відчутно менший час порівняно з брут-форс варіантом.

ОСНОВНА ЧАСТИНА

Отже, розглянемо докладніше поняття алгоритму обмеженого перебору.

Як встановлено, важливими складовими будь-якого алгоритму перебору є дві речі:

- активне перетворення y = f(x), якому потрібно піддати об'єкти, що перебираються;
- умова перебору P(x), підставляючи в яку черговий елемент, що перебирається, встановлюється, чи потрібно піддавати його перетворенню y = f(x).

На перший погляд може здатися, що можливо працювати у цих двох напрямках: скорочувати час активних перетворень та скорочувати час перевірки умов. Однак, спроби декомпозиції та спрощення перетворення y = f(x) будуть безперспективними, тому що при вирішенні конкретної прикладної задачі тип цього перетворення і всі його складові вже задані особливостями предметної області і якось змінити обсяг операцій, які необхідно виконати для досягнення потрібного результату неможливо.

Що ж до другого напряму – то тут перспектива зменшення обсягів виконуваних операцій може бути набагато кращою. По-перше, в більшості реальних завдань умова перебору буде досить складною (складеною), а це означає, що її можна представити в одній із нормальних форм, що розглядаються в математичній логіці:

а) кон'юнктивна нормальна форма (КНФ):

$$P(x) = P_1(x) \wedge P_2(x) \wedge \ldots \wedge P_m(x) \qquad (1)$$

б) диз'юнктивна нормальна форма (ДНФ):

$$P(x) = P_1(x) \vee P_2(x) \vee \ldots \vee P_m(x) \qquad (2)$$

Позитивними рисами (1) та (2) є можливості скорочених логічних обчислень у деяких випадках (які не так і рідко зустрічаються на практиці):

а) для КНФ: якщо будь-яка складова всієї умови (тобто підумова) Pi(x) дорівнює нулю, то інші складові можна не обчислювати, а результат відразу дорівнює нулю (це автоматично випливає з властивостей операції логічної кон'юнкції, тобто операції «І»). Звідси випливає, що обчислювати умови в КНФ завжди краще з тих, які з найбільшою ймовірністю є хибними (тобто рівні нулю);

б) аналогічно якщо в ДНФ якась складова Pi(x) дорівнює одиниці, то інші підумови можна не прораховувати, так як весь результат автоматично дорівнює 1, що випливає із властивостей логічної операції

«АБО» (диз'юнкції). Отже, починати обчислення підумов у ДНФ слід із тих, які найімовірніше є істинними (тобто рівні одиниці).

Якщо логічна функція, що реалізується виразом P(x), має вигляд, який відрізняється від (1) або (2), то, як відомо, існують процедури для приведення будь-якої логічної функції до КНФ або ДНФ. Для визначеності (і не обмежуючи при цьому загальності) надалі (і зокрема у програмній реалізації) говоритимемо про КНФ, тобто. про набір логічних умов, пов'язаних між собою зв'язкою «І». За потреби від КНФ можна перейти до ДНФ за допомогою законів де Моргана та інших законів логіки алгебри.

Отже, якщо прикладна задача дозволяє розбивку умови P(x) на простіші умови, то виникають можливості щодо скорочення перебору, пов'язані із застосуванням кон'юнктивної (або диз'юнктивної) нормальної форми. Зазначимо, що спочатку (наприклад, для розв'язання задачі повного перебору) ніякої розбивки умови перебору на підумови може

бути не потрібно. Цілком можлива ситуація, коли розбивка умови P(x) на компоненти є штучною, і спеціально вводиться з метою скорочення перебору. Мало того, деякі з цих умов Pi(x) можуть вибиратися так, щоб, по-перше, з великою ймовірністю бути рівними нулю (у разі використання КНФ) або одиниці (якщо за основу взята ДНФ), а, по-друге, включати якусь обчислювально складну інформацію, яку можна розрахувати попередньо, перед стартом процесу перебору.

В останньому випадку виникають нові додаткові обчислення, які (цілком або частково) були відсутні при повному переборі, що на перший погляд навіть погіршує ситуацію. Однак, введення такої додаткової інформації може суттєво скоротити процедуру перебору, тому загальний час вирішення прикладного завдання знизиться значною мірою порівняно з повним перебором. В якості додаткової інформації пропонується використовувати розбиття на класи (тобто кластеризацію) самих об'єктів, що перебираються, за будь-якими відповідними властивостями, або пов'язаних з ними інших, допоміжних об'єктів, важливих для конкретної задачі перебору.

Кращою ілюстрацією запропонованого підходу, звісно, є конкретні приклади, реалізовані мовами програмування, і дають точний числовий результат часу виконання однієї й тієї прикладної задачі різними методами перебору. Проте, для того, щоб таку програму створити, потрібно поставити якісь конкретні задачі, так як описані вище міркування є досить загальними і повинні адаптуватися (підлаштовуватись) під кожну конкретну задачу своїм власним чином.

Отже, для побудови програмної реалізації (що буде виконано в даній роботі) розглянемо одне з поширених завдань комп'ютерної графіки [11], коли з безлічі точок, заданих своїми координатами, потрібно вибрати всі такі набори по n штук, які утворюють правильний n-кутник (полігон). Точки можуть розташовуватися на площині (тоді кожна задається двома числами-координатами) або в тривимірному просторі (тоді положення точки природно задається трійкою дійсних чисел), що істотно складніше з обчислювальної точки зору. Саме цей останній випадок і будемо використовувати (беремо важче завдання, щоб легше було оцінити різницю, одержувану із застосуванням запропонованого методу).

Для побудови першої версії програмного продукту обмежимося випадком n = 3, тобто. з множини N точок, заданих у тривимірному просторі за допомогою їх координат, вибиратимемо трійки тих, які утворюють правильні трикутники.

Умова перебору P(x) може бути у такому разі сформульовано різними способами:

- якщо два кути трикутника дорівнюють 60;
- якщо трикутник рівнобедрений і один (будь-який) з його кутів дорівнює 60;
- якщо всі три сторони рівні, і т.д.

Для програмної реалізації виберемо останню умову, тобто. рівність всіх трьох сторін (позначимо їх d1, d2, d3), що у програмі можна перевірити трьома попарними порівняннями, у результаті умова правильності трикутника природним

чином перетворюється на КНФ з трьох складових підумов наступного виду:

$$P(x) = P_1(x) \wedge P_2(x) \wedge P_3(x), \quad (3)$$

де x - трикутник, що задається, взагалі кажучи (як було зазначено вище), своїми трьома вершинами. Будемо називати їх Ai, Aj, Ak, тобто:

x = {Ai, Aj, Ak};
P(x) = "x - рівносторонній трикутник";
P1(x) = "сторони d1 і d2 рівні";
P2(x) = "сторони d2 і d3 рівні";
P3(x) = "сторони d3 і d1 рівні";
d1 = | AiAj |;
d2 = | AjAk |;
d3 = | AkAi |.

Рівності сторін, що перевіряються як Pi(x) у виразі (3) не можуть у комп'ютері виконуватися точно, як це відбувається у викладках в аналітичній математиці, адже відстані di будуть дробовими числами. Тому вже на даному етапі, при описі алгоритму, слід запровадити якийсь допуск □, і якщо різниця між двома числами (відстанями) буде меншою, ніж це значення, то два числа (відстань) можна вважати рівними:

$$d1 = d2 \quad \Leftrightarrow \quad | d1 - d2 | < \varepsilon \quad (4)$$

Наявність такого допуску викликає необхідність саме триразової перевірки рівності всіх пар сторін, адже якби довжини сторін задавалися точно, то було б достатньо лише два порівняння з трьох використовуваних.

Беручи до уваги введені позначення, можна сформулювати досить простий словесний опис

процедури повного перебору у задачі, що розглядається:

1) виконати перебір всіх можливих комбінацій трійок точок (природно без урахування їх порядку, тобто йдеться про поєднання, а не розміщення, перестановку та ін.);

2) для кожного випадку (тобто для кожної трійки точок) порахувати відстані між ними d1, d2, d3;

3) перевірити рівність (за формулою (4)) всіх трьох відстаней d1, d2, d3 та у разі збігу – видати ці точки (наприклад, їх номери) на екран.

На С-подібному псевдокоді ця процедура матиме вигляд:

```
for(int i=0;i<N;i++)
for(int j=i+1;j<N;j++)
for(int k=j+1;k<N;k++){
Calculate_d1(A[i], A[j]);
Calculate_d2(A[j], A[k]);
Calculate_d3(A[k], A[i]);
if (d1==d2==d3){
out(i,j,k);
  }
}
```

Слід зазначити, що цей код вже певною мірою оптимізований (тобто. перебір обмежений), оскільки перебір тут іде саме з усіх поєднань, тобто. без повторень, що досягається завданням початкових значень для змінних двох внутрішніх циклів не з нуля, а з поточного значення змінної зовнішнього циклу. Наприклад, у другому циклі змінна циклу ініціалізується значенням j = i + 1, а у внутрішньому циклі початкове значення відповідно дорівнює k = j + 1. Зазначимо, що тіло внутрішнього циклу буде виконано раз, і, наприклад, для 1000 точок це число становитиме 166 млн. повторень.

Перейдемо до опису скороченого перебору, який буде реалізований у цьому прикладі. Основна ідея тут полягає в тому, що порівнюються всі відстані поспіль, які навіть дуже різняться одна з одною. Доцільно перед початком перебору розділити на групи (кластеризувати) всі відстані між точками (запам'ятовуючи, до якого саме кластеру належить та чи інша пара точок з допомогою їх номерів). Тоді після такої попередньої підготовки (яка не потрібна при звичайному повному переборі) здійснювати пошук однакових відстаней d1, d2, d3 можна буде вже не по всьому набору точок, а тільки перебираючи трійки точок, між якими відстані лежать в межах одного кластера.

Загальна кількість відстаней між N точками дорівнює числу поєднань . Припустимо, всі відстані більш-менш рівномірно розподілилися по K кластерам, тоді в кожному з них буде по / K відстаней. З цього числа альтернатив (різних відстаней, що лежать у межах одного кластера) можна

було б вибирати по 3, перевіряючи їх на точну рівність за допомогою константи □, але кількість операцій можна значно скоротити, якщо скористатися скороченою процедурою обчислення КНФ і перевіряти рівність лише 2 відстаней. У більшості випадків вони будуть різні і подальше обчислення третьої відстані і перевірка на збіг з першими двома не знадобиться, тобто кількість таких перевірок буде дорівнює кількості поєднань з C_N^2 / K по 2, а не по 3:

$$C^2_{C_N^2/K} = \frac{C_N^2/K \cdot (C_N^2/K - 1)}{2} = \frac{C_N^2 \cdot (C_N^2 - K)}{2K^2}$$

Наприклад, для 1000 точок та 50 кластерів отримаємо таку кількість повторень процедури порівняння трьох відстаней:

$$C^2_{C^2_{1000}/50} = \frac{499500 \cdot (499500 - 50)}{2 \cdot 50^2} = 4{,}99 \cdot 10^7.$$

Як бачимо, кількість операцій скорочується приблизно в 3 рази (166 млн проти 50 млн порівнянь), хоча точною цю оцінку не можна вважати через наявність додаткових накладних витрат у другому методі. Таким чином, попередня теоретична оцінка показує гарне поліпшення продуктивності перебору в задачі, що розглядається при використанні запропонованого методу, в порівнянні з традиційним рішенням. Очевидно, що точні показники покращення часу виконання завдання потрібно отримувати на робочій програмній реалізації.

Реалізацію алгоритму перебору було виконано мовою C# у середовищі розробки Microsoft Visual Studio 2022 Community. Ефективність запропонованого рішення оцінювалася на модельному розподілі тисячі точок у тривимірному просторі, візуалізованому на рис.1. Скріншоти, на яких видно порівняльні результати роботи програми в режимах повного та обмеженого перебору, показані на рис. 2 (для допуску 1) та рис. 3 (для допуску 0,1).

На рис. 2 видно, що при допуску 1 повний перебір 1000 точок займає 119 секунд проти 65 секунд при використанні запропонованого алгоритму на 60 кластерах (при цьому знайдено близько 700 правильних трикутників).

На рис. 3 видно, що при допуску 0,1 повний перебір 1000 пікселів займає 109 секунд проти 67 секунд при використанні запропонованого алгоритму на 60 кластерах.

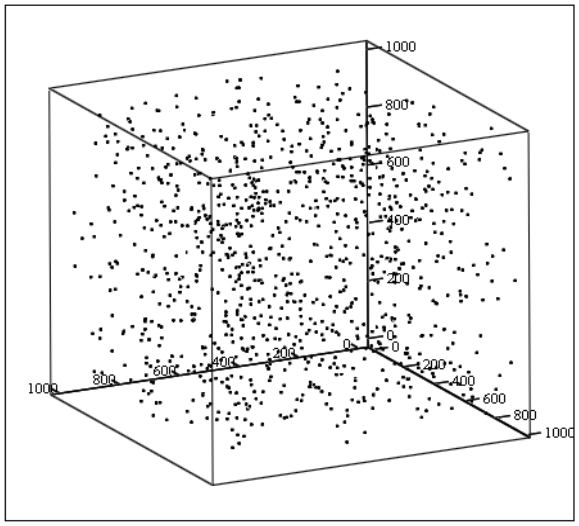

Рис. 1. Приклад генерації набору вхідних даних для проектованої програми за допомогою середовища MathCad

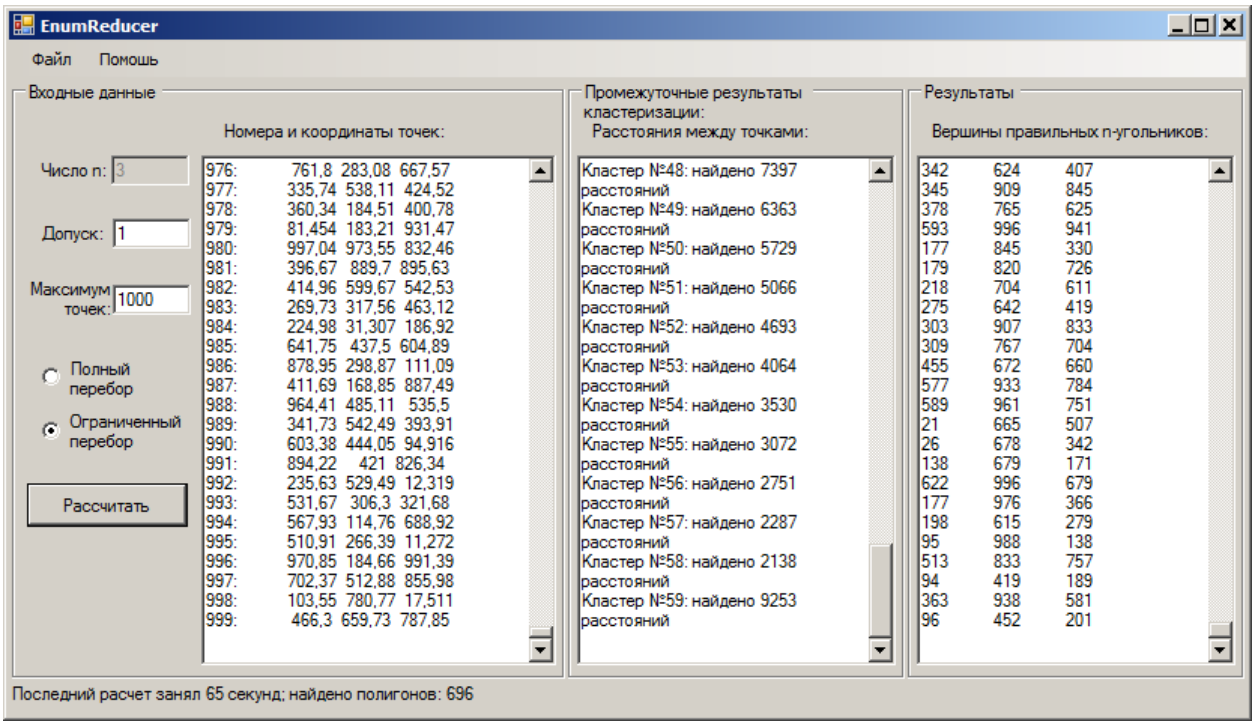

Рис. 2. Результати пошуку правильних полігонів при допуску 1 за допомогою запропонованого алгоритму при K = 60

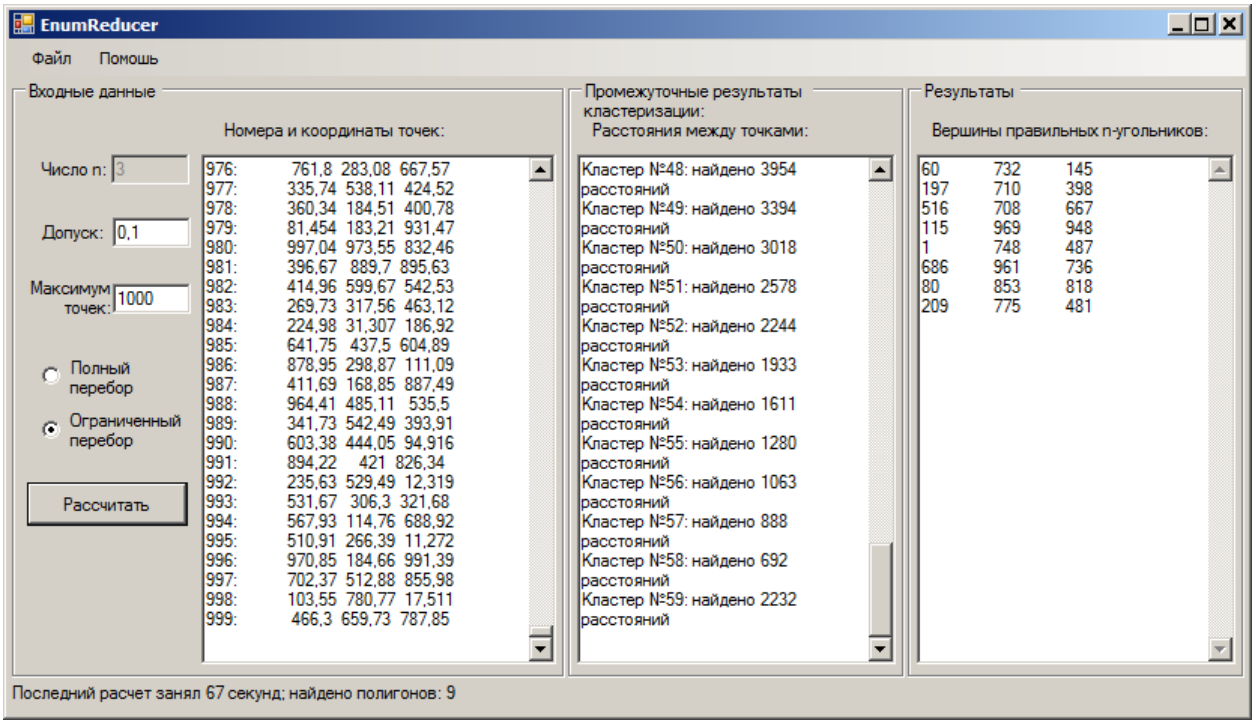

Рис. 3. Результати пошуку правильних полігонів при допуску 0,1 за допомогою запропонованого алгоритму при K = 60.

Слід зазначити, що при використанні запропонованого алгоритму виникають втрати невеликої кількості полігонів за рахунок того, що різні сторони їх потрапляють у різні кластери (а значить такі полігони не можуть бути знайдені за допомогою запропонованого алгоритму). Це рідкісна ситуація, що має низьку

ймовірність, яка зростає зі збільшенням кількості кластерів.

Проведено дослідження прискорення перебору зі збільшенням кількості кластерів K, і навіть збільшення відсотка втрачених полігонів – рис. 4 (для допуску 1) та рис. 5 (для допуску 0,1).

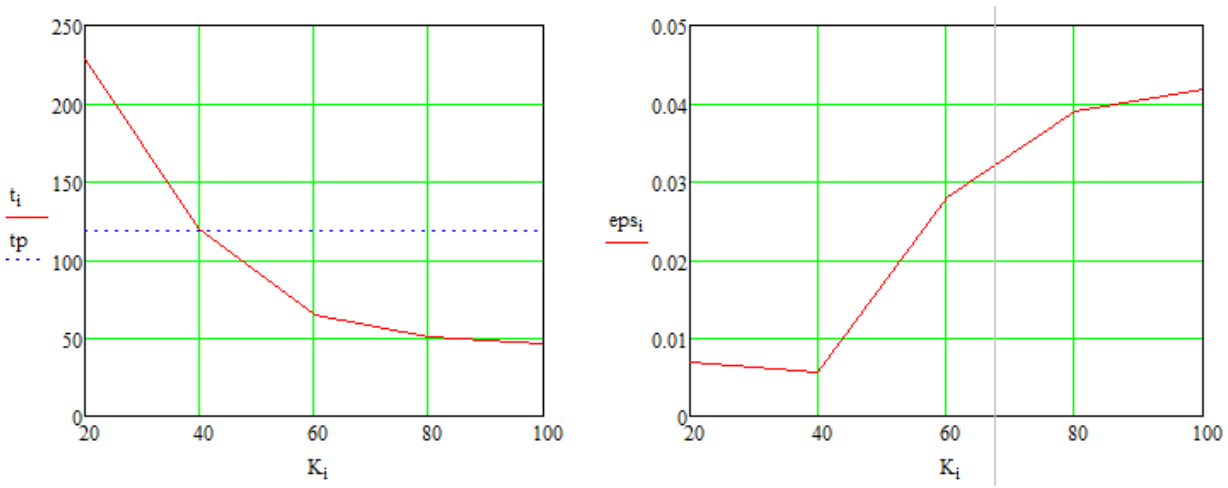

Рис. 4. Графіки зміни часу (в сек) виконання розрахунку 1000 точок при допуску 1 в залежності від числа кластерів (горизонтальним пунктиром показаний результат повного перебору) - а, а також відсоток втрачених полігонів - б.

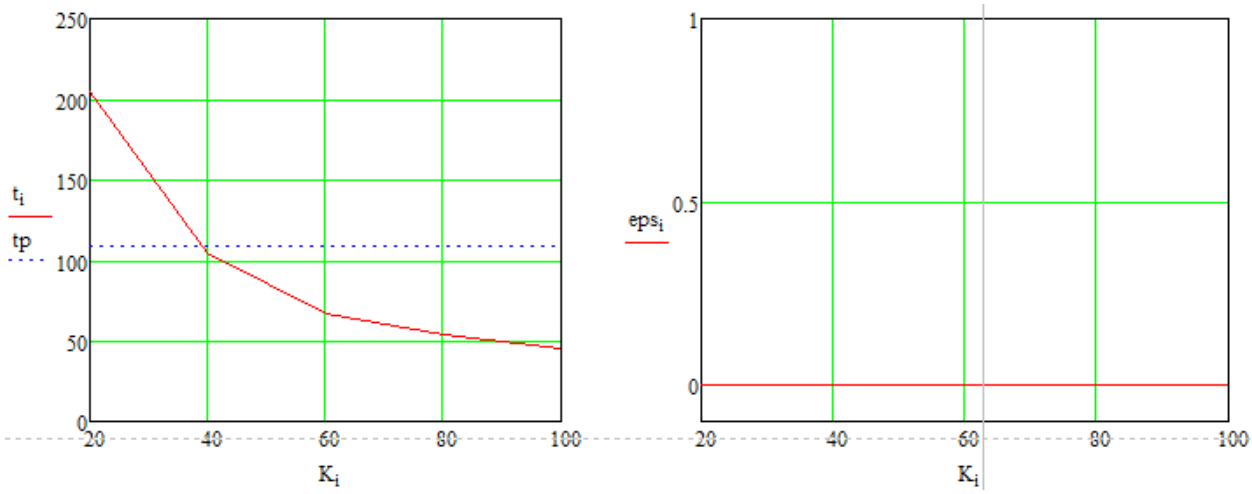

Рис. 5. Графіки зміни часу (в сек) виконання розрахунку 1000 точок при допуску 0,1 в залежності від числа кластерів (горизонтальним пунктиром показаний результат повного перебору) - а, а також відсоток втрачених полігонів - б.

Бачимо, що при малих кількостях відібраних об'єктів помилка практично не виникає через її вкрай малу ймовірність. Найбільш оптимальним є значення K в районі 60, яке дозволяє отримати майже дворазове прискорення роботи при рівні помилок менше 2%, що є цілком допустимим у завданнях комп'ютерної графіки, де втрата невеликої частини даних не погіршує якості результуючого матеріалу.

ВИСНОВКИ

Таким чином, розглянуто задачу прискорення перебору об'єктів, яка часто виникає при вирішенні прикладних завдань обробки інформації. Спочатку проведено формалізацію опису процесу перебору, а також деталізовано пов'язані з ним математичні поняття. Далі на основі аналітичного огляду існуючих методів перебору та розгляду можливостей щодо їх прискорення зроблено висновки про перспективи та необхідність власної розробки в даному напрямку. Відповідно, запропоновано удосконалений алгоритм перебору, заснований на декомпозиції умови перебору в кон'юнктивну або диз'юнктивну нормальну форму, а також з урахуванням кластеризації вхідної інформації (або кластеризації додаткової інформації, що вираховується на основі вхідної), що відноситься до однієї (або кількох) умов перебору. Ефективність запропонованого методу досліджена на прикладі поширеної задачі з галузі комп'ютерної графіки, яка полягає у пошуку правильних n-кутників (полігонів) на множині точок тривимірного простору, заданих трійками своїх координат. Попередній теоретичний розрахунок дозволив оцінити прискорення виконання завдання під час використання запропонованого алгоритму на рівні 2-3 рази, проте ці результати потрібно було перевірити ще й практично шляхом дослідження конкретної програмної реалізації.

Тестування програмного продукту показало реальне прискорення процесу перебору під час використання запропонованого методу, що становить кілька разів (у 2-3 разу при використанні до 100 кластерів). У той же час було встановлено, що за рахунок попадання різних частин одного й того ж підпадаючого під умову перебору об'єкта у різні кластери, насправді може бути втрачена невелика частка необхідних об'єктів. Відсоток втрат залежить від кількості кластерів розбиття (як і величина прискорення) і навіть для великого їх числа дає втрати не більше 2 %. Такі втрати є цілком допустимими у завданнях комп'ютерної графіки, тому загалом застосування цього методу для окремих прикладних задач є цілком обґрунтованим.

REFERENCES

1. **Coronel C., Morris S.** (2023). Database Systems: Design, Implementation, and Management, 14th ed. Boston, Cengage Learning, 784.
2. **Tamb1 V.K., Singh N.** (2024). A Comparison of SQL and NO-SQL Database Management Systems for Unstructured Data. International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering (IJAREEIE), Vol. 13, Issue 7, 2086-2093.
3. **Glake, D., Kiehn, F., Schmidt, M., Panse, F., Ritter, N. Towards** (2022). Polyglot Data Stores: Overview and Open Research Questions. URL: https://arxiv.org/abs/2204.05779.
4. **Booch G., Maksimchuk R.A., Michael W.E., Young B.J., Conallen J., Houston K.A.** (2007). Object-Oriented Analysis and Design with Applications. Boston: Addison-Wesley Professional, 3 edition, 720.
5. **Konc J., Janežič D.** (2017). An Improved Branch-and-Bound Algorithm for the Maximum Clique Problem. MATCH Communications in Mathematical and in Computer Chemistry, Vol. 78, No. 3, 569–590.
6. **Burkard R.E., Çela E., Karisch S.E., Rendl F.** (2019). A Branch-and-Bound Algorithm for the Quadratic Assignment Problem Based on the Hungarian Method. European Journal of Operational Research, Vol.112, No.3, 647–662.
7. **Gonçalves J.F., Mendes J.J.M., Resende M.G.C.** (2005). A hybrid genetic algorithm for the job shop scheduling problem. European Journal of Operational Research, Vol.167, No.1. 77–95.
8. **González M.A., Vela C.R., Varela R.** (2008). A new hybrid genetic algorithm for the job shop scheduling problem with setup times. Proceedings of the 18th International Conference

on Automated Planning and Scheduling (ICAPS 2008), 116–123.
9. **Akenine-Möller J.T., Haines E., Hoffman N., Pesce A., Iwanicki M., Hillaire S.** (2018). Real-Time Rendering. Boca Raton: CRC Press, 1198.
10. **Albahari B.A.** (2017). C# 7.0 in a Nutshell: The Definitive Reference. Sebastopol: O'Reilly Media, 1088.
11. **Brett D. McLaughlin, Pollice G., West D.** (2006). Head First Object-Oriented Analysis and Design. Sebastopol: O'Reilly Media, 636.
12. **Nagel Ch**. (2018). Professional C# 7 and .NET Core 2.0. Birmigham: Wrox, 1440.
13. **Mirjalili, S., Mirjalili, S. M., Hatamlou, A.** (2020). A Survey on Swarm Intelligence Algorithms: Principles, Applications, and Challenges IEEE Access, Vol. 8, 164848–164872.

On approaches to accelerate object search in information processing tasks

Oleksandr Haisha, Serhii Lienkov, Olena Haisha

Abstract. The paper provides a detailed examination of the concept of the enumeration process. It is performed with objects x, which are elements of a certain set X. The purpose of enumeration is to execute a certain active operation y = f(x) on the elements being processed. This operation can, of course, be applied to every element, but in the vast majority of practical tasks, the operation is applied only to those elements that satisfy a specific enumeration condition P(x).

The simplest method of enumeration is exhaustive search or "brute-force". In this approach, every single element of the set X is checked against the condition P(x), and depending on the result, the operation y = f(x) is executed. It is evident that exhaustive search serves as a benchmark process in the sense that it is convenient to evaluate the efficiency of various fast enumeration algorithms in comparison to it. Fast (or restricted) enumeration algorithms are advisable to use in most practical cases, provided that exhaustive search does not take an insignificant amount of time, such as fractions of a second. However, in some cases, it is reasonable to optimize (limit) the enumeration process even when execution times are very short, particularly if such short enumeration processes are repeated multiple times per unit of time, ultimately accumulating into a significantly large total runtime.

The branch and bound method is the most well-known algorithm for restricted enumeration, allowing the process to progressively advance from one variant to another while discarding entire sets of elements that are known in advance not to satisfy the condition P(x). However, this method does not allow for reliable preliminary estimates of the efficiency of traversing the entire tree of corresponding elements, and its applicability to a particular problem only becomes evident during the enumeration process.

This paper proposes a restriction method based on representing the enumeration condition P(x) in conjunctive or disjunctive normal form, followed by verifying this condition using a scheme of reduced logical computations that utilize only one of the conditions. Additionally, it is proposed to precompute certain supplementary information that allows obtaining the final result more quickly during enumeration. Such information may include the classification (clustering) of all objects being enumerated and performing the enumeration only within clusters, thereby reducing the total number of elements to be processed.

The proposed approach is explained in detail through an example of solving a relevant problem in the field of computer graphics, specifically the search for all regular n-angles (demonstrated using equilateral triangles) within a set of N points in three-dimensional space, where each point is defined by its coordinates. A verbal description of the exhaustive search necessary for solving this problem is provided, along with a detailed explanation of the restricted enumeration algorithm constructed according to the proposed scheme (reduced computations using conjunctive normal form and preliminary clustering of all objects being enumerated, or more precisely, clustering of the associated information – distances between all pairs of points).

The paper presents numerical calculations that allow for a theoretical assessment of the efficiency of the proposed solution. It has been established that, under the assumption of a perfectly uniform distribution of all enumerated objects across K classes, a significant reduction in the number of distance comparison operations occurs when searching for equilateral triangles. Specifically, for K = 50, the theoretical speedup reaches up to three times. This acceleration has been practically examined using a model software implementation written in C#, demonstrating that the actual speedup is quite close to the theoretical maximum, confirming that the proposed approach is indeed effective.

Keywords: enumeration, enumeration acceleration, clustering, decomposition of enumeration conditions.