УДК 004.519.712 +004.92

В.М. ТЕРЕЩЕНКО, С.Є. СКЛЯРОВСЬКИЙ

РЕГІОНАЛЬНИЙ ПОШУК ДЛЯ МНОЖИНИ РУХОМИХ ТОЧОК

Анотація. *У роботі запропоновано структури даних у вигляді B^x-дерев для побудови ефективних алгоритмів регіонального пошуку на множині рухомих точок в евклідовому просторі E^d.*
Ключові слова: *структури даних, регіональний пошук, B^x-дерево, множина рухомих точок.*

Аннотация. *В работе предложены структуры данных в виде B^x-деревьев для построения эффективных алгоритмов регионального поиска на множестве движущихся точек в эвклидовом пространстве E^d.*
Ключевые слова: *структуры данных, региональный поиск, B^x-дерево, множество движущихся точек.*

Abstract. *A modification of data structures in the form of B^x-trees for constructing effective algorithms of regional search on the set of moving points in Euclidean space E^d is proposed.*
Keywords: *data structures, regional search, B^x-tree, set of moving points.*

1. Вступ

Постановка проблеми. Однією із важливих задач обчислювальної геометрії є задача регіонального пошуку для множин рухомих точок. Вона має широке застосування в системах обробки інформації в реальному часі, системах моніторингу, системах управління базами даних, системах балансування навантажень на мережах. У роботі розглядаються ефективні підходи до представлення й обробки множини рухомих точок (об'єктів) та застосування відповідних структур даних в алгоритмах регіонального пошуку. Проте ця задача є малодослідженою, за винятком окремих випадків для фіксованих регіонів пошуку та малої розмірності простору (задачі на площині та в тривимірному просторі). Це пов'язано, перш за все, з динамічністю системи, по-друге, з слабо розробленими моделями представлення динамічних даних реального часу в обчислювальних машинах.

Аналіз останніх досліджень. На сьогоднішній день для розв'язання задачі регіонального пошуку на множині рухомих точок використовують два основних підходи, які ґрунтуються на структурах даних типу збалансованих дерев. Перший підхід, відомий як підхід без урахування часу [1, 2], представляє час як додатковий вимір простору, зберігає й працює з траєкторіями рухомих точок. Перевага даного підходу в тому, що структури даних, які він використовує, змінюються лише при зміні траєкторії точки або при додаванні чи видаленні точки. Недолік даного підходу очевидний: через використання простору більшої розмірності зростає час запиту [2]. В основі другого підходу [1, 6] лежить використання кінетичних структур даних [4] та динамічних дерев для рухомих точок. Він підтримує структуру даних у реальному часі: коли точки рухаються, структура змінюється. Не дивлячись на те, що точки рухаються неперервно, структура даних оновлюється лише в певні дискретні моменти часу, коли відбуваються деякі події, наприклад, коли координати деяких двох точок співпадають. Такий підхід дає ефективний час пошуку, проте потребує перебудови структури, навіть якщо траєкторія жодної точки не змінилась. Ще одним недоліком даного підходу є те, що він дає можливість відповідати лише на запити щодо поточної конфігурації точок. Звісно, можна використовувати перебудови для отримання результатів запитів з майбутнього, але при цьому до часу запиту додається час перебудови структури. Зазначимо, відповідно до робіт [1, 4, 6], що якщо множина S складається з n точок в $\Re^1$, то, використовуючи кінетичні B-дерева,

ISSN 1028-9763. Математичні машини і системи, 2011, № 2

можна отримати результати запиту за час $O(\log n + k)$, де k – кількість точок, що потрапляють у запитний регіон. Перебудова структури даних здійснюється для $O(n^2)$ подій, кожна перебудова потребує $O(\log n)$ часу. При зміні розмірності зростає лише час перебудови. Наприклад, для $\Re^2$ він складає $O(\log^2 n)$, але, як зазначено в роботі [6], використовуючи бінарний пошук, його можна зменшити до часу $O(\frac{\log^2 n}{\log\log n})$.

З практичної точки зору, використання *B*-дерев досить складне. Можна використовувати $k-d$ -дерева, не дивлячись на те, що їх використання залишає пошуковий час незмінним, але загальний час погіршується, так як $k-d$ -дерева потребують більше перебудов, ніж B-дерева [3, 5, 6].

Зазначимо, що в загальній постановці (без суттєвих обмежень на функції руху, пошукові регіони та розмірність простору) задача є малодослідженою. Всі відомі алгоритми дозволяють виконувати регіональний пошук лише при умові, що закони руху об'єктів лінійно залежать від часу [1]. В роботах [5, 6] розглядаються варіанти нелінійного руху, але отримані результати мають теоретичний характер, так як клас нелінійних функцій руху, що розглянуто, дуже вузький. В роботах [1, 4, 6] визначено обчислювальну складність розв'язків даної задачі в просторах розмірності більше трьох та вказано оцінки пам'яті, що дозволяють будувати ефективні пошукові алгоритми. На основі розв'язків даної геометричної задачі в роботах [4–6] розглянуто можливе використання відповідних алгоритмів обчислювальної геометрії для побудови ефективних алгоритмів та структур даних для обробки швидкоплинної інформації в дискових системах управління базами даних.

Новизна та ідея. В роботі запропоновано модифікацію існуючих підходів для випадку евклідового d -вимірного простору $(d \geq 2)$ шляхом побудови структур даних, що дозволяють ефективно використовувати дискову пам'ять, зменшуючи кількість операцій читання-запису та зберігаючи в оперативній пам'яті лише невелику кількість сторінок даних.

Мета роботи. Розробка ефективної модифікації алгоритмів регіонального пошуку для множини рухомих точок у двовимірному просторі та відповідних структур даних в евклідовому d-вимірному просторі $(d \geq 2)$.

2. Побудова розв'язку задачі регіонального пошуку для множини рухомих точок

Нехай $S = \{p_1, p_2, ..., p_n\}$ – множина з n точок у просторі $\Re^2$, кожна з яких перебуває у неперервному русі. Нехай $p_i(t)$ положення точки i в момент часу t, а $S(t) = \{p_1(t), p_2(t), ..., p_n(t)\}$ – множина точок у момент часу t. Припустимо, що кожна точка рухається з постійною швидкістю, тобто $p_i(t) = a_i + b_i t$, де $a_i, b_i \in \Re^2$, а траєкторія кожної точки є відрізком $\overline{p_i}$. Нехай задано запитний регіон R (паралелепіпед) і L – множину відрізків, що відповідають траєкторіям точок з S. Необхідно для заданого регіону R та відповідного часу запиту t_q знайти всі точки з множини S, які знаходяться всередині регіону R у час t_q, тобто знайти $S(t_q) \cap R$.

2.1. Структури даних

Множину рухомих точок будемо представляти у вигляді B^x-дерева, яке, у свою чергу, є розвиненням ідеї пошукового B-дерева. З точки зору зовнішнього логічного представлення, B^x-дерево – збалансоване, дуже гіллясте, може зберігатись у зовнішній пам'яті. Збалан-

сованість означає, що довжина шляху від кореня дерева до будь-якого з листів однакова. Гіллястість дерева – властивість кожного вузла дерева посилатись на велику кількість вузлів-нащадків. З точки зору фізичної організації, B^x-дерево представляється як мультисписка структура сторінок зовнішньої пам'яті, тобто кожному вузлу дерева відповідає блок зовнішньої пам'яті (сторінка). Внутрішні та листові сторінки мають різну структуру.

Застосування модифікованих B-дерев зумовлено логарифмічними оцінками основних операцій та пристосованістю B-дерев для обробки великої кількості даних, об'єм яких може бути настільки великим, що унеможливлює зберігання цих даних в оперативній пам'яті.

Покажемо, що ми маємо справу саме з великими об'ємами даних. Будь-яку задачу над множиною рухомих точок (динамічну задачу) можна подати як набір статичних задач. Для цього достатньо спроектувати нашу задачу на вісь часу й розглядати кожну проекцію окремо. У випадку регіонального пошуку часова оцінка не погіршиться, так як для заданого пошукового регіону та часу буде розглядатися не більше, ніж 3 проекції, а задача на проекції є звичайною пошуково-регіональною задачею, для якої відомі оцінки складності та алгоритми, що їх забезпечують. Але оцінка пам'яті у випадку використання проекцій дуже зростає і стає рівною $O\left(\frac{T}{\Delta t}\cdot M\left(N(t_0)\right)\right)$, де $\frac{T}{\Delta t}$ – кількість проекцій, а $M\left(N(t_0)\right)$ – оцінка необхідної пам'яті для статичної задачі. Отже, виникає ідея застосувати B^x-дерева для представлення відповідної великої кількості даних більш ефективно, але залишивши логарифмічними часові оцінки складності. Опишемо більш детально структуру B^x-дерева.

Означення. B^x-деревом будемо називати дерево з одним коренем:

1. Кожна вершина x містить поля, в яких зберігаються:

а) кількість $n[x]$ ключів, що містяться у вершині;

б) ключі $key_1[x] \le key_2[x] \le ... \le key_{n[x]}[x]$ (у неспадному порядку).

2. Якщо x – внутрішня вершина, то вона містить $n[x]+1$ вказівників $c_1[x], c_2[x], ..., c_{n[x]+1}[x]$ на її нащадків. У листків діти відсутні, тому відповідні поля для них не визначені.

3. Ключі $key_i[x]$ служать границями, що розділяють значення ключів у піддеревах:

$$k_1 \le key_1[x] \le k_1 \le key_2[x] \le \ldots \le key_{n[x]}[x] \le k_{n[x]+1},$$

де k_i – довільний з ключів, що зберігається в піддереві з коренем $c_i[x]$.

4. Усі листки знаходяться на однаковій глибині, яка дорівнює висоті дерева.

5. Кількість ключів, що містяться в одній вершині, обмежена зверху та знизу; границі задаються єдиним для всього дерева числом $m \ge 2$, яке називається мінімальним ступенем вершини B^x-дерева. А саме:

а) кожна вершина, за виключенням кореня, містить, принаймні, $m-1$ ключів. Таким чином, внутрішні вершини (крім кореня) мають не менше, ніж m дітей. Якщо дерево не є порожнім, то в корені повинен зберігатись, принаймні, один ключ;

б) у вершині зберігається не більше, ніж $2m-1$ ключів. Відповідно, внутрішня вершина має не більше, ніж $2m$ дітей. Вершину, в якій зберігається рівно $2m-1$ ключів, будемо називати повною.

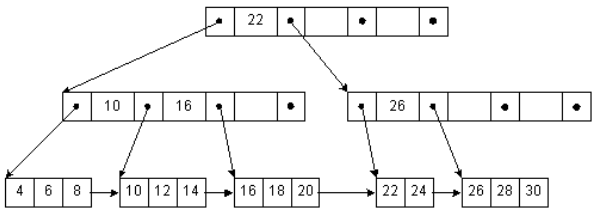

Рис. 1. Приклад B^x-дерева

На рис. 1 показано приклад *B^x-дерева* для випадку $m=3$. Усі ключі зберігаються в листках, там же зберігається інформаційна частина вузла. У внутрішніх вузлах зберігаються лише копії ключів, які

необхідні для ефективного пошуку потрібного листка. Причому лівий вказівник веде до ключів, які менші заданого, правий – до ключів, які не менші за заданий. Варто звернути увагу на те, що, наприклад, ключ 22 повторюється в листку, де зберігаються відповідні йому дані. Під час вставки або видалення необхідно коректно модифікувати батьківські вузли. Коли ключ, що модифікується, перший у листі, здійснюється прохід по дереву від листа до кореня. Останній із вказівників, що не менший за даний, знайдений при спуску по дереву, є той, який необхідно модифікувати, щоб відобразити нове значення ключа. Оскільки усі ключі повторюються в листках, можна прошити (зв'язати) їх для послідовного доступу.

2.2. Основні операції на структурі

Операції на B^x-деревах аналогічні операціям на B-деревах, які, у свою чергу, є просто розвиненням операцій на бінарних деревах, з урахуванням більшої кількості дітей та специфіки вказівників. Розглянемо їх більш детально.

Розбиття вершини B^x-дерева на дві. Додавання елемента в B^x-дерево – більш складна операція у порівнянні з бінарними деревами. Ключовим моментом є розділення повної вершини y з $2m-1$ ключами на дві з $m-1$ ключами в кожній. При цьому ключ-медіана $key_m[y]$ відправляється до батька x вершини y та стає роздільником отриманих двох вершин. Це можливо, якщо вершина x – неповна. Якщо y – корінь, процедура виконується аналогічно. В цьому випадку висота дерева збільшується на одиницю.

Додавання вершини B^x-дерева. Додавання елемента в B^x-дерево проходить у два етапи. Спочатку здійснюється пошук вершини, до якої додається задана. Далі, якщо ця вершина неповна, виконується додавання, в іншому випадку вершина розчіплюється на дві й виконується додавання до відповідної щойно одержаної вершини. Процедура видалення аналогічна відповідній процедурі для бінарних дерев з урахуванням більшої кількості дітей кожної вершини.

Представлення рухомих точок у B^x-дереві. Так як B^x-дерево представляє кінетичну (динамічну) структуру даних, то рухомі точки представляються в B^x-дереві як листки з їх поточними координатами. Всі проміжні вершини представляють собою прошиті інтервали, кожний з яких містить впорядкований список точок по іншій координаті (у тривимірному випадку – список списків) аналогічно структурі дерева регіонів. Кожна проміжна вершина також містить упорядковані списки подій, які характеризують часові точки перетину траєкторій рухомих точок та впорядковані списки подій, які характеризують перехід рухомої точки в сусідній інтервал. Зауважимо, що кожна поточна вершина може містити не більше, ніж n^2+n подій.

Процедура впорядкування B^x-дерева. Так як корінь B^x-дерева містить упорядкований список з усіх подій, причому цей список не містить подій, які вже відбулись, то для визначення гілок дерева, які необхідно перевпорядкувати, переглянемо цей список, починаючи з першої мітки, до мітки, яка більша за поточний час. Кожна поточна подія вимагає впорядкування відповідної їй гілки дерева. Для поточної події, що відповідає перетину траєкторій точок, виконується перестановка відповідних точок місцями у впорядкованих списках (так як подія полягає в зміні порядку точок). Для події, що характеризує перехід точки до сусіднього інтервалу, необхідно виконати перенаправлення вказівників на дану точку з поточної гілки до гілки, що характеризує інтервал, в який потрапляє точка, видалення точки зі списків старої гілки та додавання до списків нової гілки.

Пошуковий регіональний запит до B^x-дерева. Пошуковий регіональний запит повертає всі рухомі точки, що попадають у пошуковий регіон q, який є паралелепіпедом, у запитний час t_q. Для виконання цього запиту достатньо виконати звичайний регіональний

запит до B^x-дерева, яке в даному контексті можна вважати деревом регіонів. Зазначимо, що перед виконанням пошукового запиту потрібно виконати процедуру впорядкування дерева.

2.3. Побудова розв'язку задачі регіонального пошуку для множини рухомих точок

Алгоритм розв'язку задачі регіонального пошуку складається з двох основних моментів: з попередньої обробки рухомих точок та основного алгоритму.

2.3.1. Попередня обробка рухомих точок

На етапі попередньої обробки виконується побудова для заданих рухомих точок B^x-дерева, визначаються списки подій для перебудови дерева й прошиваються відповідні інтервали. Опишемо більш конструктивно етап попередньої обробки:

1. Для заданих рухомих точок будуємо впорядковані по відповідній координаті списки точок.

2. На основі одного впорядкованого списку будуємо дерево, яке відповідає дереву регіонів, але інтервали ділимо не бінарно, а на m частин, до кожної вершини пришиваємо впорядковані списки вершин за іншими координатами, які отримуються як елементарна вибірка зі списків, отриманих на першому кроці.

3. Знаходимо впорядкований за часом список подій типу перетин інтервалу для кожної точки та кожного інтервалу дерева, а саме: для кожної точки знаходимо час, в який вона вийде зі свого інтервалу й перейде у сусідній. Прошиваємо відповідні події до відповідного інтервалу (інтервалу, з якого точка буде переходити). Під подією розуміємо трійку: час події, рухома точка та напрямок переходу (в лівий чи правий інтервал). Зазначимо, що для задачі спостереження в реальному часі додаються події виходу точок за межі поля зору.

4. Знаходимо впорядкований за часом список подій типу перетин траєкторій для кожної пари точок. А саме, визначаємо з закону руху точок (нагадаємо, що закони лінійні відносно часу) точку перетину траєкторій, розв'язуючи систему лінійних рівнянь. Зазначимо, що система має вигляд:

$$\begin{cases} a_1 t + x_1 = a_2 t + x_2 \\ b_1 t + y_1 = b_2 t + y_2 \end{cases},$$

де невідомою змінною є лише t. Для кожної такої події визначаємо інтервал, в якому вона відбудеться, та зв'язуємо її з вершиною дерева відповідного інтервалу.

Після виконання зазначених вище 4 кроків отримуємо пошукове дерево в початковий момент часу. Опишемо попередню обробку на прикладі двовимірного простору та двох рухомих точок. Нехай задано дві рухомі точки: $A = ((0,0),(1,1))$ та $B = ((10,-1),(-2,2))$, тобто точки в початковий момент часу знаходяться за координатами $(0,0)$ та $(10,-1)$, рухаються у напрямку векторів $(1,1)$ та $(-2,2)$, причому перша рухається зі швидкістю $\sqrt{2}$, а друга – зі швидкістю 2. Тобто закони руху точок (t,t) та $(-2t+10,2t-1)$, а траєкторії рухомих точок – відповідні промені прямих $y = x$ та $y = \frac{6-x}{4}$. Впорядкуємо точки за значенням абсциси у список $U_x = \{A,B\}$, за ординатами – у список $U_y = \{B,A\}$. Так як дерево-основа є дерево регіонів для інтервалу $[0,10]$ з двома точками, то воно матиме вигляд (рис. 2) (так як точок лише 2, то немає сенсу розглядати дерево з $m > 2$).

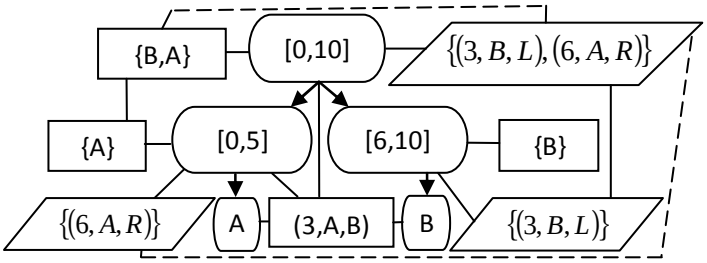

Рис. 2. Дерево-основа

Відповідно до даного списку припишемо першу подію до інтервалу $[6,10]$, а другу подію – до інтервалу $[0,5]$, загальний список запишемо в корінь дерева.

Знайдемо впорядкований за часом список подій типу перетин траєкторій рухомих точок. В даному випадку він складається з однієї події (тому що точок 2), для визначення якої прирівняємо закони руху і знайдемо з системи з 2 рівнянь та 1 змінної час t =11/4=3 (зазначимо, розв'язок знаходимо з одного рівняння, а друге слугує для перевірки, якщо знайдений розв'язок не задовольняє друге рівняння, траєкторії точок не перетинаються). Так як час у нас квантується одиницею (дискретний запитний час), то точки поміняють свій порядок лише в можливий запитний момент $t = 3$. Визначаємо інтервал, в якому точки змінять порядок, це інтервал $[0,5]$. Зв'язуємо подію з цим інтервалом та посилання до цієї події на точки A та B. Зв'язуємо також подію з усіма вершинами верхнього рівня (в нашому випадку кореня). Одержимо бажане нам дерево пошуку.

2.3.2. Основний цикл алгоритму

Основний цикл алгоритму складається з послідовного виконання операцій впорядкування дерева та виконання запиту:

1. *Впорядкування дерева* (згідно з розд. 2.4). Для впорядкування дерева спочатку переглядаємо список подій типу перетин траєкторій, який знаходиться у корені дерева. Так як список впорядкований за часом подій, то достатньо переглянути події, час яких менший за поточний. Нехай поточний час $t = 5$, і в списку є одна подія для заданого часу. Нехай вона має вигляд $(5, A_1, A_2)$. Тоді видаляємо цю подію зі списку та переставляємо точки A_1 та A_2 місцями у списках впорядкування за координатами, що зв'язані з інтервалом, в якому відбувається подія і з яким, у свою чергу, зв'язана подія $(5, A_1, A_2)$. Далі переглядаємо список подій типу перетин інтервалу, який зв'язаний з коренем дерева. Нехай для заданого часу є одна подія типу перетин інтервалу $(5, A_1, L)$, зв'язана з відповідним інтервалом I, який рухома точка A_1 перетне з лівого кінця. Для обробки цієї події видаляємо її зі списку подій, видаляємо точку A_1 зі списків впорядкування за координатою в інтервалі I та додаємо до списків впорядкування за координатою в інтервал I_L, який є найближчим лівим сусідом інтервалу I і знаходиться на одному рівні з ним. Переносимо зв'язок точки A_1 з інтервалу I в інтервал I_L (додаємо до списку вершин, що містяться в ньому). Після обробки цих двох списків дерево відповідає дереву пошуку в поточний момент часу, що дозволяє обробляти пошуковий запит.

2. *Пошуковий запит* (згідно з розд. 2.5). Нехай задано пошуковий регіон $q = (x_1, y_1, x_2, y_2)$. Здійснюємо спуск по дереву до вершин, інтервали яких перетинаються з інтервалом $[x_1, x_2]$, для кожного такого інтервалу найнижчого рівня (не рахуючи рівень листів-точок) перевіряємо, відповідно до списку впорядкованих за координатами точок, належність точок запитному регіону та формуємо відповідь. Тобто, з точки зору пошуку, дерево, що розглядається, є звичайним деревом регіонів.

3. Обґрунтування складності

Теорема 1. Часова оцінка складності розв'язку задачі регіонального пошуку для множини з n рухомих точок у просторі $\Re^d$ становить $O(\log^d n)$ операцій, за умови, що $O(n^d \log^d n)$ операцій необхідно для попередньої побудови кінетичної структури даних (у початковий момент часу) та в процесі роботи для $O(n^2)$ подій відбувається впорядкування структури, яке складає $O(\log^d n)$ операцій.

Доведення. Часова оцінка пошуку є тривіальною, так як являє собою оцінку пошуку статичної задачі регіонального пошуку для пошукового регіону типу паралелепіпед, яка виконується методом регіонального дерева. Оцінка часу попередньої обробки побудови кінетичної структури даних у початковий момент часу теж тривіальна (побудова *B*-дерева, доведення часової оцінки побудови приведені в роботах [3, 4]). Так як впорядкування структури відбувається лише в моменти перетину траєкторій рухомих точок, зрозуміло, що впорядкування потрібно виконати $O(n^2)$ в найгіршому випадку, при умові попарного перетину всіх траєкторій. Остання оцінка слідує з властивостей побудованого *B*-дерева. Так як кожна перебудова представляє перечеплення місцями всіх дітей для гілки дерева, листками якої є точки, траєкторії яких перетнулись у поточний момент часу, з урахуванням збалансованості дерева, прохід по гілці дерева потребує часу, пропорційного висоті дерева, і, з урахуванням константного часу операції, перечеплення складає $O(\log^d n)$.

Теорема 2. Мінімальна часова оцінка складності розв'язку задачі регіонального пошуку на множині з n рухомих точок у просторі $\Re^d$ дорівнює $\Omega(\log^d n)$.

Доведення. Для зведення задачі регіонального пошуку на множині рухомих точок (динамічної задачі регіонального пошуку – ДЗРП) до задачі регіонального пошуку на статичній множині точок (статичної задачі регіонального пошуку – СЗРП) розглянемо проекції ДЗРП на вісь часу. Тоді пошуковий запит ДЗРП представляє звичайний регіональний пошуковий запит типу паралелепіпед на деякій часовій проекції ДЗРП. Відповідно до результатів [1], мінімальна часова оцінка складності розв'язку СЗРП у просторі $\Re^d$ для множини з n точок та пошукового регіону типу паралелепіпеда складає $\Omega(\log^d n)$. Зрозуміло, що перетворення звідності лінійне, так як кожна часова проекція представляє собою СЗРП, тому для отримання результату достатньо знайти відповідну проекцію серед упорядкованої множини. Очевидно, що такий пошук можна виконати навіть за логарифмічний час (використовуючи, наприклад, бінарний пошук).

4. Практична частина

Алгоритм реалізовано лише для двовимірного та тривимірного простору, які представляють на даний момент часу надзвичайний інтерес з практичної точки зору (задачі моніторингу та балансування навантажень. Зазначимо, що існуючі підходи здебільшого розглядають лише двовимірний випадок. Нижче наведено графіки залежності часу роботи від розмірності задачі для двовимірного та тривимірного випадків (рис. 3, 4). Випадки розмірності більше трьох потребують великих обсягів пам'яті та в контексті комп'ютерної геометрії не є показовими (не є наочними).

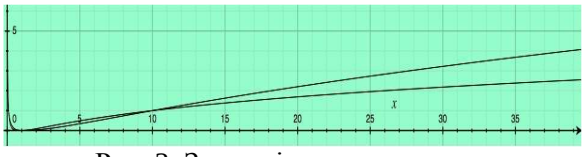

Рис. 3. Залежність часу пошуку від розмірності задачі

Звичайно алгоритм та структуру даних реалізовано з урахуванням можливості поширення на простори розмірності більше трьох.

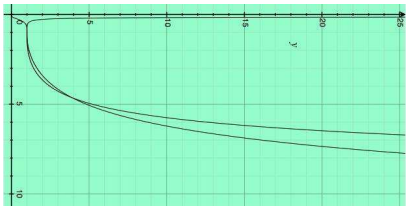

Рис. 4. Залежність часу попередньої обробки від розмірності задачі

B-дерева представляють можливість ефективно будувати індекси, які використовують зовнішню пам’ять, але в наведеній реалізації структури даних не використовують зовнішню пам’ять в початковому сенсі (є припущення, що кількість точок дозволяє зберігати все дерево в оперативній пам’яті). У випадку багатовимірних просторів в оперативній пам’яті зберігається кожне дерево зі списками подій та посиланнями на область зовнішньої пам’яті, яка містить структуру дерева кожного вузла. Подібна реалізація описана в [4, 6]. Зазначимо, що через відсутність готових реалізацій інших підходів практичне порівняння результатів і продуктивності не проводилось.

5. Висновки

У роботі запропоновано метод розв’язання задачі регіонального пошуку для множини рухомих точок, заснований на поєднанні статичних регіональних дерев та *B*-дерев, що дало змогу ефективно представляти динамічні дані про рухомі точки й ефективно виконувати регіональний пошук. На відміну від попередніх робіт розглянуто та приведено реалізацію відповідного підходу для тривимірного простору, що дозволяє поширити відповідні алгоритми та структури даних на простори більших розмірностей. У роботі показано проблемні місця в існуючих алгоритмах регіонального пошуку на динамічних даних та запропоновано метод часткового вирішення проблем, заснований на використанні оптимізуючих структур даних типу *B*-дерев, що дозволяють побудову ефективних індексів і використання зовнішньої пам’яті.

Методом звідності встановлено нижню оцінку складності алгоритмів розв’язання поставленої задачі та наведено часові оцінки складності запропонованого підходу й оцінки пам’яті, необхідної для відповідних структур даних. Реалізація підходу, що розглядається в роботі, демонструє прикладну цінність ефективних алгоритмів розв’язання задачі, а саме використання в біологічних дослідженнях для спостережень за мікроорганізмами. У цей час у більшості передових країн світу інформаційно-психологічні засоби здійснення психологічних операцій розглядаються як пріоритетні при досягненні поставлених цілей. При цьому дані засоби можуть бути використані як для підтримки психічного здоров'я людини, так і для впливу на свідомість і психіку як своїх військ і населення, так і військ і населення потенційного супротивника.

СПИСОК ЛІТЕРАТУРИ

1. Agarwal P.K. Indexing moving points / P.K. Agarwal, L. Arge, J. Erickson // J. Comput. Syst. Sci. – 2003. – Vol. 66. – P. 207 – 243.
2. Agarwal P.K. Kinetic medians and kd-trees / P.K. Agarwal, J. Gao, L.J. Guibas // Proc. 10th European Sympos. Algorithms. – London, 2002. – P. 15 – 26.
3. Guibas L.J. Kinetic data structures – a state of the art report / L.J. Guibas; P.K. Agarwal, L.E. Kavraki, M. Mason (ed.) // Proc. Workshop Algorithmic Found. Robot. – Wellesley, 1998. – P. 191 – 209.
4. Procopiuc C.M. Star-tree: An efficient selfadjusting index for moving points / C.M. Procopiuc, P.K. Agarwal, S. Har-Peled // Proc. 4th Workshop Algorithm Eng. Experiments. – San Francisco, 2002. – P. 178 – 193.
5. Indexing the positions of continuously moving objects / S. Saltenis, C.S. Jensen, S.T. Leutenegger [et al.] // Proc. ACM SIGMOD Internat. Conf. Management Data. – Vancouver, 2008. – P. 331 – 342.

Стаття надійшла до редакції 05.04.2011