КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра обчислювальної математики

Кваліфікаційна робота

на здобуття ступеня магістра
за спеціальністю 113 Прикладна математика
на тему:

ОПТИМІЗАЦІЯ АЛГОРИТМІВ ПОРІВНЯННЯ ЛАНЦЮГІВ ДНК У ГЕНОМНИХ ЕКСПЕРИМЕНТАХ

Виконав студент 2-го курсу
Майстренко Олександр Сергійович _____

Науковий керівник:
професор, доктор фіз.-мат. наук
Клюшин Дмитро Анатолійович _____

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент Майстренко О. С. _____

Роботу розглянуто й допущено до захисту на засіданні кафедри обчислювальної математики «5» травня 2023 р.,

протокол № 7

Завідувач кафедри
С. І. Ляшко _____

Київ – 2023

ЗМІСТ

ВСТУП

За останнє десятиліття відбувся технологічний зсув у галузі біоінформатики: більші за розміром геноми тепер можуть бути секвенсовані швидко та ефективно з точки зору витрат, що призводить до потреби в обчисленнях для ефективного порівняння великих та різноманітних послідовностей.

Крім того, виявлення збережених подібностей у великих колекціях геномів залишається проблемою. Розмір хромосом разом із значною кількістю шумів і кількість повторів, знайдених у послідовностях ДНК (зокрема, у ссавців і рослин), призводить до сценарію де виконання та очікування повних результатів вимагає як часу, так і ресурсів.

Етапи фільтрації, перевірка та анотація вручну, дуже довгий час виконання та високий попит на обчислювальні ресурси представляють лише деякі з багатьох труднощів, з якими стикаються під час порівняння великих геномів. У цій роботі розглянуто методи, призначені для порівняння значної кількості дуже довгих ДНК послідовностей та способи їх оптимізації.

РОЗДІЛ 1

КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Сучасні проблеми геномних експериментів

Завдяки технологічному прогресу за останні кілька десятиліть секвенування цілих геномів стає дешевшим з експоненціальною швидкістю. У результаті зростаюча база цілих секвенованих геномів стає загальнодоступною, розміром від менш ніж двохсот тисяч базових пар до більш ніж 22 $*10^9$ базових пар.

Відокремлені послідовності геномів збільшуються не лише в кількості, але й у ширині: великі еукаріотичні геноми, такі як ссавців і рослин секвенуються. Ці нещодавно включені послідовності довершують наше розуміння еволюції організмів, відповіді на такі питання, як «Який організм виник першим?» і «Які еволюційні події спричинили дивергенцію видів?».

Однак, щоб покращити наше розуміння цих тем, нові дані включаються в бази даних, але це має свою ціну: чим більше геномів, тим більше геномів погіршує сценарій, коли немислимо проводити геномні експерименти без допомоги обширних обчислювальних ресурсів, але, крім того, навіть поточні обчислювальні методи залишаються позаду зростаючої складності.

Наразі більша частина роботи з геномною інформацією (наприклад, ідентифікаційного кодування регіонів або генерування попарних синтетичних карт) виконується вручну та курується платформами, присвяченими для цієї мети, які часто включають сховище попередньо обчислених порівнянь геномів. Наприклад, NARCISSE, Genomicus і SynFind надають такі дані разом з іншими інструментами для цілей дослідження, такими як геном-браузери та аналізатори каріотипу.

Хоча ці ресурси містять високоякісну підібрану інформацію, вони зазвичай покладаються на попередньо обчислені дані та зазвичай не дозволяють користувачам запускати нові експерименти на вимогу. Як в результаті, коли

нові експерименти підтримуються цими платформами, зазвичай використовують обмежене порівняння методології, такі як підходи на основі генів (див. CoGe), щоб зменшити обчислювальні вимоги. Проте ці підходи часто базуються на BLAST алгоритмі, який спочатку не був розроблений для великого попарного геному порівняння.

Локальне виконання великих порівнянь геномів часто вимагатиме тривалого часу роботи та, швидше за все, буде здійснювати брак обчислювальних ресурсів. Такі інструменти, як GECKO, CGALN і MUMMER, були створені для великомасштабних порівнянь геномних послідовностей.

Однак обчислення порівнянь парних хромосом зазвичай займає від кількох хвилин до годин залежно від довжини послідовностей, кількості повторень та інших факторів. Час, що минув, збільшується квадратично, коли порівнюються всі хромосоми двох видів (тобто порівняння $n*m$).

Крім того, згадані раніше алгоритми витрачають більшу частину свого часу обчислення на обробку з певним ступенем вичерпності, створюючи такі результати, як повне вирівнювання або колекції високоякісних пар сегментів (HSP).

Однак у випадку порівняння всіх проти всіх порівнянь буде понад 400 хромосом (наприклад, 20*20 хромосом). Отримані порівняння для великої частини набору даних часто будуть складається з повторів і шуму, з низьким ступенем збереження синтенії. Таким чином, здається природним включити попередній, евристичний крок, який аналізує, чи представляє інтерес конкретне порівняння, і зменшує обчислення складність виконання вичерпного порівняння.

1.2 Постановка задачі

Розглянемо скінченний алфавіт S = {A, C, G, T}. Слово w визначається як будь-який кінцевий рядок літер в алфавіті S, тобто w генерується регулярним

виразом [ACGT]+. Набір усіх слів w над S буде позначатися U (універсум). Визначимо послідовність s як слово розміру n, тобто |s| = n. Набір слів w розміру k, що міститься в s, позначимо $C = \{w_1, w_2, \dots, w_m\}$, де m=n−k+1, що враховує всі можливі перекриття слова фіксованого розміру k у послідовності s.

Позначимо сукупність подібних сегментів між двома послідовностями s_a та s_b як $H_{a,b} \subseteq C_a \times C_b$, що враховує спільні слова (w_i, w_j) C_a і C_b (також відомі як «хіти» - «hits»). Розподіл множини $H_{a,b}$ сильно залежить від двох послідовностей s_a і s_b, які створюють його, оскільки більш схожі послідовності створять більше подібностей, тоді як віддалені послідовності створять менше подібностей. Також сильно вплине розмір слів на множину $H_{a,b}$.

Далі перевизначимо $H_{a,b}$. як склад трьох інших розподілів, а саме:

- $W_{a,b}$ – це розподіл випадкових збігів між s_a та s_b. Враховуючи достатньо великі n_a та n_b (довжини двох послідовностей, відповідно), ймовірність пошуку випадкових збігів відповідає біноміальному розподілу (припускаючи однаковість літер у послідовностях) $B\left(|C_a \times C_b|, \frac{1}{4}^k\right)$. Отже, $W_{a,b}$ — це розподіл чистого випадкового шуму (тому виникає випадково).
- $R_{a,b}$ — це розподіл будь-якого співпадіння, що його слова w_i, w_j неодноразово розкидані (тобто з частотою більше одиниці) в обох послідовностях s_a та s_b та/або їх складність (вимірюється як найдовший унікальний підрядок, що міститься в слові w) вважається низьким на основі біологічних стандартів. Крім того, з грубої точки зору, цей розподіл враховує будь-який повторюваний елемент, такий як тандемні повтори, області низької складності та вкраплені повтори.
- $T_{a,b}$ — це розподіл сигналів, що зберігаються між s_a і s_b, таким чином, що $T_{a,b} = H_{a,b} \backslash (W_{a,b} \cup R_{a,b})$. Тобто співпадіння, чиї слова w_i,w_j є унікальними (тобто їх частота дорівнює одиниці), є спільними щодо

спільного предка (збереженого), не виникли випадково і не можуть вважатися повторами під час біологічного дослідження.

1.3 Непряма індексація k-мерів

Ідеальна хеш-функція індексування *f* для універсуму U_k (тобто всіх існуючих слів w розміру k над S) задана функцією:

$$f(w_i^k) = |S|^0 * v(w_{i,1}^k) + |S|^1 * v(w_{i,2}^k) + \cdots + |S|^{j-1} * v(w_{i,j}^k) + \cdots + |S|^{k-1} * v(w_{i,k}^k)$$

$$f(w_i^k) = \sum_{j=1}^{k} |S|^{j-1} * v(w_{i,j}^k)$$

де $w_{i,j}^k$ позначає j-ту букву слова w_i, де $|w_i| = k$ і функція v(x) повертає цілочисельну позицію x у S, починаючи з нуля (тобто v(A)=0 і v(T)=3). Зауважте, що функція f є біекцією: $f: U^k \to N^{\{1..|S|^k\}}$, що створює ідеальне індексування без колізій для U^k; таким чином, кількість зображень у кодомені дорівнює кількості зображень у домені, що становить 4^k. Крім того, визначимо сімейство функцій F як набір функцій $F = \cup_{z=1}^{\infty} f(x)^z$, де $f(w_{i,j}^k)^z = \sum_{j=1}^{k}(|S|^{j-1} * v(w_{i,j}^k) * belongs(j,z))$, операція *belongs*(j, z) повертає 1, якщо існує таке натуральне число x, що j = z * x (тобто j є кратним z), інакше 0.

Отже, набір функцій F містить усі функції індексування $f(x)^z$, які використовують або всі літери, або половину літер, або третину букв і так далі (з перервами), щоб обчислити хеш w. Тобто, чим вище значення z функції $f(x)^z$, тим більшою буде кількість зіткнень. В ідеалі кожна пропущена літера збільшить кількість колізій на S = 4, оскільки $w_{i,j}^k$ може приймати до 4 різних значень у позиції j. Використання будь-якої функції $f(x)^z$ з z>1 вироблятиме неточне індексування, уможливлюючи виявлення довгих, ідеальних збігів, які також включають позиційну інформацію в k-мерах. На відміну від використання менших k-мерів, неточні k-мери не настільки схильні до генерації випадкового

шуму під час порівняння. Отже, будь-яка функція індексування, взята з сімейства F (з z>1), уможливить евристичну підвибірку всіх можливих неточних збігів k-мерів.

Визначення того, чи є k-мери подібними, але неточними в одній позиції, вимагає 3^k перевірок у прямому переборі. Складність виконання цього експоненціально зростає з довжиною k і кількістю дозволених неточностей.

Таким чином, за допомогою сімейства функцій F пошук неточних k-мерів виконується за час O(1) евристично.

1.4 Евристична фільтрація

Враховуючи достатньо велике k та функцію індексування f, збіги, створені розподілом $W_{a,b}$ (тобто випадкові збіги), можна ігнорувати, оскільки ймовірність пошуку випадкових збігів між наборами слів C і C′ довжини k і набору розміри m та m′ (відповідно) дорівнює $1 - (\left(1 - \frac{1}{4^k}\right)^m)^{m'}$, що становить p=0,0005419 для випадку хромосом ссавців до 100 мільйонів пар основ і k=32.

Щоб видалити шум, створений $R_{a,b}$, зберігаються лише унікальні співпадіння в $H_{a,b}$. За визначенням, повторення – це будь-яке слово w_i, частота якого перевищує одиницю. Отже, видалення всіх співпадінь (w_i, w_j) із $H_{a,b}$, де w_i з'являється більше одного разу, ефективно видалить усі можливі повтори, що належать до розподілу $R_{a,b}$.

Однак точні слова, як правило, зустрічаються рідше, оскільки k стає більшим (через мільйони років еволюції), і маленьке k може створювати випадковий шум. Щоб подолати цей сценарій, використовується гібридний підхід: велике k = 32 з евристичною функцією індексування (z>1, як пояснювалося раніше), що дозволяє знаходити довгі унікальні слова з деяким ступенем варіативності.

Тим не менш, цього підходу недостатньо для визначення збережених подібностей, оскільки навіть віддалені споріднені організми a і b матимуть значну

кількість малих збережених сигналів (наприклад, основні функціонуючі гени).

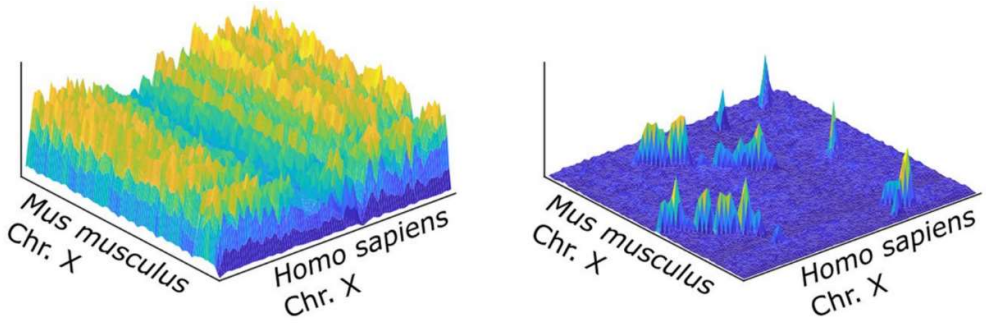
Щоб виділити збережені подібності, необхідно згрупувати (кластеризувати) збіги, які належать до довгих спільних частин геному (тобто до HSP), а не лише до збережених генів. Для цього застосовується зменшення дискретизації, тобто позиція співпадінь сильно зменшується, щоб змусити співпадіння, що належать до HSP, опинитися в тій самій позиції, отже, збільшуючи концентрацію звернень у регіонах, де можна знайти синтенії.

Цей підхід схожий на двовимірну позиційну кластеризацію хешів, ідентифікованих під час етапів зіставлення (наприклад, MOSAIK24, яка використовує позиційну кластеризацію хешів для розгляду регіонів-кандидатів на вирівнювання).

Розглянемо HSP як підмножину набору слів C_a і C_b, тобто для деяких початкових позицій i, j і для заданої довжини слова w_i і w_j у C_a і C_b, відповідно, рівні (зверніть увагу що не всі слова мають бути однаковими, тобто можуть виникнути невідповідності). Зрозуміло, що лінійне зменшення роздільної здатності відобразить співпадіння поблизу позиції i разом у нову позицію i′, якщо використовується рівномірна функція сюр’єктивного відображення, оскільки кластери будуть формуватися з сусідніх попадань, які належать до збережених сигналів.

Такий самий результат спостерігається для j. Отже, більша кількість звернень буде зіставлено з позиціями i′ і j′, якщо HSP існує, що дозволяє виявити його як збережений сигнал.

На малюнку 1 показано приклад порівняння процедури точного зіставлення слів і описаного евристичного методу. Зауважимо, що в точній процедурі збережені HSP перевищують кількість повторів і, таким чином, залишаються невиявленими. Однак в евристичній процедурі зберігаються лише збережені HSP, а повтори видаляються.



Мал. 1. Тривимірні гістограми звернень, зіставлених за допомогою точної процедури (ліворуч) або запропонованого евристичного методу (праворуч).

Такий самий результат спостерігається для j. Отже, більша кількість звернень буде зіставлено з позиціями i′ і j′, якщо HSP існує, що дозволяє виявити його як збережений

1.5 HIFI-алгоритми

Клітини - це складне динамічне середовище, яке потребує постійної регуляції їхніх генів для забезпечення виживання. Поява технологій захоплення конформації хромосоми (3C) і останні досягнення в методах візуалізації призвели до кращого розуміння організації геному та його ролі в регуляції генів. Hi-C, високопродуктивна похідна 3C, надає неперевершене уявлення про тривимірну (3D) організацію геному, фіксуючи всі контакти ДНК-ДНК, виявлені в популяції клітин.

Hi-C виявив різні рівні організації геному, включаючи топологічно асоційовані домени (TADs, subTADs) і компартменти хроматину. Тим не менш, потенціал для більш детального розуміння тривимірної організації геному залишається в основному невикористаним.

Алгоритми HIFI спрямовані на надійну оцінку контактних частот Hi-C між усіма внутрішньохромосомними парами рестрикційних фрагментів. Результатом алгоритму HIFI є матриця IF на хромосому, де кожен запис (i, j) відповідає IF для RF i та j.

Оскільки REs не перетравлюють ДНК рівномірно вздовж геному, різні рядки/стовпці відповідають ділянкам різного розміру. Залежно від використовуваного RE, досяжна роздільна здатність Hi-C коливається в середньому від 434 bp (для чотирьох різців, таких як MboI) до 3,7 kb (для шести різців, таких як HindIII).

Аналіз даних Hi-C з високою роздільною здатністю стикається з багатьма проблемами, з яких найбільш важливою є розрідженість спостережуваних даних пари читання. Наприклад, експеримент Hi-C з дуже високою глибиною секвенування в один мільярд пар зчитування дасть у середньому приблизно 0,1 пари зчитування на запис внутрішньохромосомної матриці для RE з шістьма різцями та менше 0,001 для RE з чотирма різцями. Навіть на відносно короткій відстані 100 кб спостережувана кількість пар зчитування для RE з чотирма різцями майже ніколи не перевищує 1.

Ця розрідженість призводить до спостережувана кількість пар зчитування для даної пари РЧ є поганою (з високою дисперсією) оцінкою справжньої ПЧ, за винятком рідкісних пар РЧ, розташованих у областях контактної карти Hi-C, де значення ПЧ надзвичайно високі. Усі існуючі рішення цієї проблеми використовують переваги того факту, що IF сусідніх записів у матриці IF сильно корельовані.

Зокрема, найпоширенішим підходом до компромісу роздільна здатність/точність є штучне зниження роздільної здатності шляхом групування необроблених даних до інтервалів фіксованого розміру (наприклад, 25-кілобайтні контейнери). Така нижча роздільна здатність збільшує кількість зчитувань на пару бінів і, таким чином, дозволяє отримати більш надійну оцінку IF, але ціною втрати біологічної інтерпретації.

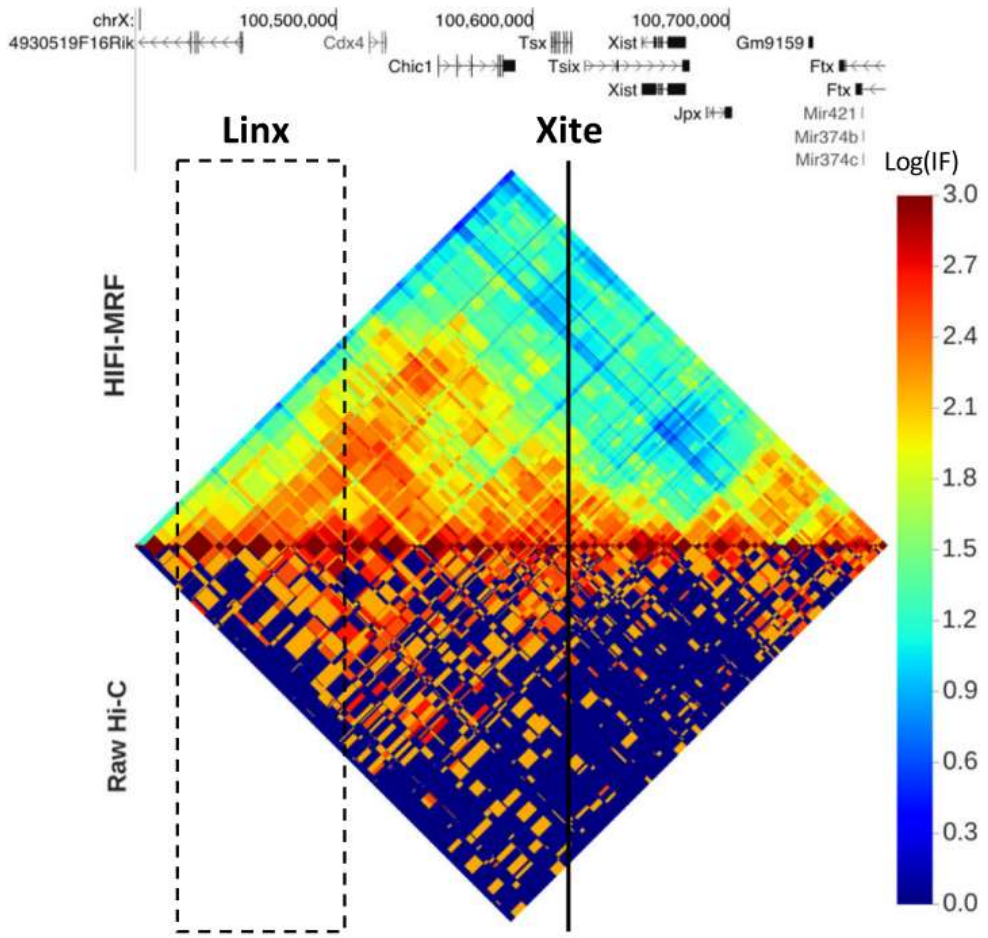
Важливо, що жоден унікальний розмір бункера не є однаково ідеальним для всієї матриці IF. Частини матриці ПЧ, де присутні високі ПЧ, можуть підтримувати аналіз високої роздільної здатності, тоді як інші, що відповідають нижчим значенням ПЧ, можуть вимагати більших відсіків для точної оцінки ПЧ.

Більш конкретно, проблема, яка розглядається, полягає в наступному: розглянемо набір даних Hi-C H, отриманий із заданим ферментом рестрикції e. Для заданої хромосоми необроблений результат зберігається у внутрішньохромосомній матриці n × n RC, де n — це кількість RF, створених e, а $RC_{i,j}$ містить кількість зчитованих пар, зіставлених із парою RF (i, j).

Мета полягає в тому, щоб якомога точніше оцінити справжню частотну матрицю взаємодії на РЧ-рівні, IF_{true}, яка є теоретичною матрицею IF n × n, яку можна було б отримати, якщо послідовність нескінченно великої версії H до нескінченної глибини (масштабована для загальна кількість прочитаних пар).

На IF_{true} впливає низка помилок у бібліотеці, секвенуванні та відображенні, які потрібно було б виправити, щоб забезпечити правильну біологічну інтерпретацію; для цього завдання вже існує багато таких методів нормалізації. Мета полягає не в удосконаленні цих методів, а в тому, щоб працювати над потоком і надати найточнішу оцінку IF_{true}.

Висока точність і роздільна здатність, які забезпечує HIFI, дозволяють дослідникам відповідати на запитання, які важко вирішити за допомогою аналізу даних Hi-C із низькою роздільною здатністю. Далі буде проілюстровано одну з таких програм: аналіз високої роздільної здатності меж TAD і subTAD. Було використано модифікований показник індексу спрямованості (DI), спочатку представлений Dixon et al., щоб ідентифікувати 5000 меж TAD у даних HindIII-GM12878 Hi-C. Граничні прогнози виконувалися з двома роздільними здатностями: (i) роздільна здатність радіочастот з використанням даних, оброблених HIFI-MRF (в середньому 3,7 кб) та (ii) підхід класичного фіксованого групування (16 радіочастот ≈ 50 кбіт на бін).



Мал. 2. Дрібномасштабні нормативні контакти в даних Hi-C виявлені HIFI-MRF.

Використовуючи набори даних ENCODE ChIPseq, кількісно визначили зайнятість білків, що зв'язують ДНК, відносно меж TAD. Відповідно до повідомлених раніше спостережень і моделей, CTCF продемонстрував значне збагачення безпосередньо за межами цих кордонів, причому сайти на плюсовій нитці різко досягали максимуму на верхніх границях TAD, а ті на мінус-ланцюзі – на піку, на кордонах нижче за течією.

Подібні збагачення на межах TAD спостерігаються для RAD21, SMC3 (когезиновий комплекс), YY1 і ZNF143, що відповідає попереднім звітам. Хоча те саме явище спостерігається в даних фіксованого групування, піки набагато

гостріші (вужчі та вищі) у даних HIFI-MRF, що вказує на те, що РЧ-роздільна здатність дозволяє точніше визначити межі TAD.

РОЗДІЛ 2

МЕТОДИ ОПТИМІЗАЦІЇ ГЕНОМНИХ ЕКСПЕРИМЕНТІВ

2.1 Використання хеш-функції та k-мерів

Розглянемо такі підходи до оптимізації порівняльних геномних експериментів як використання хеш-функції та індексація k-мерів.

У порівняльних геномних експериментах хеш-функції використовуються для швидкого та ефективного порівняння послідовностей ДНК або РНК.

Хеш-функція - це функція, яка приймає на вхід послідовність символів будь-якої довжини та повертає фіксований вихідний код фіксованої довжини. Хеш-функції використовуються в порівняльних геномних експериментах для перетворення послідовностей нуклеотидів у хеш-коди, які можуть бути порівняні швидко та ефективно.

Один з найбільш відомих алгоритмів хешування в порівняльних геномних експериментах - це алгоритм MinHash. Він використовує хеш-функції для перетворення послідовностей нуклеотидів у хеш-коди фіксованої довжини. Потім за допомогою алгоритму MinHash, ці хеш-коди порівнюються між собою та знаходять спільні хеш-коди. Чим більше спільних хеш-кодів мають дві послідовності, тим більша їх подібність.

Інші алгоритми порівняння геномних послідовностей, такі як BLAST (Basic Local Alignment Search Tool), використовують різні методи хеш-функцій та порівняння послідовностей, щоб знайти подібність між двома геномними послідовностями.

K-мери є короткими послідовностями довжиною k, що складаються з нуклеотидів (A, C, G, T для ДНК або A, C, G, U для РНК), які можуть бути взяті з геномної послідовності. В геномних експериментах, k-мери використовуються для аналізу геномних даних, зокрема для ідентифікації відмінностей між

геномними послідовностями, класифікації організмів та вивчення функцій генів.

Одним з основних застосувань k-мерів є знаходження геномних варіантів (SNP, deletions, insertions тощо). За допомогою алгоритмів, які шукають відмінності між k-мерами у двох різних геномних послідовностях, можна виявити мутації, які виникли в одному з геномів. Це може бути корисно при дослідженні генетичних хвороб та еволюційних процесів.

Крім того, k-мери можуть бути використані для класифікації організмів. Наприклад, вони можуть бути використані для ідентифікації видів бактерій або вірусів. Кожен вид має свої характеристичні k-мери, які можна використовувати для їх ідентифікації та класифікації.

Ще k-мери можуть бути використані для вивчення функцій генів. За допомогою алгоритмів, які шукають повторювані k-мери в геномній послідовності, можна виявити місця зв'язку факторів транскрипції або місця зі зв'язку рибосом з мРНК, що вказує на початок трансляції гена.

2.2 Порівняння програмних імплементацій

Для порівняння ефективності та виміру необхідного часу для роботи алгоритмів порівняння ДНК ланцюгів з використанням хеш-функції та індексації k-мерів було змодельовано наступні експерименти.

Для експерименту було використано базу даних ДНК ланцюгів приматів.

Загальна кількість пар – 20871.

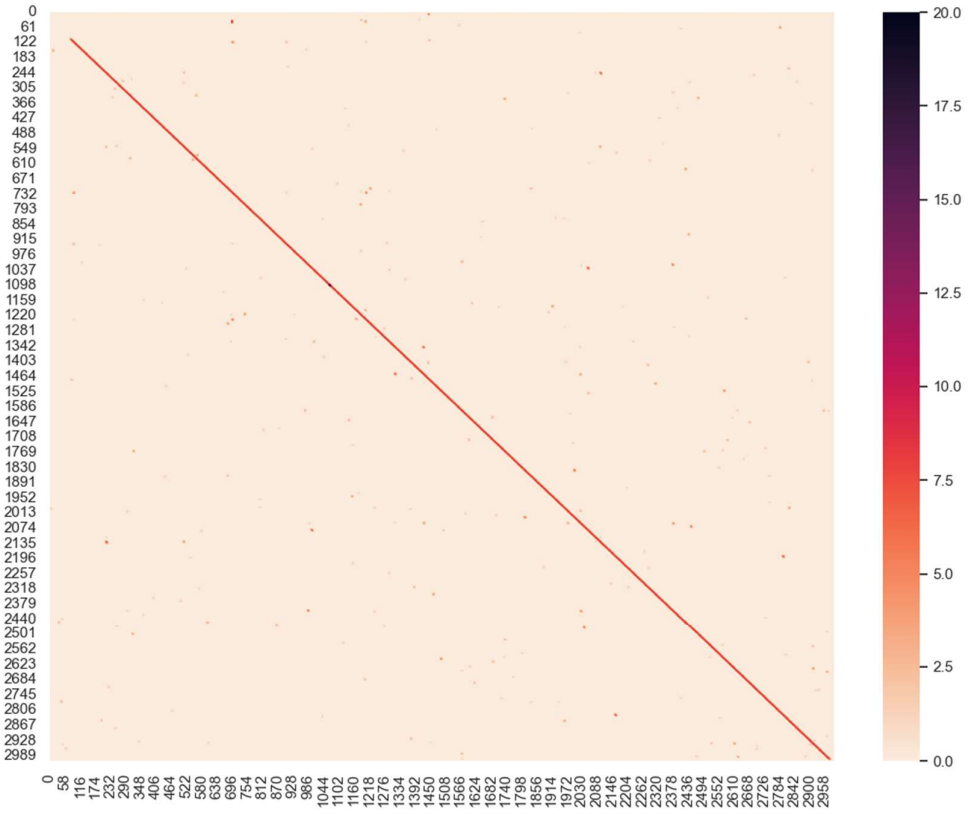
Для наочності до експерименту також був доданий аглгоритм порівняння без використання методів оптимізації для демонстрації ефективності даних підходів.

N	Назва	Метод	Час виконання
1	initial	–	4 год 1 хв 49 с

2	hash	хеш-функція	14 хв 56 с
3	kmers	індексація k-мерів	13 хв 25 с

Табл. 1. Результати порівняльного експерименту

Як можна побачити, експерименти з використанням оптимізаційних методів зайняли значно менше часу.



Мал. 3. Матриця співпадінь порівняльного геномного експерименту

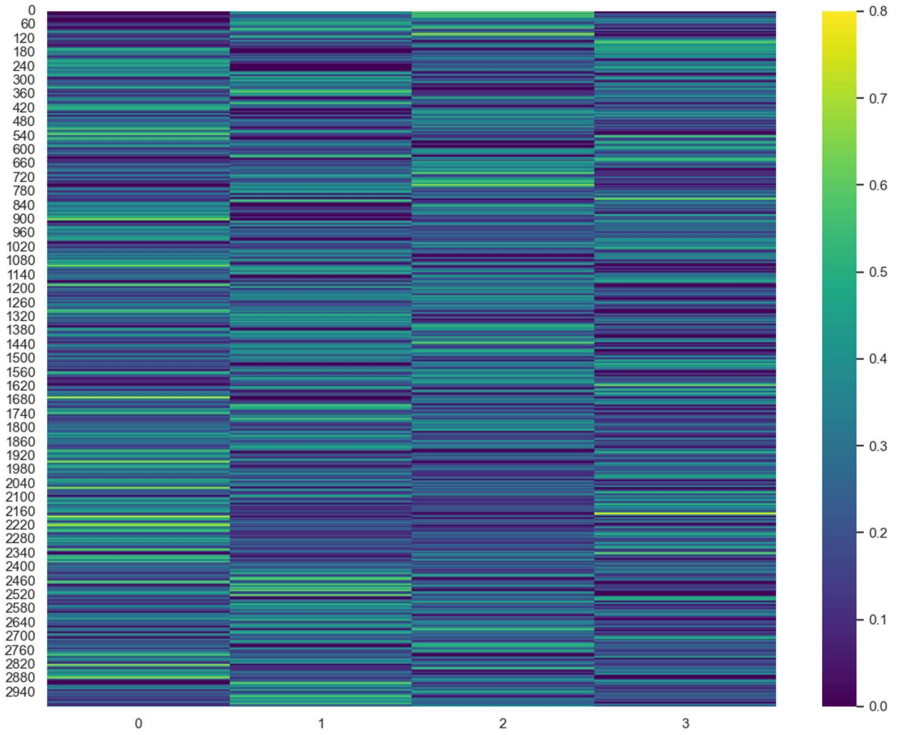
Додатково було створено функцію для генерації матриць розподілу випадкових величин у порівняльному експерименті.

Вхідні дані – пари ДНК ланцюгів розмірністю (2960,1)

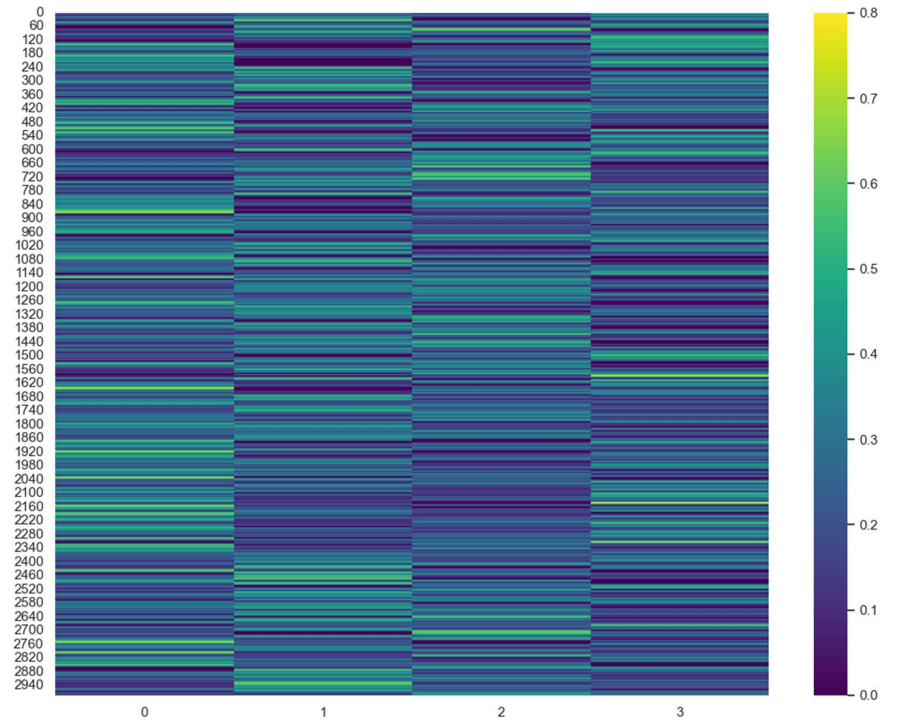
Код приймає дві послідовності ДНК (seq1 і seq2), а також розмір вікна

(window_size) і порівнює послідовності за допомогою hit-based підходу. Він обчислює матриці розподілу випадкових величин для кожної послідовності та повертає їх як матрицю1 і матрицю2. Матриці нормалізуються таким чином, що сума кожного рядка дорівнює 1, представляючи розподіл ймовірностей основ у кожній позиції.

Кожен зі стовпчиків 0, 1, 2, 3 відповідає за частоти кожної з літер алфавіту {A,C,T,G} на n-тому індексі ДНК послідовності.



Мал. 4.а Матриця розподілу випадкових величин для seq1



Мал. 4.б Матриця розподілу випадкових величин для seq2

2.3 Двовибірковий критерій Колмогорова–Смирнова

Критерій Колмогорова–Смірнова також можна використовувати для перевірки того, чи відрізняються два базові одновимірні розподіли ймовірностей. У цьому випадку статистика Колмогорова–Смирнова є

$$D_{n,m} = \sup_x \left|F_{1,n}(x) - F_{2,m}(x)\right|,$$

де $F_{1,n}$ та $F_{2,m}$ – емпіричні функції розподілу першої та другої вибірок відповідно.

Для великих вибірок нульова гіпотеза відхиляється на рівні α якщо

$$D_{n,m} > c(\alpha)\sqrt{\frac{n+m}{nm}},$$

де n та m – розміри першої та другої вибірки відповідно.

Зауважимо, що тест двох вибірок перевіряє, чи дві вибірки даних походять з одного розподілу. Це не визначає, що це за загальний розподіл (наприклад, нормальний він чи ні). Недоліком одновимірного тесту Колмогорова–Смірнова є те, що він не надто потужний, оскільки він розроблений таким чином, щоб бути чутливим до всіх можливих типів відмінностей між двома функціями розподілу. Деякі стверджують, що тест Кукконі, спочатку запропонований для одночасного порівняння розташування та масштабу, може бути набагато потужнішим, ніж тест Колмогорова–Смирнова, коли порівнюють дві функції розподілу.

Обчислимо двовибірковий критерій Колмогорова – Смирнова для матриць розподілу p попереднього порівняння.

Нульова гіпотеза полягає в тому, що два розподіли ідентичні, F(x)=G(x) для всіх x. Для відповідних колонок у кожній матриці розподілів обчислимо статистику критерію Колмогорова – Смірнова та відповідне p-значення.

	KS-stat	p-value
0	**0.8643131344248086**	**0.015357461768257986**
1	**0.9999999220087451**	**0.006495511922290098**
2	**0.9568014436111185**	**0.01304437943588263**
3	**0.9795367830574211**	**0.012022345813806436**

Табл. 2. Результати обчислення двовибіркового критерія Колмогорова – Смирнова

Можемо побачити, що для усіх чотирьох порівнянь статистики критерію мають високі значення, а p-значення менше або рівні за 0.05. Отже, можна відхилити нуль-гіпотезу і прийняти альтернативну: розподіли є різними.

Не дивлячись на те, що ДНК ланцюги належать тваринам одного ряду ссавців, мутації, що сталися, розрізнили послідовності так, що розподіли нуклеооснов відрізняються. Це також демонструє актуальність порівняльних геномних експериментів та необхідність у пошуку нових методів їх оптимізації.

ВИСНОВОК

Оптимізація у порівняльних геномних експериментах має важливе значення для забезпечення точності та достовірності результатів порівняння геномів.

Порівняльна геноміка є важливим інструментом в молекулярній біології та генетиці, який дозволяє порівняти геноми двох або більше організмів, щоб встановити їх подібність та відмінності. Оптимізація методів порівняння геномів дозволяє збільшити швидкість та точність порівняння геномів та знайти нові залежності та зв'язки між геномами.

У роботі було висвітлено як використовуються оптимізаційні методи алгоритмів порівняння ланцюгів ДНК у геномних експериментах, наведено приклади їх застосування та розроблено їх програмні імплементації (див. у додатки А, Б, В). Додатково було створено функцію для генерації матриць розподілу випадкових величин для порівняльного експерименту та обчислення двовибіркового критерію Колмогорова – Смірнова для таких матриць (див. у додатку Г).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Goodwin, S., McPherson, J. D. & McCombie, W. R. Coming of age: ten years of next-generation sequencing technologies. Nature
Reviews Genetics 17(6), 333 (2016).
2. Riley, A., Dohyup, K. & Hansen, A. K. Genome sequence of “Candidatus Carsonella ruddii” strain BC, a nutritional endosymbiont
of Bactericera cockerelli. Genome announcements 5(17), e00236–17 (2017).
3. Neale, D. B. et al. Decoding the massive genome of loblolly pine using haploid DNA and novel assembly strategies. Genome biology
15(3), R59 (2014).
4. Courcelle, E. et al. Narcisse: a mirror view of conserved syntenies. Nucleic acids research 36, D485–D490 (2007).
5. Louis, A., Mufato, M. & Crollius, H. R. Genomicus: fve genome browsers for comparative genomics in eukaryota. Nucleic acids
research 41(D1), D700–D705 (2012).
6. Tang, H. et al. SynFind: compiling syntenic regions across any set of genomes on demand. Genome biology and evolution 7(12),
3286–3298 (2015).
7. Mirzaghaderi, G. & Marzangi, K. IdeoKar: an ideogram constructing and karyotype analyzing sofware. Caryologia 68(1), 31–35
(2015).
8. Lyons, E. & Freeling, M. How to usefully compare homologous plant genes and chromosomes as DNA sequences. Te Plant Journal
53(4), 661–673 (2008).
9. Altschul, S. F. et al. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic acids research
25(17), 3389–3402 (1997).
10. Torreno, O. & Trelles, O. Breaking the computational barriers of pairwise genome comparison. BMC bioinformatics 16(1), 250
(2015).

11. Nakato, R. & Gotoh, O. Cgaln: fast and space-efcient whole-genome alignment. BMC bioinformatics 11(1), 224 (2010).

12. Delcher, A. L., Salzberg, S. L. & Phillippy, A. M. Using MUMmer to identify similar regions in large sequence sets. Current protocols
in bioinformatics 1, 10–3 (2003).

13. Nicolas, J., Peterlongo, P. & Tempel, S. Finding and characterizing repeats in plant genomes. Plant Bioinformatics 1374, 293–337
(2016).

14. https://www.nature.com/articles/s41598-019-46773-w

15. https://genomebiology.biomedcentral.com/articles/10.1186/s13059-019-1913-y

Додаток А (*лістинг програми з порівняльним експериментом*):

```
import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import time

# Get data

chimp_dataset = pd.read_table('/input/dna-sequence-
dataset/chimpanzee.txt').to_dict()

sequences = chimp_dataset['sequence'].tolist()

chimp_data = {'seq1': [], 'seq2': []}

for i in range(len(sequences)):

    for j in range(i+1, len(sequences)):

            chimp_data['seq1'] = sequences[i]

            chimp_data['seq2'] = sequences[j]

def compare_sequences(seq1, seq2, window_size, threshold):

    """

    Compare two DNA sequences for hits.


    Parameters:

    - seq1 (str): the first DNA sequence

    - seq2 (str): the second DNA sequence

    - window_size (int): the size of the window to use when comparing sequences

    - threshold (float): the minimum similarity score required for a hit
```

```
    Returns:
    - list of tuples: a list of tuples representing hits, where each tuple contains the start and end index of the hit
    """

    hits = []
    seq1_len = len(seq1)
    seq2_len = len(seq2)

    # Iterate over seq1 using a sliding window of size window_size
    for i in range(seq1_len - window_size + 1):
        window1 = seq1[i:i+window_size]

        # Iterate over seq2 using a sliding window of size window_size
        for j in range(seq2_len - window_size + 1):
            window2 = seq2[j:j+window_size]

            # Calculate the similarity score of the two windows
            score = sum(1 for x, y in zip(window1, window2) if x == y) / window_size

            # If the score is above the threshold, record the hit
            if score >= threshold:
                hits.append((i, i+window_size-1, j, j+window_size-1))
```

```
    return hits
window_size = 10
threshold = 0.9

print("INITIAL")
start_time = time.time()
for seq1 in chimp_data['seq1']:
    for seq2 in chimp_data['seq2']:
        hits = compare_sequences(seq1, seq2, window_size, threshold)
end_time = time.time()
print(end_time - start_time)

print("HASH")
start_time = time.time()
for seq1 in chimp_data['seq1']:
    for seq2 in chimp_data['seq2']:
        hits = compare_sequences_hash(seq1, seq2, window_size, threshold)
end_time = time.time()
print(end_time - start_time)

print("KMERS")
start_time = time.time()
for seq1 in chimp_data['seq1']:
```

```
    for seq2 in chimp_data['seq2']:
        hits = compare_sequences_kmers(seq1, seq2, window_size, threshold)
end_time = time.time()
print(end_time - start_time)
```

Додаток Б (*лістинг програми з хеш-функцією*):

```
def compare_sequences_hash(seq1, seq2, window_size, threshold):
    """
    Compare two DNA sequences for hits using a hash table.

    Parameters:
    - seq1 (str): the first DNA sequence
    - seq2 (str): the second DNA sequence
    - window_size (int): the size of the window to use when comparing sequences
    - threshold (float): the minimum similarity score required for a hit

    Returns:
    - list of tuples: a list of tuples representing hits, where each tuple contains the start and end index of the hit
    """

    hits = []
    seq1_len = len(seq1)
    seq2_len = len(seq2)
```

```
# Create a hash table for seq2
seq2_table = {}
for i in range(seq2_len - window_size + 1):
    window = seq2[i:i+window_size]
    hash_value = hash(window)
    if hash_value not in seq2_table:
        seq2_table[hash_value] = [(i, window)]
    else:
        seq2_table[hash_value].append((i, window))

# Iterate over seq1 using a sliding window of size window_size
for i in range(seq1_len - window_size + 1):
    window1 = seq1[i:i+window_size]
    hash_value = hash(window1)

    # Check if the hash value exists in seq2_table
    if hash_value in seq2_table:
        for j, window2 in seq2_table[hash_value]:
```

```
            # Calculate the similarity score of the two windows
            score = sum(1 for x, y in zip(window1, window2) if x == y) /
window_size

            # If the score is above the threshold, record the hit
            if score >= threshold:
                hits.append((i, i+window_size-1, j, j+window_size-1))

    return hits
```

Додаток В (*лістинг програми з k-мерами*):

```
def index_kmers(sequence, k):
    """
    This function indexes k-mers in a DNA sequence.

    Args:
    sequence (str): the DNA sequence to index.
    k (int): the length of the k-mers to index.

    Returns:
    dict: a dictionary where the keys are the k-mers and the values are the indices where the k-mer occurs.
    """
    kmers = {}
    for i in range(len(sequence) - k + 1):
        kmer = sequence[i:i+k]
        if kmer in kmers:
            kmers[kmer].append(i)
        else:
```

```
            kmers[kmer] = [i]
    return kmers



def compare_sequences_kmers(seq1, seq2, window_size, threshold):
    """
    Compare two DNA sequences for hits.

    Parameters:
    - seq1 (str): the first DNA sequence
    - seq2 (str): the second DNA sequence
    - window_size (int): the size of the window to use when comparing sequences
    - threshold (float): the minimum similarity score required for a hit

    Returns:
    - list of tuples: a list of tuples representing hits, where each tuple contains the start and end index of the hit
    """
    hits = []
```

```
seq1_kmers = index_kmers(seq1, window_size)

seq2_kmers = index_kmers(seq2, window_size)



intersect_kmers = set(seq1_kmers.keys()).intersection(set(seq2_kmers.keys()))

for k in intersect_kmers:

    for i in seq1_kmers[k]:

        for j in seq2_kmers[k]:

            hits.append((i, i+window_size-1, j, j+window_size-1))

return hits
```

Додаток Г (*лістинг програми з матрицями розподілів випадкових величин*):

```
def get_distribution_matrices(seq1, seq2, window_size):
    """
    Compares two DNA sequences using a hit-based approach with a given window size.
    Returns the distribution matrices of random variables for each sequence.
    """
    # Initialize variables
    n = len(seq1)
    m = len(seq2)
    matrix1 = np.zeros((n - window_size + 1, 4))
    matrix2 = np.zeros((m - window_size + 1, 4))
    bases = {'A': 0, 'C': 1, 'G': 2, 'T': 3}

    # Calculate hit counts for seq1
    for i in range(n - window_size + 1):
        window = seq1[i:i+window_size]
        for j in range(window_size):
            matrix1[i, bases[window[j]]] += 1
```

```
    # Calculate hit counts for seq2
    for i in range(m - window_size + 1):
        window = seq2[i:i+window_size]
        for j in range(window_size):
            matrix2[i, bases[window[j]]] += 1

    # Normalize matrices
    matrix1 = matrix1 / np.sum(matrix1, axis=1, keepdims=True)
    matrix2 = matrix2 / np.sum(matrix2, axis=1, keepdims=True)

    return matrix1, matrix2

def perform_ks_test_matrices(matrix1, matrix2):
    """
    Perform the two-sample KS test for each column of two matrices.
```

```
    Args:
        matrix1: First matrix.
        matrix2: Second matrix.

    Returns:
        A list of tuples, where each tuple contains the KS statistic and p-value for a
column.
    """
    ks_results = []

    for col1, col2 in zip(matrix1.T, matrix2.T):
        statistic, p_value = stats.ks_2samp(col1, col2)
        ks_results.append((statistic, p_value))

return ks_results
```