УДК 575

**Кисляк Сергій Володимирович**
*старший викладач кафедри біомедичної кібернетики*
*Національний технічний університет України*
*«Київський політехнічний інститут імені Ігоря Сікорського»*
**Кисляк Сергей Владимирович**
*старший преподаватель кафедры биомедицинской кибернетики*
*Национальный технический университет Украины*
*«Киевский политехнический институт имени Игоря Сикорского»*
**Kyslyak Sergii**
*Senior Lecturer of the Department of Biomedical Cybernetics*
*National Technical University of Ukraine*
*«Igor Sikorsky Kyiv Polytechnic Institute»*

**Шалаєва Ольга Сергіївна**
*студентка кафедри біомедичної кібернетики*
*Національного технічного університету України*
*«Київський політехнічний інститут імені Ігоря Сікорського»*
**Шалаева Ольга Сергеевна**
*студентка кафедры биомедицинской кибернетики*
*Национального технического университета Украины*
*«Киевский политехнический институт имени Игоря Сикорского»*
**Shalaeva Olga**
*Student of the Department of Biomedical Cybernetics of the*
*National Technical University of Ukraine*
*«Igor Sikorsky Kyiv Polytechnic Institute»*

# ОПТИМІЗАЦІЯ АЛГОРИТМУ ГЛОБАЛЬНОГО ВИРІВНЮВАННЯ З ВИКОРИСТАННЯМ АФІННОЇ ШТРАФНОЇ ФУНКЦІЇ

# ОПТИМИЗАЦИЯ АЛГОРИТМА ГЛОБАЛЬНОГО ВЫРАВНИВАНИЯ С ИСПОЛЬЗОВАНИЕМ АФФИННОЙ ШТРАФНОЙ ФУНКЦИИ

# OPTIMIZATION OF THE GLOBAL ALIGNMENT ALGORITHM USING THE AFFINE PENALTY FUNCTION

***Анотація.*** *В роботі проаналізовано проблему алгоритму вирівнювання з використанням афінного штрафу, досліджено та порівняно популярні методи оптимізації даного алгоритму та описано новий алгоритм з квадратичною часовою складністю.*

***Ключові слова:*** *алгоритм Нідлмана-Вунша, вирівнювання біологічних послідовностей, глобальне вирівнювання, афінний штраф, динамічне програмування.*

***Аннотация.*** *В работе проанализирована проблема алгоритма выравнивания с использованием аффинного штрафа, исследованы и сравнены популярные методы оптимизации данного алгоритма и описан новый алгоритм с квадратичной временной сложностью.*

***Ключевые слова:*** *алгоритм Нидлмана-Вунша, выравнивание биологических последовательностей, глобальное выравнивание, аффинный штраф, динамическое программирование.*

***Summary.*** *The work analyzes the problem of the alignment algorithm using the affine penalty, popular methods of optimization of this algorithm have been investigated and compared and a new algorithm with quadratic time complexity has been described.*

***Key words:*** *Nidlman-Wunsh algorithm, biological sequencing alignment, global alignment, affine penalty, dynamic programming.*

**Вступ.** Сучасний розвиток методів секвенування біологічних послідовностей та вдосконалення алгоритмів асемблювання геномів зумовлює експоненціальне зростання кількості нуклеотидних та амінокислотних послідовностей, що зберігаються в базах даних (таких як GenBank та UniProt). Накопичення біологічних даних вимагає від дослідників вирішення однієї з основних проблем біоінформатики, що пов'язана з великим відставанням кількості описаних послідовностей в порівнянні з автоматично проанатованими. З появою у 1995 році NGS, секвенування нового покоління, відставання тільки збільшувалось.

Вирівнювання є одним з основних методів біоінформатики, що дозволяє оцінити схожість двох послідовностей та зробити висновок щодо їх гомології. Гомологічні послідовності мають спільне походження, виконують однакові функції та формують однакові просторові сполучення. Якщо гомологія доведена, дослідник може частково перенести анотацію, що значно полегшує опис нової послідовності [1, с. 23].

**Постановка задачі.** В процесі еволюції в генетичних послідовностях можуть відбуватись три види подій: заміни, вставки та делеції (видалення). Дистанцію Левенштейна можна визначити за допомогою алгоритму Вагнера-Фішера через кількість замін, вставок та делецій, необхідних для перетворення однієї послідовності в іншу. В результаті вирівнювання можуть бути отримані декілька варіантів, що мають мінімальну дистанцію. Тому існує поняття оптимального вирівнювання. Деякі еволюційні події мають більшу ймовірність, ніж інші. Наприклад, видалення великої ділянки послідовності більш ймовірне, ніж видалення багатьох коротких. Тому базовим принципом вирівнювання є встановлення найбільш ймовірного сценарію еволюційних подій, а саме: максимізація кількості збігів при мінімізації кількості пропусків та замін [2, с. 42].

**Алгоритм глобального вирівнювання.** Алгоритм вирівнювання біологічних послідовностей оснований на методі динамічного програмування. Для проведення вирівнювання в першу чергу необхідно встановити бали за певні еволюційні події: бали за збіг та заміну та штраф за пропуск. В результаті роботи алгоритму отримуємо вагу вирівнювання як суму всіх балів та штрафів. Вага вирівнювання є кількісною характеристикою редакційної відстані між двома послідовностями. У випадку вирівнювання амінокислотних послідовностей бал за збіг та штраф за заміну беруться з матриць замін таких як PAM або BLOSUM [3, с. 11]. Штраф за пропуск встановлюється на свій розсуд.

Метод динамічного програмування передбачає побудову матриці вирівнювання та пошук шляху зворотного проходу. Вирівнювання поділяється на три етапи: ініціалізація матриці вирівнювання F, заповнення матриці та зворотній прохід.

На етапі ініціалізації розраховується значення для нульових стовпця та рядка відповідно до співвідношення:

$$F(i,0) = -i \cdot d,$$
$$F(0,j) = -j \cdot d,$$

де $d$ — штраф за пропуск.

Етап заповнення матриці

Для кожної окремої клітини розраховується значення оптимального проходу за формулою:

$$F(i,j) = \max\begin{cases} F(i-1,j-1) + g(i,j) \\ F(i-1,j) - d \\ F(i,j-1) - d \end{cases}$$

$$g(i,j) = \begin{cases} m, if\ S1[i] = S2[j] \\ s, if\ S1[i] \neq S2[j] \end{cases}$$

В кожній комірці зберігається два значення: бал за проходження даної комірки та шлях у попередню комірку (по діагоналі, вгору або вліво).

В результаті заповнення матриці отримаємо максимальну вагу, що буде записна в комірці з координатами ($n$, $m$). Також з цієї комірки починається зворотній прохід. Рухаючись за оптимальними напрямками кожної комірки, ми встановлюємо шлях зворотного проходу з координат ($n$, $m$) до (0,0). Зворотній прохід дозволяє отримати оптимальне вирівнювання [4, с.12]. Даний алгоритм має однакову часову та просторову складність, де $N$ та $M$ — довжини послідовностей, що вирівнюються [4, с. 12].

**Особливості реалізації алгоритму з афінною штрафною функцією.** Класичний підхід не враховує усіх особливостей еволюційних подій. Відомо, що делеція блоку амінокислот або нуклеотидів має більшу ймовірність, ніж одиничне видалення [2, с. 45]. Тому алгоритм було вдосконалено афінною штрафною функцією. Ідея полягає в окремих штрафах за відкриття пропуску та за його продовження, при чому перший має бути відчутно більшим за другий. Для реалізації цієї ідеї необхідно зберігати в матриці не одне значення, а три, окремо для кожного напряму руху, для того щоб врахувати всі можливі варіанти продовження пропуску. Значення для напрямку по діагоналі позначають $M$, для напряму вліво $I_x$ та вгору $I_y$ [4, с. 17].

Ініціалізація матриці $F$ проводиться відповідно до співвідношення:

$$M(0,0)=0,\ I_x(0,0)=0,\ I_y(0,0)=0$$
$$I_x(i,0)=-d-(i-1)e,\ M(i,0)=I_y(i,0)=-\infty,$$
$$для\ \ i=1,\dots,n$$
$$I_y(0,j)=-d-(j-1)e,\ M(0,j)=I_x(0,j)=-\infty,$$
$$для\ \ j=1,\dots,m$$

Заповнення матриці:

$$I_x(i,j)=\max\begin{cases}M(i,j-1)-d,\\ I_x(i,j-1)-e;\end{cases}$$
$$I_y(i,j)=\max\begin{cases}M(i-1,j)-d,\\ I_y(i-1,j)-e;\end{cases}$$
$$M(i,j)=\max\begin{cases}M(i-1,j-1)+s(x_i,y_j),\\ I_x(i-1,j-1)+s(x_i,y_j),\\ I_y(i-1,j-1)+s(x_i,y_j).\end{cases}$$

При використанні цього алгоритму результат вирівнювання більше відповідає найбільш вірогідному шляху еволюції, проте в такому випадку часова складність алгоритму зростає до кубічної — $O(NM(N+M))$. Якщо з використанням класичного алгоритму вирівнювання послідовностей довжиною в 10 тисяч символів займає приблизно 1 годину, то з афінним штрафом на це потрібно майже 3 дні.

**Методи оптимізації.** Оптимізація алгоритму глобального вирівнювання з використанням афінної штрафної функції залишається актуальною до сьогодення. Варіантів оптимізації існує дуже багато, проте серед них вже є майже класичні підходи. Перед тим, як їх розглянути, хочу звернути увагу на один момент: будь які зміни алгоритму з метою покращення просторової або часової складності впливають на кінцевий результат алгоритму. Дуже важко зробити так, щоб алгоритм працював краще, але при цьому так само, як і раніше.

Перший алгоритм зустрічається майже у кожній статті про афінну штрафну функцію. Умовно метод має назву смугове вирівнювання (Рис. 1). Найчастіше шлях зворотного проходу пролягає поблизу головної діагоналі, і частина матриці взагалі не використовується. Ідея методу полягає в тому, щоб з самого початку обмежити матрицю зоною певної ширини поблизу головної діагоналі і розраховувати матрицю лише в межах цієї зони [4, с. 20]. Це дає виграш у просторі, бо матриця займає менше пам'яті, та у часі, бо потребується менше часу на заповнення матриці. Складність такого алгоритму дорівнює $O(WM(W+M))$, де $W$ — ширина смуги, яка має бути меншою за $N$. Але такий підхід має певні недоліки. По-перше, є ризик занадто обмежити матрицю і через це втратити оптимальне вирівнювання, якщо воно пролягає за межами встановленої зони. Для невеликих послідовностей ця проблема вирішується встановленням нового значення ширини та перерозрахунком вирівнювання. Проте для таких невеликих послідовностей початкова просторова та часова складності також не критичні. Проблеми починаються на дуже великих послідовностях. І якщо за рахунок виграшу у часі втрачається оптимальне вирівнювання та необхідно знову запускати алгоритм, то така оптимізація не має сенсу. По-друге, при неграмотній реалізації цього методу є ймовірність, що час та простір, зекономлені на порожніх частинах матриці, будуть витрачені на додаткові розрахунки та організацію внутрішніх даних програми, тобто ми отримаємо майже ту саму складність, але з ймовірністю отримати неоптимальне вирівнювання.

Наступний метод оптимізації — це оптимізація за допомогою алгоритму Міллера-Маєрса (Рис. 2). Цей метод направлений в першу чергу на покращення просторової складності [5, с. 287]. Одна з послідовностей ділиться навпіл, тобто в матриці вирівнювання обирається центральний рядок. Знаходиться оптимальний шлях від клітини з координатами (0,0) до центрального рядка та від цього рядка до координат ($n$, $m$). Таким чином на центральному рядку знаходимо точку $X$, що однозначно належатиме оптимальному вирівнюванню. Далі матриця ділиться на квадранти на основі точки $X$. Можна однозначно сказати, що оптимальне вирівнювання пролягатиме у лівому верхньому та правому нижньому квадрантах, отже розрахунки матриці у інших квадрантах можна видалити з пам'яті. Далі процедура ділення навпіл повторюється у квадрантах до отримання усіх точок оптимального вирівнювання.

Цей алгоритм займає в найгірший момент в половину менше пам'яті, ніж оригінальний алгоритм, оскільки під час знаходження першої точки X розраховується лише половина матриці. Серед інших переваг алгоритму вказано, що кожний квадрант можна обчислювати у окремих незалежних потоках одночасно. Проте багатопоточність не дає реального виграшу

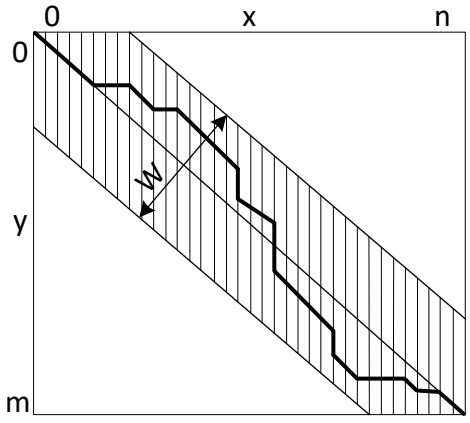


Рис. 1. Смугове вирівнювання

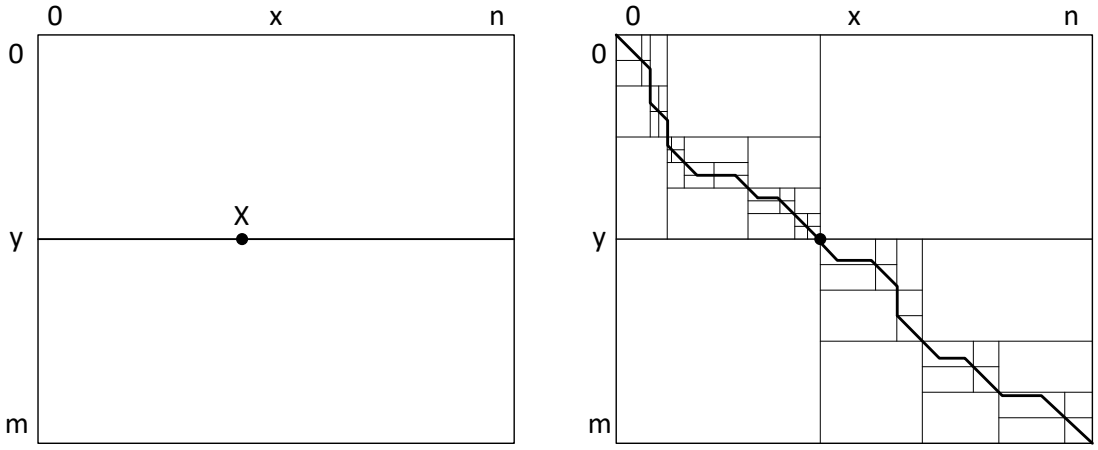


Рис. 2. Алгоритм Міллера-Маєрса

у часі на комп'ютерах зі стандартною архітектурою. До того ж ускладнення розрахунків приводить до того, що часова складність зростає до $n^4$. Для ефективного використання алгоритм Міллера-Маєрса можна реалізувати на графічному процесорі з використанням OpenCL. Тоді кожен квадрант буде обчислюватись одночасно завдяки справжній багатопоточності, що дійсно дасть виграш у часі [5, с. 289].

**Реалізація алгоритму афінної штрафної функції за квадратичний час.** Для досягнення більш кардинальної оптимізації афінної штрафної функції необхідно було змінити підхід до самої оптимізації. В результаті було розроблено новий алгоритм з афінним штрафом, який має квадратичну часову складність. За основу було взято алгоритм вирівнювання без афінного штрафу. Маємо прапорці DIAG, UP, LEFT, що вказують напрям зворотного проходу. Кожна комірка містить два значення: бал за проходження комірки та прапорець оптимального шляху.

Ініціалізація матриці відбувається за співвідношенням:

$$F(0,j) = d_{open} + d_{ext} \cdot (j-1),$$

$$F(0,j) = d_{open} + d_{ext} \cdot (j-1),$$

де $d_{\text{open}}$ — штраф за відкриття пропуску,

$d_{\text{ext}}$ — штраф за продовження пропуску.

Заповнення матриці. Для кожної клітини необхідно встановити бал для кожного з трьох шляхів. Для шляху по діагоналі знаходиться значення $g$ з матриці амінокислотних замін PAM або Blosum. Для встановлення виду штрафу для проходу вгору або вліво треба перевірити значення прапорця для попередньої комірки окремо для обох напрямків. Якщо в попередній комірці не було штрафу за розрив (прапорець має значення DIAG), тоді встановлюємо штраф $d_{\text{open}}$. В протилежному випадку (прапорець має значення UP або LEFT) береться штраф за продовження $d_{\text{ext}}$. В коді мовою Python це має вигляд:

```
if F_flag[i - 1][j] == DIAG:
   d_up = d_open
else:
   d_up = d_ext
if F_flag [i][j – 1] == DIAG:
   d_left = d_open
else:
   d_left = d_ext
```

*Таблиця 1*

**Порівняння часової складності розглянутих алгоритмів**

| № | Назва алгоритму | Часова складність |
|---|---|---|
| 1 | Класичний алгоритм | $O(nm) \approx O(n^2)$ |
| 2 | З афінним штрафом | $O(nm(n+m)) \approx O(n^3)$ |
| 3 | Смугове вирівнювання | $O(nw(n+w)) < O(n^3)$ |
| 4 | Алгоритм Міллера-Маєрса | $O(n^4)$ |
| 5 | Новий оптимізований алгоритм | $O(n^2)$ |

Далі розраховуємо значення балу за проходження клітини за співвідношенням:

$$F(i,j)=\max\begin{cases}F(i-1,j-1)+g,\\ F(i-1,j)+d_{up},\\ F(i,j-1)+d_{left}.\end{cases}$$

У комірці зберігаємо отримане значення та відповідний прапорець. Процедура зворотного проходу така ж сама, як при оригінальному алгоритмі.

**Висновки.** Було розглянуто алгоритми глобального вирівнювання біологічних послідовностей та основні принципи, на яких ці алгоритми базуються. Доведено необхідність оптимізації алгоритму з афінною штрафною функцією. Розглянуто запропоновані методи оптимізації та порівняна їх ефективність (Табл. 1).

Порівнювалась часову складність, оскільки з сучасним рівнем розвитку технологій просторова складність не є критичною. Смугове вирівнювання дає виграш у часі в залежності від величини смуги $w$, проте є ймовірність отримати неоптимальне вирівнювання. Алгоритм Міллера-Маєрса має навіть більшу часову складність на комп'ютерах зі стандартною архітектурою, проте може бути дуже виграшним у реалізації на графічних процесорах. Також було описано новий розроблений алгоритм з афінною штрафною функцією, який має квадратичну часову та просторову складності.

**Література**

1. Основы биоинформатики / С. Игнасимуту // Институт Компьютерных исследований, R&C Dynamic — Москва, Ижевск — 2007. — 324 с.

2. Методы биоинформационного анализа / Огурцов А. Н. — НТУ «ХПИ» — 2011. — 114 с.

3. Молекулярная эволюция и филогенетический анализ / Лукашов В. В. — ИД «БИНОМ». — 2009. — 256 с.

4. Биоинформатика. Множественное выравнивание. Филогенетические деревья / Васильева Н. Ю. — Одесса. — ОНУ. — 2014. — 70 с.

5. Algorithms in Bioinformatics. Pairwise sequence alignment [Електронний ресурс] / D. Huson // Bioinformatics I, WS'14/15. — November 1. — 2014. — с. 8.

6. Выравнивание двух ДНК последовательностей, оптимальное для реализации на графическом поцессоре [Електронний ресурс] / Гриорьев А. В. // Вестник СГТУ. — 2012. — № 1. — Вип. 2. — с. 285.

7. Молекулярная эволюция и филогенетика / М. Ней, С. Кумар. — Киев КВЩ — 2004. — 421 с.

8. Dynamic Gap Selector: A Smith Waterman Sequence Alignment Algorithm with Affine Gap Model Optimisation [Електронний ресурс] / Gianvito Urgese, Giulia Paciello // IWBBIO 2014. — Granada 7–9 April. — с. 1347.

9. Математические методы анализа алгоритмов / Д. Грин. — ИД «Мир». — 1987. — 120 с.