ДОКЛАДЫ АКАДЕМИИ НАУК, 2019, том 484, № 4, с. 401–404

МАТЕМАТИКА

УДК 004.021

МЕТОД БЫСТРОГО МНОЖЕСТВЕННОГО ПОПАРНОГО ВЫРАВНИВАНИЯ НА ОСНОВЕ ПРЕФИКСНЫХ ДЕРЕВЬЕВ

П. А. Яковлев

Представлено академиком РАН К.В. Рудаковым 05.09.2018 г.

Поступило 18.08.2018 г.

Представлен метод для эффективного сравнения символьной последовательности со всеми строками из некоторого множества, работающий существенно быстрее, чем наивный перебор сравнений со всеми строками подряд. Для ускорения процедуры предлагается оригинальный алгоритм, объединяющий использование префиксного дерева и стандартного алгоритма динамического программирования для поиска редакционного расстояния (метрики Левенштейна) между строками. Эффективность метода подтверждена в вычислительных экспериментах на массивах в десятки миллионов биологических последовательностей вариабельных доменов моноклональных антител.

Ключевые слова: выравнивание последовательностей, строковые алгоритмы, биоинформатика, динамическое программирование, префиксные структуры данных.

DOI: https://doi.org/10.31857/S0869-56524844401-404

Сравнение символьных последовательностей имеет широкое применение в прикладных алгоритмах для совершенно различных областей: от анализа текстов до биоинформатики. Большую популярность здесь имеют методы, основанные на редакционном расстоянии Левенштейна [1], или, иначе, выравнивании (alignment) строк. Элементарные модификации данной метрики позволяют сравнивать строки с учётом стоимости замен символов [2], учитывать инверсии [3], искать наибольшие общие подстроки с ошибками [4], пересечения и вложения строк [5] и решать многие другие задачи.

Обратимся к оригинальной формуле вычисления расстояния для строк s и t длины N и M соответственно. Через $D_{i,j}$ будем обозначать редакционное расстояние между префиксом строки s длины i и префиксом строки t длины j. Стоимость редакционных операций определяется весами $w_{k,l}$, определяющими стоимость замены k-го символа из s на l-й символ из t, а также весами удаления и вставки символов w_{del} и w_{ins} соответственно:

$$D_{k,0} = i \cdot w_{\text{del}},$$

$$D_{0,j} = j \cdot w_{\text{ins}},$$

$$D_{i,j} = \min \begin{cases} D_{i-1,j} + w_{\text{del}}, \\ D_{i,j-1} + w_{\text{ins}}, \\ D_{i-1,j-1} + w_{i,j}. \end{cases}$$

ЗАО “Биокад”, Санкт-Петербург
E-mail: yakovlev@biocad.ru

Редакционное расстояние двух строк в данном случае будет равно $D_{N,M}$. Прямое вычисление по формуле ведёт к экспоненциальной сложности, которая, однако, может быть значительно снижена с помощью динамического программирования. Алгоритм Вагнера–Фишера [6] позволяет достигнуть вычисления за $O(NM)$ времени и памяти. Для этого вычисление строится путём последовательного заполнения матрицы размера $(N + 1) \times (M + 1)$. При этом для вычисления каждой следующей ячейки требуется только три: левая, верхняя и диагональная (слева сверху). Первая строка и аналогично первый столбец, представляют собой редакционное расстояние между одной из последовательностей и пустой строкой, а потому могут быть заполнены без информации о второй последовательности. Дальнейшее заполнение может происходить равно как по столбцам, так и по строкам (см. алгоритм 1 далее).

Помимо построения матрицы с последующим получением расстояния возможен также “обратный проход”, задачей которого является восстановление набора редакционных операций, приведших к оптимальному выравниванию строк (рис. 1). Подобный проход осуществляется за $O(N + M)$ и не меняет асимптотической сложности алгоритма.

Сложность алгоритма, несмотря на значительный выигрыш по сравнению с “наивным” экспоненциальным вариантом, затрудняет использование данного метода для поиска ближайшей

Алгоритм 1

1: **for** $j \in [0 \ldots M]$ **do** ▷ Заполнение первой строки
2: $D_{0,j} \leftarrow j \cdot w_{\text{ins}}$
3: **end for**
4: **for** $i \in [1 \ldots N]$ **do**
5: $D_{i,0} \leftarrow i \cdot w_{\text{del}}$ ▷ Заполнение первого столбца
6: **for** $j \in 1 \ldots M$ **do** ▷ Заполнение столбцов строки
7: $D_{i,j} \leftarrow \min(D_{i-1,j} + w_{\text{del}}, D_{i,j-1} + w_{\text{ins}}, D_{i-1,j-1} + w_{i,j})$
8: **end for**
9: **end for**

	_	F	A	T	H	E	R
_	0	1	2	3	4	5	6
F	1	0	1	2	3	4	5
A	2	1	0	1	2	3	4
Z	3	2	1	1	2	3	4
E	4	3	2	2	2	2	3
R	5	4	3	3	3	3	2

F A T H E R
F A Z - E R

Рис. 1. Матрица расстояний и схема обратного прохода для получения выравнивания.

последовательности к некоторой заданной в множестве большого объёма, поскольку в этом случае для каждого элемента этого множества требуется провести однотипное выравнивание с рассматриваемой. Рассмотрим способ снижения сложности подобной задачи.

Цикл на строке 4 алгоритма 1, очевидно, можно разбить на два последовательных цикла, итерируя сначала $i \in [0, \ldots, K]$, а затем $i \in [K+1, \ldots, N]$, где $K \in [0, \ldots, N]$, без какого-либо изменения в алгоритме. При этом вид матрицы $D_{0:K,0:M}$ (матрица, образованная строками с нулевой по K-ю исходной матрицы из алгоритма Вагнера–Фишера) не будет зависеть от последующего суффикса второй строки. Это означает, что при выравнивании некоторой строки t на строки s и s' длин N_1 и N_2 соответственно, обладающих одинаковым префиксом длины K, первые $K+1$ строки матрицы будут совпадать. Мы можем воспользоваться этим наблюдением для построения алгоритма 2.

Суммарная сложность двух выравниваний будет $O((N_1 + N_2 - K)M)$ вместо $O((N_1 + N_1)M)$ в случае использования двух независимых выравниваний. Очевидно выигрыш тем значительнее, чем больше строк s_i с одинаковым префиксом мы будем рассматривать. Это и становится ключом к построению метода поиска выравнивания строки t на все строки s_i из некоторого словаря. В этом нам поможет структура префиксного дерева, или бора. Эта структура часто используется в поисковых автоматах, когда требуется работать одновременно с большим количеством строк. В частности, широко применимы модификации алгоритма Ахо–Корасик [7], использующиеся для составления чёрных или белых

Алгоритм 2

1: $fill\_first\_row(D, M)$
2: $K \leftarrow longest\_common\_prefix\_length(s, s')$
3: **for** $i \in [1 \ldots K]$ **do** ▷ Выравнивание общего префикса
4: $fill\_row(D, i, s_{0:K}, t)$
5: **end for**
6: **for** $i \in [K+1 \ldots N_1]$ **do** ▷ Выравнивание суффикса первой строки
7: $fill\_row(D, i, s_{K+1:N_1}, t)$
8: **end for**
9: $result_{s,t} \leftarrow D_{N_1,M}$
10: **for** $i \in [K+1 \ldots N_2]$ **do** ▷ Выравнивание суффикса второй строки
11: $fill\_row(D, i, s'_{K+1:N_2}, t)$
12: **end for**
13: $result_{s',t} \leftarrow D_{N_2,M}$

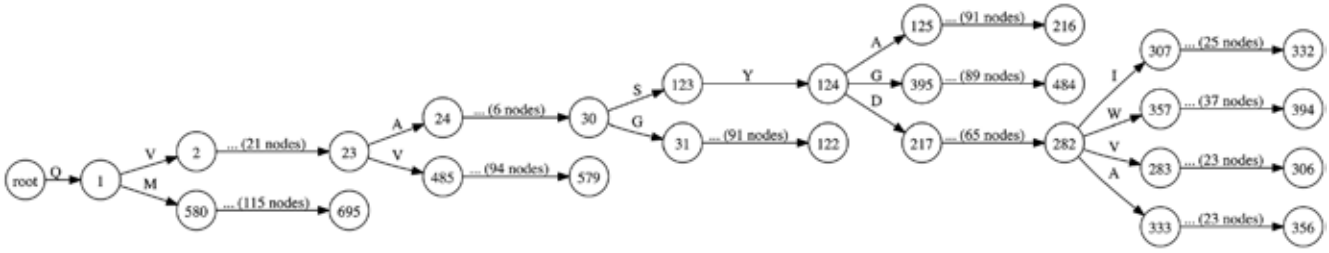

Рис. 2. Префиксное дерево, построенное по 10 последовательностям антител (сжатый вид).

списков слов, а также поиска заранее определённых паттернов в тексте.

При добавлении последовательностей в префиксное дерево получается структура, в которой символы последовательности написаны на рёбрах, соединяющих узлы, содержащие произвольную информацию. Мы будем хранить в этих узлах индексы, описывающие порядок добавления элемента в бор. Пример подобной структуры приведён на рис. 2.

Теперь, пользуясь предыдущим наблюдением и полученной структурой, мы можем построить алгоритм, сочетающий обход префиксного дерева в глубину с заполнением матрицы расстояний. На прямом ходе поиска в глубину матрица выравнивания будет построчно заполняться с прохождением каждого ребра. Таким образом в каждый момент времени матрица будет отражать выравнивание строки-запроса на текущий путь в дереве от корня, что даёт возможность в любой момент времени получить как текущую стоимость выравнивания, так и восстановить набор редакционных операций. При обратном ходе по дереву для перехода в другую ветвь очищаются неактуальные строки матрицы, которые далее заполняются в соответствии с новым суффиксом, определяемым следующей рассматриваемой ветвью (см. алгоритм 3).

Полученный алгоритм легко модифицируется для работы с аффинными пропусками и прочими вариантами алгоритма выравнивания, упомянутыми в начале сообщения. Помимо таких модификаций, алгоритм также можно ускорить за счёт небольших изменений. Например, при добавлении строк в бор для каждого следующего узла за постоянное время можно считать скользящую хэш-функцию для суффикса фиксированного размера [8]. Сохранение отображения таких хэшей в соответствующие им списки узлов позволяет отфильтровать ветви дерева, содержащие конкретные

Алгоритм 3

```
1: function align(query, trie, callback)
2:     matrix ← new_matrix(max_depth(trie) + 1, |query| + 1)
3:     fill_first_row(matrix, query)
4:     previous_node ← None
5:     fork_stack ← empty_stack()
6:     for node in dfs_order(nodes(trie)) do                ▷ см. рис. 3
7:         if is_leaf(previous_node) then
8:             while top(fork_stack) ≠ parent(node) do
9:                 row_shift ← distance(top(fork_stack), previous_node)
10:                move_row(matrix, row_shift)              ▷ Сдвигаем текущую строку матрицы наверх
11:                pop(fork_stack)
12:            end while
13:        end if
14:        fill_row(matrix, query, symbol(node))
15:        if is_fork(node) then
16:            push(fork_stack, node)
17:        end if
18:        if is_leaf(node) then
19:            callback(matrix, query, sequence_to(node))
20:        end if
21:        previous_node ← node
22:    end for
23: end function
```

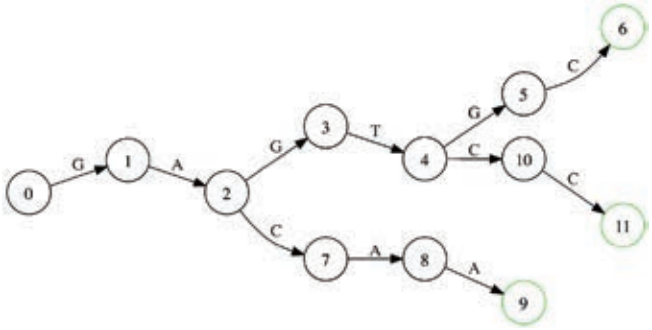

Рис. 3. Порядок обхода дерева в порядке DFS: 0, 1, 2, 3, 4, 5, 6, **10**, 11, **7**, 8, 9.

паттерны и проводить выравнивание только на данные ветви. Другой модификацией является хранение в аналогичном боре всех последовательностей запросов. Модификация алгоритма позволяет осуществить выравнивание двух боров друг на друга, используя преимущества сжатия префиксов для обоих наборов последовательностей.

Практическое применение описанных методов даёт значительный выигрыш при работе с биологическими последовательностями B- и T-клеточных рецепторов, а также антител. Особенность биологического процесса V(D)J-рекомбинации, лежащей в основе образования иммунных рецепторов [9], даёт группы существенно совпадающих префиксов. Значение компрессии как отношения общего числа символов в базе к числу рёбер в дереве в отдельных случаях достигает 10–12 раз, а в среднем составляет 5–6 раз. Использование выравнивания на основе префиксных деревьев обеспечило возможность сравнения со словарём, состоящим из миллионов последовательностей, а сжатие данных позволило уместить такой словарь в память графического ускорителя и получить эффективную GPGPU-реализацию.

СПИСОК ЛИТЕРАТУРЫ

1. *Левенштейн В.И.* // ДАН. 1965. Т. 163. № 4. С. 845–848.
2. *Needleman S.B., Wunsch C.D.* // J. Mol. Biol. 1970. V. 48. № 3. P. 443–453.
3. *Damerau F.J.* // Commun. ACM. 1964. V. 7. № 3. P. 171–176.
4. *Smith T.F., Waterman M.S.* // J. Mol. Biol. 1981. V. 147. № 1. P. 195–197.
5. *Brudno M., et al.* // Bioinformatics. 2003. V. 19. P. 54–62.
6. *Wagner R.A., Fischer M.J.* // J. ACM. 1974. V. 21. № 1. P. 168–173.
7. *Aho A.V., Corasick M.J.* // Commun. ACM. 1975. V. 18. № 6. P. 333–340.
8. *Cohen J.D.* // ACM Trans. Inf. Sys. 1997. V. 15. № 3. P. 291–320.
9. *Market E., Papavasiliou F.N.* // PLoS Biol. 2003. V. 1. № 1. P. 24–27.

FAST TRIE-BASED METHOD FOR MULTIPLE PAIRWISE SEQUENCE ALIGNMENT

P. A. Yakovlev

Presented by Academician of the RAS K.V. Rudakov September 5, 2018

Received August 18, 2018

A method for efficient comparison of a symbol sequence with all strings of a set is presented, which performs considerably faster than the naive enumeration of comparisons with all strings in succession. The procedure is accelerated by applying an original algorithm combining a prefix tree and a standard dynamic programming algorithm searching for the edit distance (Levenshtein distance) between strings. The efficiency of the method is confirmed by numerical experiments with arrays consisting of tens of millions of biological sequences of variable domains of monoclonal antibodies.

Keywords: sequence alignment, string algorithms, bioinformatics, dynamic programming, prefix data structures.