МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В. Ломоносова
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ

ISSN 2411-1473

Современные информационные технологии и ИТ-образование

Научный журнал

Том 2 (№ 11)

Москва
2015

УДК [004:377/378](063)
ББК 74.5(0)я431+74.6(0)я431+32.81(0)я431
С 56

Современные информационные технологии и ИТ-образование. Т. 2 (№ 11), 2015. - 614 с. (ISSN 2411-1473)

В данном выпуске журнала представлены доклады X Юбилейной международной научно-практической конференции «Современные информационные технологии и ИТ-образование», прошедшей в Московском государственном университете имени М.В. Ломоносова 20-22 ноября 2015 года.

Журнал «Современные информационные технологии и ИТ-образование» включен в наукометрическую базу «Российский индекс научного цитирования» с размещением полнотекстовых версий в научной электронной библиотеке eLIBRARY.RU. URL: http://elibrary.ru/title_about.asp?id=52785

Издание осуществлено при финансовой поддержке Российского фонда фундаментальных исследований (Грант РФФИ № 15-07-20760_г)

Учредитель:
Фонд содействия развитию интернет-медиа, ИТ-образования, человеческого потенциала «Лига интернет-медиа»

Издатель:
Фонд содействия развитию интернет-медиа, ИТ-образования, человеческого потенциала «Лига интернет-медиа»

Адрес редакции:
119991, г. Москва, ГСП-1, Ленинские горы, д. 1, стр. 52, факультет ВМК МГУ имени М.В. Ломоносова, каб. 375. E-mail: sukhomlin@mail.ru, тел./факс (495) 939-46-26.

Журнал зарегистрирован Федеральной службой по надзору в сфере связи, информационных технологий и массовых коммуникаций (Роскомнадзор).
Свидетельство о регистрации средства массовой информации ПИ № ФС77-61433 от 10 апреля 2015 г.
Издается с 2005 года. Выходит 1 раз в год.

Рубанов Л.И.[1], Селиверстов А.В.[2], Любецкий В.А.[3]

[1]ИППИ РАН, г. Москва, к.т.н., в.н.с., rubanov@iitp.ru

[2]ИППИ РАН, г. Москва, к.ф.-м.н., в.н.с., slvstv@iitp.ru

[3]ИППИ РАН, г. Москва, д.ф.-м.н., зав.лаб., lyubetsk@iitp.ru

ШИРОКОМАСШТАБНЫЙ ПОИСК УЛЬТРАКОНСЕРВАТИВНЫХ ЭЛЕМЕНТОВ В ПОЛНЫХ ГЕНОМАХ

КЛЮЧЕВЫЕ СЛОВА

Большие данные, высокопроизводительные вычисления, полный геном, ультраконсервативный элемент, полулокальное выравнивание, плотный подграф, кластеризация.

АННОТАЦИЯ

Описывается метод совместного анализа большого числа полных геномов с целью нахождения в них ультраконсервативных элементов – похожих друг на друга (не обязательно тождественных) участков, встречающихся сразу во многих геномах. Задача связана с обработкой больших данных, для ее решения разработан ряд оригинальных параллельных алгоритмов с линейной или близкой к ней сложностью.

Введение и постановка задачи. Задача состоит в поиске ультраконсервативных элементов в данном наборе геномов, последовательностей порядка 3 миллионов – 6 миллиардов букв в четырёхбуквенном алфавите. Число геномов может достигать нескольких сотен. Нужно кластеризовать слова, подпоследовательности небольшой длины, взятые из данных последовательностей, таким образом, чтобы один кластер состоял из «почти одинаковых» слов. Последнее означает, что каждый кластер, называемый ещё ультраконсервативным элементом, допускает хорошее выравнивание входящих в него слов, т.е. редакционное расстояние между ними небольшое. Забегая вперёд, отметим, что типичная возникающая трудность состоит в том, чтобы не было слишком больших («гигантских») кластеров. Это зависит от правильного выбора параметров задачи, что само по себе является нетривиальной задачей.

Задача относится к числу типичных трудных задач обработки больших данных, о чем свидетельствуют характерные размеры полных геномов, а также число уже известных геномов и темпы расшифровки новых. На неформальном уровне каждый такой элемент представляет собой участок ДНК, который встречается одновременно во многих геномах, но не обязательно в виде точной копии, а с возможностью замен, вставок и удалений отдельных букв. Таким образом, задача соединяет в себе две важнейшие задачи – кластеризацию слов и построение их множественного выравнивания.

Более строго, рассматривается следующая задача. Даны *M* символьных последовательностей над конечным алфавитом Σ. Любую конечную последовательность $A = \langle a_1 a_2 \ldots a_N \rangle$ будем называть строкой, а произвольный ее отрезок между позициями *i* и *j* включительно – подстрокой (или *словом*), которую обозначим $A[i..j] \subseteq A$. Длины строк могут различаться, но не превосходят заданной величины *N*; без ограничения общности можно считать длину каждой строки равной *N*. Для пары слов определено редакционное расстояние [1, 2] λ, используя фиксированные цены элементарных операций, которые предполагаются симметричными и однородными, т.е. цена вставки равна цене удаления ($\delta_i = \delta_d = \delta_{id}$) и цены всех замен равны между собой. Часто предполагают также, что цена вставки/удаления больше цены замены, $\delta_{id} > \delta_s$, но для простоты изложения будем временно считать $\delta_{id} = \delta_s = 1$; при этом редакционное расстояние между двумя словами равно минимальному числу операций, преобразующих одно слово в другое.

Определение. Будем говорить, что слово $w = \langle w_1 w_2 \ldots w_l \rangle$ *встречается* в данной строке

$A = \langle a_1 a_2 \ldots a_N \rangle$, если в ней существуют две позиции $1 \le i \le j \le N$, такие, что редакционное расстояние $\lambda(w, A[i,j]) \le \varepsilon$, где ε – заданный порог. Для таких случаев будем использовать обозначения $w \in A, w \approx A[i..j]$.

Задача состоит в том, чтобы найти все слова длины не менее $l = N$, которые встречаются, по меньшей мере, в $m \le M$ строках, с указанием их позиций в каждой строке.

Согласно определению, здесь под «встречей» понимается не только точное совпадение слов, т.е. равенство алфавитных символов на соответственных позициях, но и приблизительное совпадение, когда слова неодинаковы, но близки в смысле редакционного расстояния. При этом порог ε предполагается небольшим, например, не более 6 редакционных операций (замен, вставок, удалений) при длине $l = 150$. Основная сложность вытекает из размерности задачи: длина строк *N* очень велика (общая длина полного генома составляет порядка 10^9), а число строк *M* может достигать сотен. Трудоемкость прямого перебора квадратичная и составляет порядка $O(M^2 N^2 l^2)$ операций сравнения, т.е. свыше 10^{25}, а для параллельного перебора необходима общая память размером по меньшей мере $O(MN)$ – терабайтного объема. Это не позволяет решать такую задачу даже с использованием современных суперкомпьютеров. Требуется более эффективная технология, основанная на применении быстрых алгоритмов и лучше приспособленная к распараллеливанию. С теоретико-графовых позиций предлагаемый метод состоит из двух этапов.

На *первом этапе* анализируются две строки и в них выделяются все пары искомых слов-кандидатов (для определенности – слов второй строки, которые встречаются в первой строке). Каждая найденная пара слов задает ребро будущего графа, в котором вершины соответствуют словам, а ребрами соединены вершины, для которых редакционное расстояние между соответствующими словами не превосходит ε. Для поиска разработан быстрый алгоритм с линейной (или почти линейной) сложностью. При наличии в составе вычислительной системы $M(M-1)/2$ процессоров данный этап может выполняться параллельно для каждой неупорядоченной пары строк, независимо от прочих пар. Однако алгоритм наиболее эффективен, когда в каждой параллельной ветви выполняется сравнение одной строки со всеми оставшимися, для чего достаточно $M-1$ процессоров. Возможны также и промежуточные по числу ветвей варианты, позволяющие лучше сбалансировать неоднородную вычислительную нагрузку между процессорами; этот вопрос исследован нами в [3]. Результатом первого этапа являются независимые ребра, которые в совокупности образуют *M*-дольный граф низкой плотности (в котором число ребер ближе по порядку к числу вершин, чем к квадрату числа вершин), обычно состоящий из большого числа связных компонент.

На *втором этапе* проводится уплотнение начального графа с последующим выделением в нем плотных подграфов (в идеале – являющихся кликами) с вершинами, принадлежащими *m* и более различным долям. Каждый такой *m*-плотный подграф в целом описывает одно найденное искомое слово; вершина этого подграфа, относящаяся к *k*-й доле начального графа, определяет позицию этого слова в *k*-й исходной строке. Совокупность всех найденных *m*-плотных подграфов дает решение задачи. Уплотнение графа выполняется путем склейки вершин, которым соответствуют одни и те же или сильно пересекающиеся (не менее чем на длине *d*) слова, с одновременным удалением возникающих кратных ребер. Для поиска *m*-плотных подграфов в большом графе разработан новый параллельный алгоритм, построенный по принципу клеточного автомата, в котором каждой вершине исходного графа соответствует свой многошаговый процесс, однако эти процессы исполняются с единой синхронизацией шагов. Как показали эксперименты на реальных данных, даже для очень больших графов (например, порядка 10^7 вершин и 10^9 ребер) алгоритм завершается после небольшого числа шагов (обычно менее сотни), и гибко масштабируется для любого числа доступных процессоров. В следующих разделах излагаются подробности указанных двух этапов метода и описываются лежащие в их основе алгоритмы.

Поиск слов, встречающихся в двух строках. Вначале уточним ряд моментов, опущенных выше для краткости. Во-первых, из введенного определения встречаемости следует, что для слова $w \in A$ любое его подслово также встречается в этой строке: $u \subseteq w \Rightarrow u \in A$. Поэтому для уменьшения числа решений задачи естественно потребовать, чтобы искомые слова были

непродолжаемыми, т.е. никакое найденное слово не было подстрокой другого найденного слова. Однако такое требование может вступить в конфликт с основной целью задачи. В самом деле, легко представить ситуацию, когда некоторое слово *w* встречается в *m* исходных строках, а его подслово $u \subseteq w$ – в $m' > m$ строках. Оба эти решения могут представлять практический интерес, поэтому ограничим требование непродолжаемости слова пределами каждого выявленного множества из *m* и более строк, в которых встречается это слово.

Во-вторых, на практике часто представляют интерес не любые слова, широко встречающиеся в исходных строках, а только обладающие достаточной *колмогоровской сложностью*. Например, в полном геноме иногда присутствуют длинные подстроки, состоящие из многих тысяч повторений одного-двух алфавитных символов, например, АТ-повторы. Найденные в таких подстроках элементы обычно не представляют интереса и должны быть отброшены. Как показано ниже, это также помогает снизить затраты памяти и ускорить работу алгоритма. Для оценки сложности найденных кандидатов использован алгоритм сжатия данных Зива–Лемпеля [4] в реализации GNU Gzip [5] с установленным порогом для коэффициента компрессии *r*; величина порога определялась эмпирически.

Основная трудность этого этапа связана с квадратичной (от длины строки) сложностью непосредственного сравнения слов двух последовательностей. Известные быстрые алгоритмы поиска подстроки в строке в данном случае неприменимы, поскольку способны отыскивать только точно совпадающие слова. Требовался алгоритм более низкой сложности, который был разработан на базе трех принципов:

1) *Индексация строк.* Наивный квадратичный алгоритм проверяет для каждого слова строки *B*, встречается ли оно в одной и той же строке *A*. Ясно, что такую проверку можно в среднем существенно ускорить с помощью надлежащей индексации строки *A*, которая проводится один раз, а затем многократно используется. Эта идея хорошо известна и широко используется, например, поисковыми машинами Интернет. Подробности разработанных для данной задачи индексов обсуждаются ниже.

2) *Предварительный поиск по ключу*. В рассматриваемой постановке ищутся элементы с высокой консервативностью, так что порог допустимого редакционного расстояния между искомыми словами предполагается низким: различия между любыми двумя словами устраняются не более чем за ε операций вставки, удаления или замены одного символа. Отсюда следует, что искомые слова гарантированно содержат *точно* совпадающие подслова некоторой длины *k*, которые можно использовать в качестве обычных поисковых ключей. Например, для указанных выше значений параметров $\varepsilon = 6,\ l = 150$ длина ключа может составлять $k \approx 30$.

3) *Полулокальное выравнивание*. После поиска по точному ключу каждое подслово-кандидат подлежит расширению в обе стороны с соблюдением порога ε, пока не будет достигнута общая длина не менее *l*, иначе кандидат отвергается. Для нахождения редакционного расстояния между словами обычно применяется алгоритм Нидлмана–Вунша [6] со сложностью $O(l^2)$. Этот алгоритм глобального выравнивания предполагает концы слов зафиксированными, тогда как в нашем случае известно только начало или конец слова, в зависимости от того, в какую сторону выполняется расширение. Использование общих алгоритмов локального выравнивания (Смита–Уотермена, BLAST и др.) в рассматриваемом случае явно избыточно, коль скоро один из концов каждого слова фиксирован (примыкает к ключу). На основе алгоритма [6] был разработан полулокальный алгоритм выравнивания слов с одним закрепленным концом, причем благодаря учету порога ε и ограничению числа подряд идущих делеций *D*, созданный алгоритм имеет пространственно-временную сложность $O(Dl)$, т.е. является линейным по *l*, в отличие от исходного квадратичного алгоритма.

Пусть *k* – длина ключа, определенная в соответствии со вторым принципом. Проиндексируем строку *A* следующим образом. Строится хеш-таблица H_A, в которую поочередно заносятся все присутствующие ключи, т.е. слова длины *k* в составе строки *A,* вместе с позицией их начала в строке:

$$H_A = \left\{ h_A(i) \in \left(K_A, j\right) \mid K_A = A[j..(j+k-1)], i = Hash\left(K_A\right) \right\},$$

где $Hash(K_A)$ – хеш-функция, преобразующая значение очередного ключа в адрес внутри хеш-таблицы. При построении хеш-таблицы неизбежно возникают коллизии, которые приводят к

снижению производительности алгоритма и в данном случае имеют двоякую природу: (а) возникающие из-за несовершенства хеш-функции и (б) вызванные повторениями одного и того же слова длины *k* в исходной строке. Частоту появления коллизий первого рода можно пытаться снизить путем уменьшения коэффициента заполнения таблицы и/или выбора другой хеш-функции, но коллизии второго рода неустранимы в принципе, хотя с увеличением *k* их частота естественным образом снижается. Частоту коллизий второго рода для реальных больших данных (полный геном *Sarcocystis neurona*, длина строки *N* = 124377056, размер алфавита $|\Sigma| = 4$) иллюстрирует табл. 1. В этом примере при длине ключа $k \geq 24$ коллизии возникают в среднем менее чем в 4% случаев, но при уменьшении длины ключа до 16 эта доля превосходит 20%. Многократно повторяющиеся ключи обычно принадлежат участкам генома с низкой колмогоровской сложностью, которые в дальнейшем будут отброшены, однако для ускорения мы используем порог *t* для числа встреч ключа в анализируемой последовательности. Поскольку коллизии неизбежны, хоть и достаточно редки, для построения индекса выбран вариант хеш-таблицы с цепочками. Размер хеш-таблицы определяется исключительно длиной строки *A* и желаемым коэффициентом заполнения. Поскольку строка *B* на этом этапе не участвует, индекс для каждой из *M* исходных строк может быть построен заранее (и параллельно), а затем многократно использоваться со всевозможными *B*.

Таблица 1. Число повторяющихся ключей в строке большой длины

Число вхождений ключа	*k*=16	*k*=24	*k*=32	*k*=48
1	94919216	119034327	121193592	121823700
2	5284353	764077	401812	242260
3, 4	1438861	190900	65945	24494
5–8	486137	60275	15503	4875
9–16	183799	20869	4129	1116
17–32	72581	7429	1202	194
33–64	29861	2996	437	155
65–128	11196	1217	208	85
129–256	4986	498	86	41
257–512	1776	148	9	3
513–1024	739	67	8	6
Более 1024	447	90	3	1
Число различных ключей	102433952	120082826	121682934	122096930
Среднее число вхождений	1.21408	1.03559	1.02191	1.01833

С построенной хеш-таблицей H_A, для каждой строки *B* выполняется следующий алгоритм.

1. Поочередно от каждой позиции *j* строки *B* берем слово $K_j = B[j..(j+k-1)]$, которое будет использоваться в качестве ключа для обращения к хеш-таблице.
2. Проверяем, присутствует ли этот ключ в H_A; в случае отсутствия переходим на шаге 1 к следующей позиции *j*.
3. Если существует такое *s*, что $h_A(s) = (K_j, i)$, то найдена новая пара кандидатов (*i*, *j*): на позиции *i* в строке *A* и на позиции *j* в строке *B*. Если элемент $h_A(s)$ содержит цепочку позиций *j* и имеет место коллизия второго рода, то перебираются все пары (*i*, *j*), пропуская элементы с другими значениями ключа (при коллизии первого рода).
4. Для найденной пары кандидатов проверяем возможность расширения ключей до искомых слов и в случае успеха запоминаем их. Алгоритм расширения подробно описывается ниже, здесь просто предположим, что его сложность равна некоторой константе *C*.
5. Переходим на шаге 1 к следующей позиции *j* строки *B* вплоть до конца строки.

Если не учитывать коллизии, сложность алгоритма $O(N) \cdot C$, т.е. в среднем линейная от длины строки. Используемая память $O(N)$ также линейная, но и это может создавать трудности при очень больших *N*. Однако имеется важный резерв экономии памяти, связанный с представлением ключей в хеш-таблице. В случае короткого алфавита хешируемые ключи можно перекодировать с использованием меньшего числа бит на символ и хранить в упакованном виде в

меньшем числе байтов. Поскольку решаемая задача относится к строкам из 4-буквенного алфавита нуклеотидов (A, C, G, T), то этим способом затраты памяти на хранение ключей в хеш-таблице удается снизить в четыре раза. В этом случае удобно выбирать k кратным 4. Перед проверкой по хеш-таблице слова-кандидаты подвергаются такому же преобразованию.

В п. 4 алгоритма для найденной пары (i, j) позиций ключа в строках A и B проверяется возможность расширения ключей до искомых слов, а именно, в каждой строке ищутся положения начала, $i_1 \le i, j_1 \le j$ и конца, $i_2 \ge i+k-1, j_2 \ge j+k-1$, такие что $i_2 - i_1 \ge l-1, j_2 - j_1 \ge l-1$ и $A[i_1..i_2] \approx B[j_1..j_2]$. Согласно определению выше, последнее условие подразумевает редакционное расстояние не более ε между расширенными словами. Вариантов такого расширения может быть много; мы хотим выбрать тот, который доставляет максимум величины $L = i_2 - i_1 + j_2 - j_1$ (суммарная длин слов), что обеспечивает непродолжаемость. Для этого одновременно с расширением ключа в каждую сторону строится полулокальное выравнивание добавляемых подстрок и запоминаются те позиции, на которых редакционное расстояние между ними изменяется. Расширение выполняется параллельно в обе стороны и продолжается до тех пор, пока не будет превышен порог ε или достигнуто начало (конец) строки. В двух полученных списках позиций выбирается лучший по суммарной длине вариант расширения вправо и влево, удовлетворяющий всем вышеприведенным условиям; отсутствие такого варианта означает неуспех проверки. Учитывая малость ε, перебор всех вариантов не создает трудностей.

Опишем алгоритм расширения ключа с полулокальным выравниванием в одну сторону, для определенности – от концов ключей в сторону конца строк. Как известно, классический алгоритм [6] выравнивания двух строк с длинами m и n опирается на вычисление двух матриц F, P порядка $(m+1)(n+1)$, откуда и проистекает его квадратичная пространственная и временная сложность. Этот алгоритм неэффективен для нашего применения, потому что позиции строк A и B, до которых расширяется ключ, заранее неизвестны, а матрицы требуется вычислить полностью, несмотря на наличие известного порога ε для максимума редакционного расстояния. Для устранения этого недостатка внесены следующие изменения (рис. 2): в каждой матрице выделена область, прилегающая к главной диагонали и содержащая дополнительно по D соседних диагоналей сверху и снизу (рис. 2, а). Смысл параметра D – максимально допустимое число подряд идущих делеций. Назовем такую область D-поясом. Матрица редакционного предписания нужна для составления вышеупомянутого списка позиций расширения, на которых редакционное расстояние возрастает.

Заметим, что траектория обратного хода алгоритма оптимального выравнивания обязана лежать внутри D-пояса независимо от позиции, до которой расширяется ключ в данную сторону, иначе нарушается ограничение на число подряд идущих делеций, т.к. каждый сдвиг в сторону от диагонального направления в матрице соответствует вставке или удалению символа.

Разместим выделенные D-пояса в двух матрицах G и Q порядка $(AD+1)\times(V+1)$, где V – заранее неизвестная граница расширения вправо (рис. 2, б)). Переход от координат u, v в матрицах G, Q к координатам m, n в матрицах F, P и строках A, B осуществляется по формулам $m = v + (u-D)^+$, $n = v + (D-u)^+$ где $(D-u)^+$ означает положительную часть. С учетом этого преобразования алгоритм выравнивания строится как обычно, но очередность вычисления элементов матриц и смысл направлений указателей соответствуют указанным на рис. 2, в). Временная и пространственная сложность этого алгоритма равны $O(VD) = O(l)$, поскольку $D = l$. Таким образом, для проверки и расширения слов-кандидатов построен *линейный* от длины слов алгоритм.

Суммируя изложенное, заключаем, что общая сложность первого этапа нашего метода составляет $O(Nl)$ для одной пары строк. Для всех пар строк эта величина умножается еще на $O(M^2)$, но пары строк можно обрабатывать параллельно – по отдельности или группами, как было указано выше. Перейдем теперь ко второму этапу нашего метода.

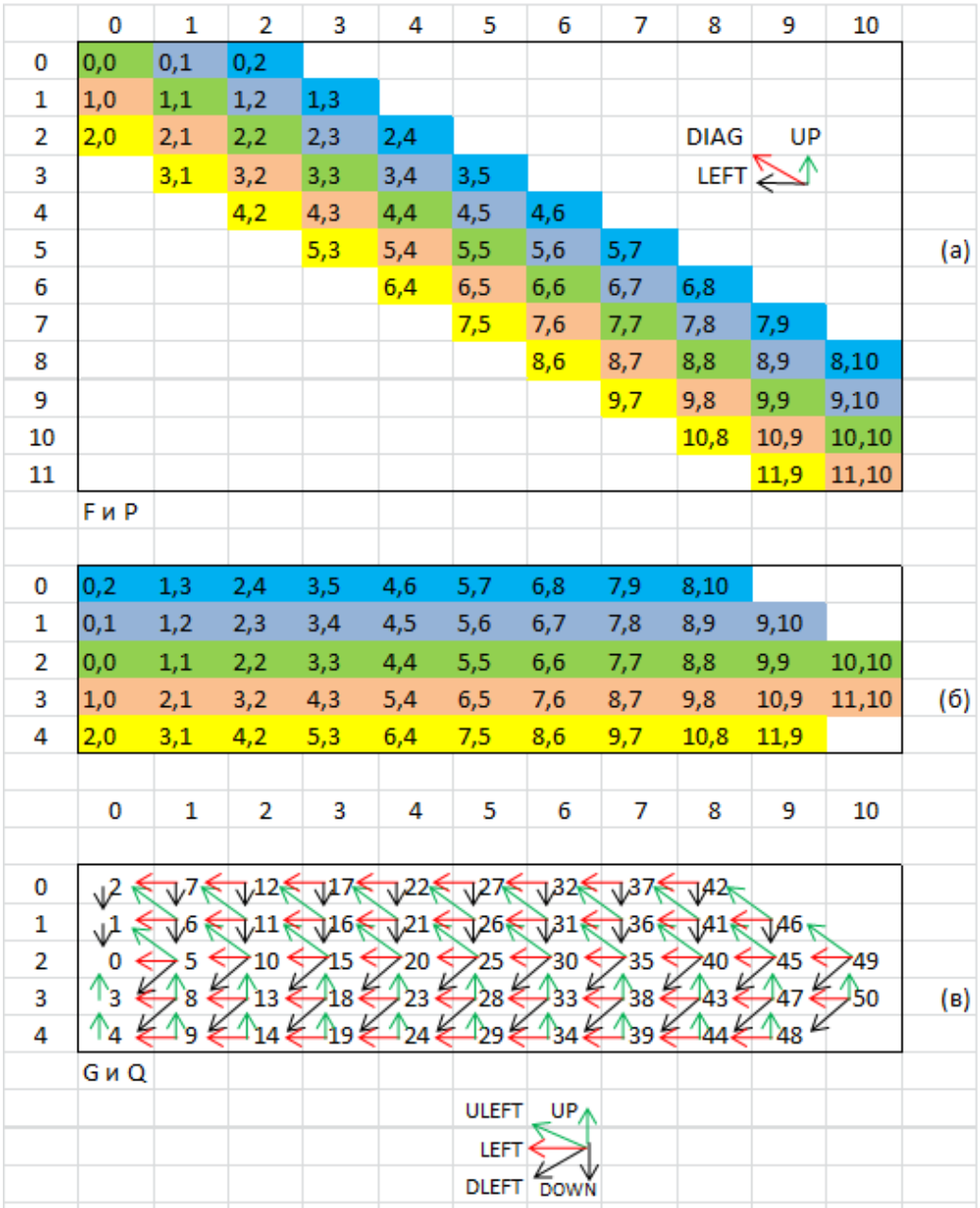

Рис. 2. Преобразование матриц алгоритма Нидлмана–Вунша: (а) вид исходной матрицы с выделенным D-поясом для m=11, n=10, D=2; в клетках записаны координаты элементов; (б) расположение D-пояса в преобразованной матрице; (в) очередность вычисления элементов преобразованной матрицы и направления указателей обратного хода в матрице редакционного предписания.

Нахождение *m*-плотных подграфов в *M*-дольном графе. После первого этапа имеем совокупность пар слов, которые принадлежат двум различным исходным строкам A и B и совпадают с точностью до редакционного расстояния ε: $A[i_1..i_2] \approx B[j_1..j_2]$. Сопоставим каждой такой паре ребро графа; этот граф содержит M долей, вершины которых соединены ребрами (внутри долей ребер нет). Рассматриваемая задача в целом эквивалентна нахождению в этом графе m-клик, т.е. подграфов с вершинами, которые принадлежат m долям, и каждая вершина соединена ребрами со всеми вершинами прочих $m-1$ долей. Это известная NP-трудная задача, для которой не существует быстрого алгоритма. Поэтому в рамках нашего метода решается близкая задача нахождения m-плотных подграфов, прежде всего – с наибольшим суммарным весом ребер. В качестве веса ребра естественно использовать монотонную функцию от длины и/или степени консервативности слов. Предполагается, что среди найденных решений этой

задачи можно по очереди отобрать присутствующие среди них клики, либо уже перебором выделить внутри них подграфы, являющиеся *m*-кликами.

Поскольку расширение точно совпадающих слов, т.е. построение вершин на конце ребра выше проводилось независимо от других ребер, на конце нескольких ребер могут стоять вершины, относящиеся к одному и тому же месту некоторой строки, но с несовпадающими концами слов и потому формально различные. Это снижает плотность всего графа и ведет к ненужному увеличению числа компонент связности, поэтому вначале проводится операция склейки вершин. При этом требуется, чтобы все объединяемые в одну вершину участки последовательности не просто пересекались попарно, а для любой пары содержали один и тот же общий фрагмент длиной не менее *d*.

После того, как выполнены все возможные операции склейки вершин, выполняется собственно алгоритм нахождения *m*-плотных подграфов. Этот алгоритм характеризуется глубоким внутренним параллелизмом и построен аналогично клеточному автомату, у которого в роли клеток выступают вершины графа, а роль соседей каждой клетки играют инцидентные ей вершины графа. В каждой вершине графа запускается независимый процесс, который состоит из последовательности шагов двух видов и начинается с шага 1. После каждого шага выполняется синхронизация процессов во всех вершинах графа, так что очередной шаг каждого процесса начинается только после окончания предыдущего шага во всех вершинах. Содержание шагов:

- Если вершина соединена ребрами менее чем с $(m-1)$ долями графа, то сама эта вершина и все инцидентные ей ребра удаляются.
- Если вершина соединена с некоторой долей только одним ребром, то помечаем его; в дальнейшем помеченное ребро может быть удалено только вместе с одним из его концов.
- Если при выполнении шага 1 в графе произошли какие-либо изменения, то снова выполняется шаг 1 во всех вершинах графа. В противном случае в каждой вершине однократно выполняется шаг 2.
- Если инцидентное вершине ребро не помечено и его вес строго меньше весов всех других не помеченных инцидентных ребер (или оно единственное), то ребро удаляется.
- Если при выполнении шага 2 в графе произошли какие-либо изменения, то все вершины снова переходят к шагу 1, иначе конец алгоритма. Каждая компонента связности получившегося графа есть искомый *m*-плотный подграф.

Данный алгоритм предоставляет гибкие возможности масштабирования вплоть до числа процессоров, равного числу вершин графа, и не требует общей памяти. Если число доступных процессоров меньше числа вершин, они равномерно распределяются между имеющимися процессорами, а затем каждый из них выполняет очередной шаг процесса по очереди для каждой назначенной ему вершины. Для этого алгоритма трудно получить аналитические оценки сложности, но это не имеет большого значения: практические расчеты с реальными большими данными показали, что второй этап предложенной технологии выполняется значительно быстрее первого.

Однако серьезная трудность состоит в том, что при неудачном выборе параметров или способа построения исходного графа в нём возникает гигантская связная компонента, объединяющая значительную часть вершин графа. Критериями успеха служат отсутствие гигантской компоненты и наличие *m*-плотных подграфов для значений *m*, больших половины от числа долей графа, то есть числа геномов. Как показали эксперименты, риск появления и размеры гигантской компоненты уменьшаются, если предварительно удалить из графа все ребра с весом меньше некоторого порога, но вопрос требует дальнейшего изучения. В любом случае, существование гигантской компоненты в исходном графе не обязательно является препятствием к выделению *m*-плотных подграфов с малым числом вершин при достаточно большом *m*.

Пример использования. Анализировались полные геномы 22 видов из надтипа Alveolata (максимальная длина генома составляла $1{,}24*10^8$). Счет проводился для значений параметров $l=60$, $k=16$, $t=200$, $\delta_s=1$, $\delta_{id}=2.1$, $\varepsilon=17.5$, $D=2$, $r=2.2$, $d=40$, $m=3$. Первый этап изложенного метода выполнялся в МСЦ РАН на суперкомьютерах МВС-100К и МВС-10П [7] с использованием 256–512 процессоров и потребовал около 200 часов. В результате было найдено $7.75\cdot10^8$ подходящих пар участков (общее число участков $4.54\cdot10^7$). Второй этап выполнялся в ИППИ РАН на 64-ядерном сервере с 256 Гб оперативной памяти. Сборка графа потребовала 13 час. на 16 процессорах, в результате был построен граф с 4977108 вершинами (и

тем же числом ребер $7.75 \cdot 10^8$, т.к. кратные ребра удалялись в процессе поиска плотных подграфов). Поиск плотных подграфов продолжался 13 час. на 22 процессорах, в результате чего было построено 845 кластеров, в которые вошли 3924 вершины (помимо гигантской компоненты, содержащей около 3 млн. вершин). Данные о числе кластеров в зависимости от числа долей, представленных в кластере: 2 с представителями из 13 долей, 1 – из 12 долей, 4 – из 11 долей, 8 – из 10 долей, 17 – из 9 долей, 24 – из 8 долей, 35 – из 7 долей, 75 – из 6 долей, 131 – из 5 долей, 290 – из 4 долей, 258 – из 3 долей. Подавляющая часть кластеров (787) содержит не более одной вершины из каждой доли, и только 9 кластеров содержат более двух вершин из какой-нибудь доли (максимум 5 вершин). Отметим, что значительная часть машинного времени была потрачена на чтение и запись огромных файлов. Анализ обнаруженных ультраконсервативных элементов и полученные биологические результаты представлены в отдельном докладе.

В заключение укажем, что описанный метод поиска ультраконсервативных элементов может быть применен и в других областях, где используется сопоставление любых текстов, в том числе на естественном языке, с целью определения авторства, стиля, заимствований и т.д. [8].

Работа выполнена за счёт гранта Российского научного фонда (проект № 14–50–00150).

Литература

1. Левенштейн В.И. Двоичные коды с исправлением выпадений, вставок и замещений символов // Доклады Академии Наук СССР. 1965, т. 163, № 4, стр. 845-848.
2. Gusfield D. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, Cambridge, UK, 1997.
3. Рубанов Л.И. Параллельное моделирование Монте-Карло на системах с распределённой памятью // *International Journal of Open Information Technologies*, 2014, т. 2, № 2, стр. 12-20.
4. Ziv J., Lempel A. A universal algorithm for sequential data compression // *IEEE Transactions on Information Theory*. 1977, vol. 23, no. 3, pp. 337-343.
5. GNU Operating System. GNU Gzip. http://www.gnu.org/software/gzip/
6. Needleman S.B., Wunsch C.D. A general method applicable to the search for similarities in the amino acid sequence of two proteins // *Journal of Molecular Biology*. 1970, vol. 48, no. 3, pp. 443-453.
7. Межведомственный суперкомпьютерный центр РАН. http://www.jscc.ru/scomputers.shtml
8. Herranz J., Nin J., Solé M. Optimal Symbol Alignment Distance: A New Distance for Sequences of Symbols // *IEEE Transactions on Knowledge & Data Engineering*, 2011, vol. 23, no. 10, pp. 1541–1554.