МЕТОДЫ ОБНАРУЖЕНИЯ
ЭМПЕРИЧЕСКИХ ЗАКОНОМЕРНОСТЕЙ
(Вычислительные системы)
2001 год Выпуск 167
УДК 519.764

УЧЕТ ПРОЯВЛЕНИЙ ПОВТОРНОСТИ, СИММЕТРИИ И ИЗОМОРФИЗМА В СИМВОЛЬНЫХ ПОСЛЕДОВАТЕЛЬНОСТЯХ: АЛГОРИТМИЧЕСКИЕ АСПЕКТЫ [1]

В.Д. Гусев, Л.А.Немытикова

Введение

Проявления повторности в символьных последовательностях (текстах) различной языковой природы *неизбежны* и *многообразны*. Говоря о неизбежности, мы подразумеваем чисто комбинаторные аспекты. Например, если размер алфавита равен 4, а длина последовательности составляет не менее 5 символов, то в ней обязательно найдутся повторяющиеся элементы алфавита. Аналогично, если при том же размере алфавита длина последовательности превысит 17 символов, в ней обязательно найдутся повторяющиеся биграммы (связные цепочки длины 2) и т.д.

Однако на практике интерес представляют не такого рода повторения, которых не удается избежать, а достаточно длинные и (или) хорошо структурированные повторы, функциональная значимость которых не вызывает сомнений. Таковы, например, длинные концевые повторы в геномах различных микроорганизмов, короткие тандемные повторы высокой кратности в некодирующих участках ДНК, устойчиво повторяющиеся словосочетания в текстах на естественном языке, повторы концевых строк в песенных мелодиях и т.п.

[1]Работа выполнена в рамках проекта № 00–06–80420, поддержанного Российским фондом фундаментальных исследований.

Говоря о *многообразии* проявлений повторности, мы имеем в виду, что симметричные и изоморфные фрагменты, встречающиеся в текстах, также являются составляющими этого многообразия. Действительно, пара симметричных фрагментов образует повтор, если их прочесть в противоположных направлениях. Пара же изоморфных фрагментов образует повтор с точностью до подстановки, осуществляющей переименование элементов алфавита. Многие симметричные и изоморфные фрагменты в текстах являются функционально значимыми. Таковы, например, палиндромно-шпилечные конструкции в генетических текстах, характеризующиеся проявлениями как симметрии, так и изоморфизма.

Степень насыщенности последовательности *повторами равной длины* удобно оценивать с помощью введенной Лемпелем и Зивом числовой характеристики, названной ими *сложностью* последовательности [1]. Различные обобщения этой меры позволяют существенно расширить спектр учитываемых повторов [2]. В частности, мера C_f из [2] реагирует как на проявления повторности (в обычном смысле), так и на наличие симметричных и изоморфных фрагментов.

Целью работы является иллюстрация на примере вычисления меры C_f *единого подхода к обнаружению и учету всевозможных проявлений повторности* в тексте. Предлагаемые конструкции и алгоритмы незначительно уступают оптимальным по трудоемкости, но обладают более простой логикой и удобны для организации вычислений в режиме скользящего окна. Последний используется для обнаружения локальных структурных особенностей в тексте.

1. Определения и обозначения

Будем использовать следующие обозначения: Σ — конечный алфавит, $n = |\Sigma|$ — число элементов алфавита; S — текст (символьная последовательность), составленный из элементов Σ, $N = |S|$ — длина текста S; $S[i]$ — элемент S, стоящий в i-й позиции, $1 \le i \le N$; $S[i:j]$ — символы текста с i-го по j-й включительно, $1 \le i < j \le N$; $S = QT$ — конкатенация (сцепление) последовательностей Q и T, $|S| = |Q| + |T|$;

$S = Q^k$ — k-кратное повторение последовательности Q. Тот же смысл, что и текст, имеют термины "цепочка", "слово", но мы их будем применять по отношению к коротким текстам или фрагментам длинного текста. Слово длины l будем называть l-граммой.

Пусть $f : \Sigma \to \Sigma$ — взаимнооднозначное отображение алфавита Σ на себя ("переименование" элементов алфавита), F — множество всевозможных подстановок (переименований) элементов алфавита, $|F|$ — число подстановок ($|F| = n!$, где $n = |\Sigma|$). Пусть $u = u_1u_2 \dots u_k$ — произвольное слово в алфавите Σ, тогда результатом применения подстановки f к слову u является слово $f(u)$ той же длины, такое что $f(u) = f(u_1)f(u_2) \dots$ $\dots f(u_k)$. Если прочесть слово u в обратном порядке, получим слово $u^R = u_k u_{k-1} \dots u_1$. Результатом последовательного применения преобразований f и R к u является слово $(f(u))^R =$ $= f(u_k)f(u_{k-1}) \dots f(u_1)$. Слова $u = u_1u_2 \dots u_k$ и $v = v_1v_2 \dots v_k$ совпадают ($u = v$), если имеет место $u_i = v_i$ для всех $1 \le i \le k$.

Пусть u и v — два произвольных фрагмента (слова) текста S одинаковой длины. Будем говорить, что пара (u, v) образует *прямой повтор*, если $u = v$; *симметричный повтор*, если $v = u^R$; *прямой f-повтор* (повтор с подстановкой f), если $v = f(u)$; *симметричный f-повтор*, если $v = (f(u))^R$.

ПРИМЕР 1. Пусть $\Sigma = \{A, C, G, T\}$ — алфавит ДНК-последовательностей, $R = \{A, G\}$ и $Y = \{C, T\}$ — подмножества Σ, соответствующие пуринам и пиримидинам, $f_{RY} = \{A \to G,$ $G \to A,\ C \to T,\ T \to C\}$ — подстановка, отображающая пурин в пурин, пиримидин в пиримидин. Тогда описанные выше типы повторов можно проиллюстрировать следующим образом:

		u		v		
1)	...	*GAACTC*	...	*GAACTC*	...	$(v = u)$;
2)	...	*GAACTC*	...	*CTCAAG*	...	$(v = u^R)$;
3)	...	*GAACTC*	...	*AGGTCT*	...	$(v = f(u))$;
4)	...	*GAACTC*	...	*TCTGGA*	...	$(v = f(u)^R)$.

2. Определение меры сложности C_f

Мера C_f, введенная нами в [2], является обобщением известной меры сложности конечной последовательности, предложенной Лемпелем и Зивом (мера LZ [1]). Они оценивают сложность последовательности числом шагов порождающего ее процесса. В их схеме порождения допустимыми являются две операции: генерация нового символа (если он не встречался ранее) и копирование "готового" фрагмента из предыстории, т.е. из уже синтезированной части текста. Из всех возможных схем порождения выбирается минимальная по числу шагов (их может оказаться несколько).

Операция копирования, используемая при определении меры LZ, соответствует прямому повтору в нашем смысле. Целесообразность обобщения меры LZ определяется следующими обстоятельствами: а) во многих языковых системах функционально значимыми являются не только повторы в обычном смысле, но и различные типы f–повторов; б) сложность текста существенно зависит от типа операции копирования; в) использование на конкурентной основе сразу нескольких операций копирования может привести к уменьшению сложности.

Наше обобщение меры LZ состоит: 1) в увеличении числа допустимых операций копирования до $n!$, где $n = |\Sigma|$, что позволяет использовать любые типы f–повторов при порождении текста; 2) в копировании фрагментов не только в прямом, но и в обратном направлении, что эквивалентно расширению спектра допустимых операций до $2 \cdot n!$; 3) в выборе оптимального способа копирования на каждом шаге. Операция генерации символа сохраняется, но в схеме LZ она используется каждый раз при появлении нового символа алфавита, а в нашем подходе она принципиально необходима лишь для порождения первого символа последовательности.

Условимся характеризовать тип операции копирования параметром p $(1 \div 2 \cdot n!)$. Тогда запись $S[i:i+l-1] = S^{(p)}[j:j+l-1]$ будет фиксировать факт наличия повтора типа p, образуемого двумя фрагментами длины l, начинающимися в позициях i и j. Схему порождения последовательности S можно представить в

виде конкатенации фрагментов

$$H(S) = S[1 : i_1]\, S[i_1{+}1 : i_2] \ldots S[i_{k-1}{+}1 : i_k] \ldots S[i_{m-1}{+}1 : N] \quad (1)$$

Здесь $S[i_{k-1}+1 : i_k]$ — фрагмент, порождаемый на k-м шаге с помощью операции копирования $p(k)$, $1 \le p(k) \le 2n!$, $m = C_f(S)$ — число шагов процесса, определяющее *сложность последовательности* S. В дальнейшем будем называть (1) *сложностным разложением* последовательности S. В схеме Лемпеля и Зива на каждом шаге используется одна и та же операция копирования (прямое копирование с тождественной подстановкой). В нашей схеме выбор операции копирования на k-м шаге оптимизируется и призван максимизировать длину $i_k - i_{k-1}$ копируемого фрагмента. Формально

$$i_k - i_{k-1} = \max_{j \le i_{k-1}} \{\max_p l_j^{(p)} \ :$$

$$S[i_{k-1}+1 : i_{k-1} + l_j^{(p)}] = S^{(p)}[j : j + l_j^{(p)} - 1]\}. \quad (2)$$

Значения $j(k)$ и $p(k)$, обеспечивающие максимум (2), будем называть соответственно указателями места ("откуда") и способа ("как") копирования на k-м шаге. Если вместо копирования используется генерация символа, полагаем $j(k) = 0$. С учетом сделанных пояснений k-й компонент сложностного разложения можно записать в виде:

$$S[i_{k-1}+1 : i_k] =$$

$$= \begin{cases} S^{(p(k))}[j(k) : j(k) + l_{j(k)}^{(p(k))} - 1] & \text{при } j(k) \neq 0, \\ S[i_{k-1}+1] & \text{при } j(k) = 0. \end{cases} \quad (3)$$

Заметим, что максимум в (2) может достигаться не на единственной паре $j(k)$ и $p(k)$. Это означает возможность существования альтернативных прототипов и способов копирования для порождаемого фрагмента. Стратегия выбора указателей копирования при наличии альтернативных вариантов зависит от конкретных приложений.

ПРИМЕР 2. Пусть $\Sigma = \{A, C, G, T\}$ — алфавит ДНК-последовательностей. Множество подстановок на нем зададим следующей таблицей из 24 строк:

1:	*ACGT*	7:	*CAGT*	13:	*GACT*	19:	*TACG*
2:	*ACTG*	8:	*CATG*	14:	*GATC*	20:	*TAGC*
3:	*AGCT*	9:	*CGAT*	15:	*GCAT*	21:	*TCAG*
4:	*AGTC*	10:	*CGTA*	16:	*GCTA*	22:	*TCGA*
5:	*ATCG*	11:	*CTAG*	17:	*GTAC*	23:	*TGAC*
6:	*ATGC*	12:	*CTGA*	18:	*GTCA*	24:	*TGCA*

Здесь подстановка с номером p, $1 \le p \le n! = 24$, образуется поэлементным отображением первой строки ($ACGT$) в i-ю. Например, подстановка номер 20 переводит (A, C, G, T) в (T, A, G, C), т.е. A переходит в T ($A \to T$), T в C ($T \to C$), C в A ($C \to A$), а G не меняется ($G \to G$). В соответствии с этим тождественная подстановка, фиксирующая прямые повторы, имеет номер 1, отношение комплементарности ($A \to T$, $T \to A$, $C \to G$, $G \to C$, или, для краткости, $A \leftrightarrow T$, $C \leftrightarrow G$) задается подстановкой со значением $p = 24$ и т.д. Условимся обозначать номер подстановки положительным числом для случая прямого копирования (когда оба фрагмента, составляющие f-повтор, читаются в одном направлении) и отрицательным — для симметричного копирования (когда фрагменты читаются в противоположных направлениях).

В качестве анализируемой последовательности рассмотрим промоторную область одного из человеческих генов (Human aldelase A gene, идентификационный номер HSALDA1 в базе данных EMBL):

$S =$ *GCGGAGGGCG GAGTGGTGCC TTTAAAAGGC*
CGGGCGCCGC CTTCCGCCTG CCCGCCTCCT
GCGCCGCCCC TTCCGAGGCT AAATCGGCTG
CGTTCCTCTC

Сложностное разложение по мере C_f для последовательности S представлено ниже. В скобках справа от каждого компонента разложения указана тройка чисел $(k, j(k), p(k))$, где k — номер компонента (приводится для удобства отсылок), $j(k)$ — указатель места копирования для k-го компонента, $p(k)$ — указатель способа копирования (номер подстановки со знаком "+" или "−"). Имеем:

$H(S) =$ *G*(1, 0, 0) *CG* (2, 1, 3) *GAGG* (3, 1, 7) *GCGGAG* (4, 1, 1) *TGG* (5, 10, -12) *TGC* (6, 12, 12) *CTTTA* (7, 5, 8) *AAAGGC*

(8, 18, -24) $CGGG$ (9, 25, -9) $CGCCGC$ (10, 13, 4) $CTTCCG$ (11, 27, -23) $CCTG$ (12, 17, -2) $CCCGCCTC$ (13, 6, 24) $CTGC$ (14, 48, 1) $GCCGCC$ (15, 55, -2) $CCTTCCG$ (16, 40, 1) $AGGCT$ (17, 72, 24) $AAATC$ (18, 49, -11) $GGCTG$ (19, 58, 5) $CGTTCC$ (20, 17, 6) $TCTC$ (21, 96, 6).

Нетрудно видеть, что параметр k меняется от 1 до 21, т.е. значение сложности $C_f = 21$. Первый компонент (G) генерируется $(j(1) = 0)$, второй (CG) копируется с фрагмента GC, начинающегося в первой позиции, с помощью подстановки №3 ($G \leftrightarrow C$, A и T не меняются). Важно отметить, что копирование всегда начинается с элементов, синтезированных на предыдущих шагах, но закончиться может на элементах, синтезированных на текущем шаге, что и имеет место при формировании второго компонента. Иными словами, синтезируемый компонент может частично накладываться на прототип, сигнализируя о возможной периодичности. Более ярко этот эффект прослеживается на компонентах №№17 и 21. В первом случае имеет место прямой комплементарный тандемный повтор $(TCCG)(AGGC)T\ldots$, во втором фрагмент $(TC)^2$ копируется с фрагмента $(CT)^2$, сдвинутого относительно него на символ влево, с помощью подстановки, переводящей C в T, а T в C.

Из других компонентов разложения, представляющих интерес, отметим №4, фиксирующий тандемный повтор $(GCGGAG)G(GCGGAG)$, расположенный в начале S, компонент №8, фиксирующий комплементарный палиндром $\overrightarrow{GCCTTT}\ A\ \overleftarrow{AAAGGC}$, начинающийся в позиции 18, и компонент №15, демонстрирующий возможность получения тандемного повтора $(GCC)^2$ путем симметричного копирования с $(CCT)^2$ при помощи подстановки №2, переводящей G в T, T в G и сохраняющей A и C.

Ниже рассмотрен алгоритм вычисления меры C_f, из которого в частном случае (при фиксированной подстановке) следует способ вычисления меры Лемпеля и Зива. Основные вычислительные проблемы связаны:

а) с отысканием максимального прототипа для копирования в предыстории (т.е. в уже синтезированной части последовательности) при фиксированной подстановке;

б) организацией симметричного копирования;

в) поиском подстановки, оптимизирующей длину копируемого компонента на каждом шаге (фактически речь идет о способе обхода факториального в зависимости от размера алфавита n перебора, связанного с определением максимального изоморфного прототипа для копирования);

г) организацией режима обработки в скользящем окне.

3. Вычисление сложности при фиксированной подстановке (мера LZ)

Рассмотрим для простоты случай прямого копирования с тождественной подстановкой. Пусть $S[1 : N]$ — исходный текст, а $S[1 : i_{k-1}]$ — его префиксная часть ($i_{k-1} < N$), синтезированная за $(k-1)$ шагов (каждый шаг соответствует получению очередного компонента сложностного разложения). Для вычисления очередного (k-го) компонента требуется найти цепочку максимальной длины, начинающуюся с элемента $S[i_{k-1}+1]$ и представленную (или берущую начало) в $S[1 : i_{k-1}]$. Это задача поиска образца со "свободным" правым концом в тексте, длина которого, в свою очередь, увеличивается после каждого акта копирования. Для представления текста будем использовать структуру данных типа "trie", описанную в [3].

Пусть L — целочисленный параметр, сравнимый с длиной максимального (случайного) повтора в тексте. Скользящим окном размера L выделим полную совокупность L-грамм $S[i : i+L-1]$, $1 \le i \le i_{k-1}$, и представим их в виде лексикографического дерева $Tr(i_{k-1})$ со склеенными общими префиксными частями. Ребра дерева пометим символами из упакованных цепочек. Последовательность реберных меток на пути из корня дерева в заданную вершину уровня l определяет конкретную l-грамму текста, $1 \le l \le L$.

С вершинами дерева связываем позиционную информацию о вхождениях заданной l-граммы в текст. Для внутренних вершин дерева ($l < L$) достаточно ограничиться информацией о по-

следнем вхождении l–граммы в текст. В листьях дерева ($l = L$) содержится информация о всех вхождениях L–граммы в текст.

Заметим, что в дерево $Tr(i_{k-1})$ наряду с L–граммами, расположенными в уже синтезированной части текста, включены и такие, которые частично выходят за пределы синтезированного фрагмента (их всего $L-1$). Это позволяет реализовать режим копирования с наложением прототипа на вновь синтезируемый фрагмент, что удобно для выявления периодичностей (см. пример 2).

Итак, будем считать, что к началу выполнения k–го шага дерево $Tr(i_{k-1})$ построено. Тогда поиск максимального по длине прототипа для синтезируемого на k–м шаге компонента сложностного разложения сводится к прохождению дерева от корня к листьям по цепочке $S[i_{k-1}+1]S[i_{k-1}+2]\ldots$ При этом возможны два случая:

1) поиск обрывается через D шагов ($D < L$), т.е. из внутренней вершины, достигнутой на уровне D, не существует перехода (ребра помеченного символом $S[i_{k-1}+D+1]$) на уровень $(D+1)$. Это означает, что длина максимального прототипа равна D;

2) поиск заканчивается в листе дерева z, с которым ассоциирован список позиций $(n_1, n_2, \ldots, n_{r(z)})$ всех вхождений цепочки $S[i_{k-1}+1 : i_{k-1}+L]$ в текст $S[1 : i_{k-1}+L-1]$. Это означает, что $D \geq L$. Чтобы определить, какой случай имеет место (равенство или неравенство), расширяем по тексту вправо каждую из L–грамм $S[n_i : n_i+L-1]$, $1 \leq i \leq r(z)$, до тех пор, пока не обнаружится рассогласование с продолжением цепочки $S[i_{k-1}+1 : i_{k-1}+L]$. Выявляем

$$D^* = \max_{1 \leq r \leq r(z)} \{(D_r : S[n_r : n_r+L-1+D_r] = \\ = S[i_{k-1}+1 : i_{k-1}+L+D_r]\}.$$

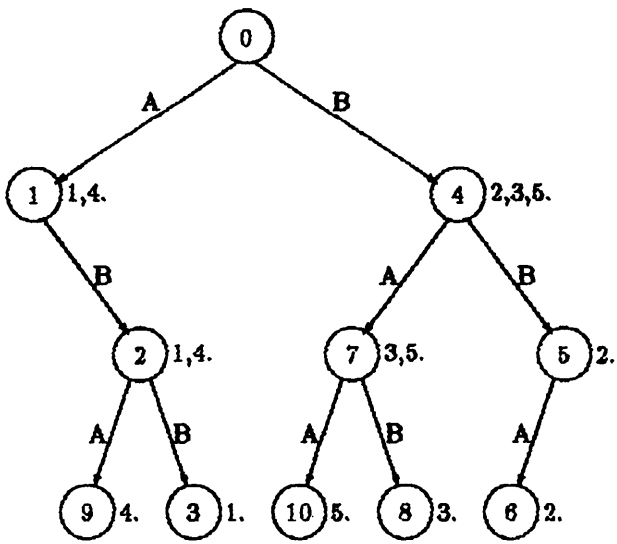
Полагаем $D = L + D^*$.

В обоих случаях (1 и 2) фрагмент $S[i_{k-1}+1 : i_{k-1}+D]$ фиксируется как очередной (k–й) компонент сложностного разложения. Перед началом поиска следующего ($(k+1)$–го) компонента, начинающегося с позиции $(i_{k-1}+D+1)$, достраиваем дерево $Tr(i_{k-1})$ до $Tr(i_k)$, добавляя в него L–граммы $S[i_{k-1}+1 : i_{k-1}+L]$, $S[i_{k-1}+2 : i_{k-1}+L+1], \ldots, S[i_{k-1}+D : i_{k-1}+L+D-1]$.

ПРИМЕР 3. Пусть $\Sigma = \{A, B\}$, $S = ABBABAABBAABA$ и требуется построить сложностное разложение последовательности S. Для иллюстрации возьмем значение $L = 3$. Предположим, что выполнены 4 шага алгоритма, где каждый шаг предполагает поиск очередного компонента разложения и достройку дерева. После 4 шагов синтезированы 5 элементов последовательности:

$$H(S[1:5]) = A \cdot B \cdot B \cdot AB.$$

Дерево $Tr(5)$ имеет вид:



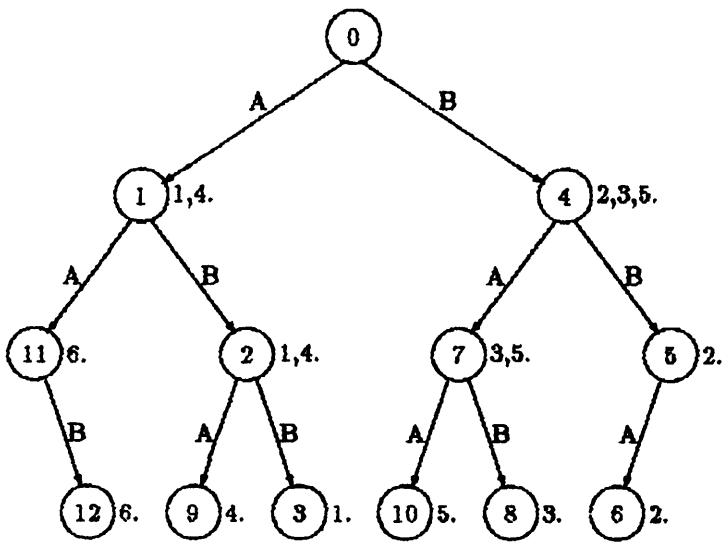
Здесь каждой вершине соответствует своя цепочка символов, маркирующая путь из корня дерева в данную вершину. Вхождения каждой такой цепочки в текст (номера начальных позиций) указаны рядом с вершиной (в принципе, для внутренних вершин можно ограничиться указанием лишь последнего вхождения цепочки в текст). Вершины для наглядности пронумерованы в порядке их построения, хотя при реализации удобно использовать другую нумерацию, при которой номера внутренних вершин предшествуют номерам листьев [4].

На шаге 5:

а) для несинтезированной еще части последовательности ($S[6:N] = AABB\ldots$) ищем по дереву $Tr(5)$ максимальный префикс, представленный (или начинающийся) в $S[1:5]$. Поиск заканчивается в вершине (1) первого уровня, т.е. длина очередного компонента разложения равна 1 (символ A присутствует в $S[1:5]$, а цепочка AA — уже нет). Таким образом

$$H(S[1:6]) = A \cdot B \cdot B \cdot AB \cdot A;$$

б) достраиваем дерево $Tr(5)$, добавляя 3-грамму $S[6:8] = AAB$. Получаем дерево $Tr(6)$:



На шаге 6:

а) ищем по дереву $Tr(6)$ максимальный префикс последовательности $S[7:N] = ABBA\ldots$, представленный (или начинающийся) в $S[1:6]$. Поиск заканчивается в листе (3), т.е. цепочка ABB (начальная в $S[7:N]$) фигурирует в $S[1:6]$. Позиционная информация, связанная с вершиной (3), говорит о том, что цепочка встречается в $S[1:6]$ один раз и начинается в позиции 1

Продолжаем сравнение совпавших фрагментов $S[7:9]$ и $S[1:3]$ по тексту, расширяя их вправо. Выявляем, что $S[7:10]$ и $S[1:4]$ еще совпадают, а $S[7:11]$ и $S[1:5]$ уже отличаются последним элементом. Таким образом, длина очередного компонента разложения равна 4, а само разложение имеет вид:

$$H(S[1:10]) = A \cdot B \cdot B \cdot AB \cdot A \cdot ABBA;$$

б) достраиваем дерево $Tr(6)$ до $Tr(10)$, добавляя в него четыре 3-граммы, начинающиеся в позициях 7, 8, 9, 10 (это ABB, BBA, BAA, AAB). Новых вершин и ребер при этом не образуется, изменяется лишь позиционная информация, связанная с вершинами дерева.

Трудоемкость описанного алгоритма имеет порядок $L \cdot N$ в предположении, что поиск нужного ветвления в каждой вершине дерева требует константного времени, т.е. не зависит от размера алфавита, а аномально длинные повторы, длина которых существенно превышает L, встречаются не слишком часто. Ориентиром для выбора L может служить оценка длины максимального повтора в случайной последовательности той же длины, что и исследуемый текст, и с тем же частотным составом [5]:

$$L_{\max} \sim \frac{2 \ln N}{\left| \ln \sum_{r=1}^{n} p_r^2 \right|},$$

где N — длина текста, n — размер алфавита, p_r $(1 \le r \le n)$ — вероятности появления элементов алфавита. Уменьшение значения L по сравнению с рекомендуемым приводит к снижению затрат памяти, но увеличению трудоемкости, и наоборот. Для небольших алфавитов (в частности, для ДНК-последовательностей) выбор L в диапазоне от 8 до 10 уже позволяет обрабатывать практически в реальном масштабе времени тексты достаточно большой длины ($N \sim 10^4 \div 10^5$).

ЗАМЕЧАНИЕ 1. Вычисление сложностного разложения можно было бы реализовать с помощью конструкции типа "суффиксное дерево" [6] с гарантированно линейной (в зависимости от N) трудоемкостью. Однако из-за сложной логики соответствующий алгоритм будет иметь большую мультипликативную константу

перед N в оценке трудоемкости. Как следствие, на практике результаты будут сопоставимыми с приведенным выше алгоритмом в широком диапазоне значений N.

ЗАМЕЧАНИЕ 2. Для наглядности мы рассматривали вариант копирования с помощью тождественной подстановки. В случае произвольной подстановки f алгоритм не изменится, лишь на вход построенного дерева $Tr(i_k)$ будет подаваться цепочка $f(S[i_k+1])\ f(S[i_k+2])$ и т.д.

ЗАМЕЧАНИЕ 3. Дерево $Tr(i_k)$ для текста $S[1:i_k]$ можно использовать и для организации симметричного копирования. Поскольку параметр D (длина максимального префикса последовательности $S[i_k+1:N]$, для которого существует симметричный прототип в $S[1:i_k]$) нам неизвестен, его можно определить итеративно, последовательно осуществляя в дереве $Tr(i_k)$ поиск цепочек $S[i_k+r]S[i_k+r-1]\ldots S[i_k+1]$ $(r=1,2,\ldots)$. Значение r, при котором поиск впервые окажется неудачным, равно $D+1$. Для уменьшения числа итераций поиск можно начинать со значения r ближайшего к средней длине компонента сложностного разложения в $H(S[1:i_k])$ и в зависимости от результата увеличивать или уменьшать последующие значения r на 1. Описанной процедурой можно воспользоваться при дефиците оперативной памяти.

Другое решение, не требующее итераций, связано с построением для текста $S[1:i_k]$ еще одного дерева $Tr^R(i_k)$ по цепочкам длины L, прочитанным справа-налево. Максимальный симметричный прототип будет обнаружен, если подать на вход $Tr^R(i_k)$ цепочку $S[i_k+1]S[i_k+2]\ldots$

4. Алгоритм вычисления меры сложности с оптимизацией выбора подстановки на каждом шаге (мера C_f)

В определении меры C_f фигурируют все $n!$ возможных подстановок (n — размер алфавита). Поэтому ключевой становится проблема факториального перебора для больших алфавитов. Вводимая ниже конструкция играет важную роль в решении этой проблемы.

Пусть u — произвольная цепочка длины L над алфавитом Σ. Информация о наличии и расположении повторяющихся эле-

ментов в u содержится в наддиагональной матрице $B(u)$ порядка $L \times L$ с элементами $b_{ik} = 0$ при $k \le i$ и

$$b_{ik} = \begin{cases} 1, & \text{если } u[i] = u[k], \\ 0, & \text{если } u[i] \neq u[k], \end{cases} \qquad k > i.$$

Будем называть $B(u)$ *структурной матрицей* цепочки u. Эту матрицу можно преобразовать в двоичный (структурный) вектор $\Phi(u)$ длины $L(L-1)/2$, путем конкатенации (слева-направо) наддиагональных столбцов.

ПРИМЕР 4. Пусть $\Sigma = \{a, b, c\}$, $u = abaccaba$. Тогда

$B(u) =$

a	b	a	c	c	a	b	a	
$\equiv$	0	1	0	0	1	0	1	a
	$\equiv$	0	0	0	0	1	0	b
		$\equiv$	0	0	1	0	1	a
			$\equiv$	1	0	0	0	c
				$\equiv$	0	0	0	c
					$\equiv$	0	1	a
						$\equiv$	0	b
							$\equiv$	a

и соответственно $\Phi(u) = \underline{0}\ \underline{10}\ \underline{000}\ \underline{0001}\ \underline{10100}\ \underline{010000}\ \underline{1010010}$ (подчеркнуты цепочки, соответствующие отдельным столбцам $B(u)$). Запись матрицы по столбцам удобна при удлинении цепочки. Так, если $u' = ua$, где $a \in \Sigma$, то $\Phi(u') = \Phi(u)\phi(u, a)$, где $\phi(u, a)$ — последний столбец матрицы $B(u')$ (результат сравнения добавленного символа "a" с символами цепочки u).

На принципиальную возможность обхода факториального перебора при вычислении меры C_f указывает следующее

УТВЕРЖДЕНИЕ. *Пусть x — произвольный компонент длины L в сложностном разложении последовательности S по мере C_f, y — его прототип ($|y| = |x|$), f — подстановка, переводящая y в x. Тогда $B(y) = B(x)$ (соответственно $\Phi(y) = \Phi(x)$).*

Справедливость данного утверждения следует из того, что подстановка f переводит одинаковые элементы цепочки y в одинаковые же элементы цепочки x и оставляет их на тех же самых местах. В то же время, поскольку отображение является

взаимнооднозначным, разные элементы цепочки y переходят в разные элементы цепочки x.

ПРИМЕР 5. Пусть $\Sigma = \{A, C, G, T\}$, $x = AGCTTG$, $y = TAGCCA$, $f = (A \to G,\ G \to C,\ C \to T,\ T \to A)$. Нетрудно видеть, что $f(y) = x$. Имеем:

	A	G	C	T	T	G
A	≡	0	0	0	0	0
G		≡	0	0	0	1
C			≡	0	0	0
T				≡	1	0
T					≡	0
G						≡

$= B(x) = B(y) =$

T	A	G	C	C	A	
≡	0	0	0	0	0	T
	≡	0	0	0	1	A
		≡	0	0	0	G
			≡	1	0	C
				≡	0	C
					≡	A

Из сформулированного утверждения следует, что структурный вектор является инвариантом, связывающим копируемый компонент с его прототипом, т.е. он не зависит от используемой подстановки. На этом и основана идея обхода факториального перебора: нужно оперировать не с цепочками исходного текста, а с их структурными векторами. Тогда изоморфные фрагменты будут обнаружены как повторяющиеся двоичные цепочки. Аналогичная идея, но с несколько другой формой инварианта, была использована в [7] для выявления плагиатов в текстах программ.

Алгоритм вычисления меры C_f повторяет описанный выше алгоритм вычисления меры LZ с точностью до перехода к структурным векторам. Отыскание максимального f-прототипа на k–м шаге для текущей цепочки $S[i_{k-1}+1]S[i_{k-1}+2]\ldots$ сводится:

а) к замене всех L–символьных цепочек $S[i : i+L-1]$, $1 \le i \le i_{k-1}$, их структурными векторами $\Phi(S[i : i+L-1])$;

б) построению дерева по множеству структурных векторов;

в) поиску по этому дереву максимального структурного вектора $\Phi(S[i_{k-1}+1]S[i_{k-1}+2]\ldots)$, получаемого путем последовательного наращивания его длины при добавлении новых символов (для этого и используется запись матрицы B по столбцам). Здесь также возможны ситуации, когда поиск закончится во внутренней вершине дерева, либо будет достигнута конечная вершина и потребуется расширение по тексту. Следует заметить, что внутренняя вершина двоичного дерева, на которой заканчивается поиск, может располагаться лишь на определенных уровнях,

поскольку длины структурных векторов квантованы определенным образом (1, 3, 6, 10, 15 и т.д.);

г) восстановлению подстановки, используемой на k-м шаге, путем сопоставления найденного прототипа (в исходном алфавите) с текущей цепочкой.

Если наряду с прямым копированием допускается копирование в обратном направлении, реализуется (на уровне структурных векторов) схема, описанная в замечании 3 к разделу 3.

Поскольку длина структурных векторов, по которым строится дерево, имеет порядок L^2, трудоемкость вычисления меры C_f составляет $O(L^2 \cdot N)$.

5. Вычисление меры C_f в режиме скользящего окна.

Схема обработки, при которой сложностное разложение проводится не для всей последовательности, а лишь для фрагментов фиксированной длины W, выделяемых "окном", движущимся вдоль последовательности, называется режимом скользящего окна. Он позволяет выделять локальные структурные особенности текста, которые подчас оказываются замаскированными в разложении полного текста (случай $W = N$). В этом смысле параметр W определяет разрешающую способность метода.

Последовательность значений

$$CP(S, W) = c_1 c_2 \dots c_{N-W+1}, \tag{4}$$

где c_i — сложность фрагмента $S[i : i+W-1]$, $1 \le i \le N-W+1$, (в смысле любого из рассмотренных выше определений) назовем *сложностным профилем* текста S [8]. Важными характеристиками текста, получаемыми из (4), являются:

$$C_{\min}(S, W) = \min_i c_i;$$

$$C_{\max}(S, W) = \max_i c_i;$$

$$\overline{C}(S, W) = \frac{\sum_i c_i}{N-W+1};$$

$$\sigma^2(S,W) = \frac{\sum_i (c_i - \overline{C})^2}{N - W}$$

и ряд других. Эти характеристики, вычисляемые для разных значений W, могут использоваться для классификации и сравнения текстов (в том числе разной длины).

Ниже описан быстрый алгоритм вычисления сложностного профиля текста по мере C_f. Нетрудно видеть, что прямой (неоптимизированный) метод вычисления сложностного профиля будет иметь трудоемкость $O(L^2 \cdot W \cdot N)$ при $W \ll N$. Наша цель — избавиться от множителя W в оценке трудоемкости. Он становится критичным при увеличении размера окна анализа. Отметим, что алгоритм вычисления сложностного профиля для меры LZ с единственной (причем тождественной) подстановкой был описан в [9]. Он отличается от нашего более сложным конструктивным решением: в [9] для представления текста использовалось дерево префикс–идентификаторов и, как следствие, профиль строился справа–налево, т.е. от значения c_{N-W+1} до c_1. Эта схема требует, чтобы текст заранее был представлен полностью, что неудобно для приложений, в которых данные поступают и должны обрабатываться в реальном масштабе времени. Алгоритм вычисления сложностного профиля для меры C_f публикуется впервые.

Мы будем использовать факт зацепленности соседних окон, который позволяет построить рекуррентную схему пересчета сложности при движении окна вдоль текста. Первые W символов текста обрабатываются в соответствии с описанным в предыдущем разделе алгоритмом. Далее начинает работать схема пересчета.

Пусть на i–м шаге получено сложностное разложение фрагмента текста $S[i : i + W - 1]$ и построено дерево L–грамм для этого фрагмента (обозначим его $Tr(i)$). Как и ранее (см. раздел 3) будем предполагать, что в $Tr(i)$ включены также L–граммы, которые берут начало в $S[i : i + W - 1]$, а закончиться могут за пределами этого фрагмента. Более того, для корректной организации пересчета нам потребуется указать для каждой вершины

дерева полный список всех вхождений в $S[i : i+W-1]$ l-граммы, $1 \le l \le L$, соответствующей данной вершине. Это не потребует слишком больших затрат памяти, поскольку размер окна, как правило, много меньше длины текста.

При переходе к следующему фрагменту $S[i+1 : i+W]$ проводится коррекция $Tr(i)$, переводящая его в $Tr(i+1)$. Для этого удаляется L-грамма, начинающаяся с элемента $S[i]$, и добавляется L-грамма, начинающаяся с элемента $S[i+W]$. Далее с помощью дерева $Tr(i+1)$ корректируются те компоненты разложения фрагмента $S[i : i+W-1]$, которые меняются при удалении элемента слева и добавлении элемента справа. Это первый и последний компоненты, а также те внутренние, указатель копирования для которых был равен i. Корректировка последних может повлечь за собой изменение следующих за ними компонентов вплоть до достижения точки синхронизации — позиции текста, начиная с которой разложения в обоих окнах будут вновь совпадать. Опишем эти шаги более подробно.

Корректировка дерева $Tr(i)$ включает *удаление* из него цепочки $S[i : i+L-1]$ и *добавление* цепочки $S[i+W : i+W+L-1]$. Добавление делается по той же схеме, по которой проводилась достройка дерева после получения очередного компонента сложностного разложения (см. раздел 3). Рассмотрим порядок действий при удалении цепочки. Следуя символам цепочки $S[i : i+L-1]$, проходим из корня дерева по интересующему нас пути и анализируем в каждой вершине позиционную информацию. Если список вхождений в $S[i : i+W-1]$ соответствующей l-граммы, $1 \le l \le L$, содержит более, чем одну запись, корректируем его, убирая первую запись (значение i). Если список содержит всего одну запись (это может быть только значение i), удаляем данную и следующие за ней вершины вместе с заходящими в них ребрами. Номера удаленных вершин становятся пригодными для дальнейшего использования и заносятся в специальный список ("чистка мусора").

Если дерево $Tr(i)$ построено по структурным векторам $\Phi(S[k : k+L-1])$, $i \le k \le i+W-1$, то удаление элемента $S[i]$ затрагивает двоичную цепочку $\Phi(S[i : i+L-1]$. Корректировка проводится по схеме, описанной выше. Отличие состо-

ит в том, что позиционная информация определена не для всех вершин двоичного дерева, а только для тех, глубина которых соответствует позициям, занимаемым в структурных векторах последними элементами столбцов матрицы B.

В случае использования дерева $Tr^R(i)$ для организации симметричного копирования удаление элемента $S[i]$ затрагивает L цепочек: $S[i+L-1:i] \stackrel{\text{def}}{=} S[i+L-1]S[i+L-2]\ldots S[i]$, $S[i+L-2:i]$, $S[i:i] = S[i]$. При их обработке доходим по каждой цепочке до элемента $S[i]$ и осуществляем коррекцию списков, связанных с проходимыми вершинами, или удаление самих вершин по схеме, описанной выше. В случае структурных векторов действуем аналогично.

Рассмотрим теперь *корректировку компонентов сложностного разложения* при сдвиге окна анализа (переход от $S[i:i+W-1]$ к $S[i+1:i+W]$). Предполагаем, что дерево $Tr(i+1)$ уже получено, а в качестве начального приближения положим $c_{i+1} = c_i$. Корректировке подвергаются первый и последний компоненты разложения фрагмента $S[i:i+W-1]$, т.е. места, где символ исчезает и добавляется, а также внутренние компоненты с указателем копирования равным i и, возможно, незначительное число следующих за ними компонентов.

Первый компонент в любом окне получается с помощью операции генерации символа (копировать еще не с чего), длина его равна единице. Этот компонент удаляется, значение c_{i+1} уменьшается на единицу.

Последний компонент, если он не затронут перестройкой одного из внутренних компонентов (см. ниже), может удлиниться за счет добавления символа $S[i+W]$ или остаться прежним. Удлинение имеет место, когда символ, непосредственно следующий за одним из возможных прототипов (для последнего компонента), совпадает с $S[i+W]$. В этом случае число компонентов не меняется, значение c_{i+1} не корректируется, но может измениться указатель копирования. Если же последний компонент не удлиняется при добавлении символа справа, то $S[i+W]$ образует новый компонент и значение c_{i+1} увеличивается на 1.

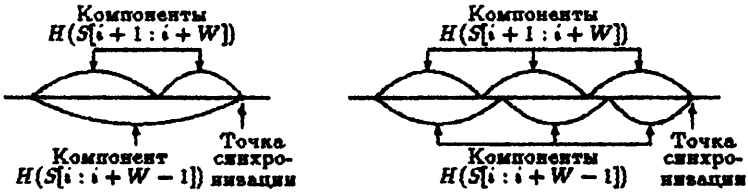
Для быстрого выявления компонентов, указатель копирования которых равен i, используется вспомогательная структура данных. А именно, по ходу вычисления сложностного разложения для каждого фрагмента происходит сортировка компонентов по значениям указателей копирования (их разнообразие не превышает W), т.е. для фрагмента $S[i:i+W-1]$ формируются списки компонентов (точнее, отсылок на них) со значениями указателей копирования $j=i,i+1,\ldots,i+W-2$. Этот процесс также организован рекуррентно: для первого фрагмента $S[1:W]$ списки вычисляются, а дальше корректируются при каждом сдвиге окна параллельно с корректировкой компонентов.

Итак, пусть коррекции подвергается компонент из окна $S[i:i+W-1]$, начинающийся в позиции $i_1 > i$ и имеющий указатель копирования i. По деревьям $Tr(i+1)$ и $Tr^R(i+1)$ определяем длину компонента в окне $S[i+1:i+W]$. Для этого идем по каждому дереву до тех пор, пока в списке позиций, связанном с проходящей вершиной, еще имеются значения, не превышающие i_1. Это соответствует схеме копирования из предыстории.

Длина "нового" компонента (если он не последний) может либо уменьшиться, либо остаться прежней. Она не может увеличиться, поскольку при удалении элемента $S[i]$ новые возможности для копирования не появляются. Если длина компонента не изменилась, то меняется лишь указатель копирования. Рассинхронизации разложений $H(S[i:i+W-1])$ и $H(S[i+1:i+W])$ в данном месте не происходит. Дальше разложения будут совпадать вплоть до следующего компонента с указателем копирования равным i или до последнего компонента.

Если длина компонента, начинающегося в позиции i_1, уменьшилась, возникает рассинхронизация двух разложений. В этом случае очередные компоненты разложения $H(S[i+1:i+W])$ вычисляются путем прохождения деревьев $Tr(i+1)$ и $Tr^R(i+1)$ (а при достижении листа — непосредственным расширением по тексту). Процесс продолжается вплоть до достижения точки синхронизации. На практике она обычно достигается за 2–3 шага [9]. Описанная выше стратегия не меняется в зависимости от того, построены ли деревья $Tr(i+1)$ и $Tr^R(i+1)$ по

цепочкам текста S или их структурным векторам. На рисунке приведена иллюстрация рассинхронизации двух разложений: в случае (а) точка синхронизации достигается через два шага, в случае (б) — через три шага.



а) б)

Нетрудно видеть, что трудоемкость предложенного алгоритма коррекции деревьев при движении окна не зависит от размера окна W и имеет порядок LN. Трудоемкость алгоритма коррекции сложностных разложений зависит от числа перестраиваемых в окне компонентов. В наихудшем случае эта величина имеет порядок $\log W$ (перестраиваются все компоненты, а среднее число их логарифмическим образом зависит от длины текста [1], т.е. размера окна в нашем случае). Следовательно, в наихудшем случае суммарная оценка трудоемкости имеет порядок $L \cdot \log W \cdot N$. Вариант маленького окна ($W \sim 20$–30) реально отражает эту ситуацию. Однако при достаточно больших окнах доля перестраиваемых компонентов мала и практически не зависит от размера окна. Этому способствует и использование в качестве указателя копирования (при наличии альтернативных вариантов) последнего из допустимых значений. Таким образом при достаточно больших окнах оценка трудоемкости в среднем близка по порядку к LN (а для структурных векторов — к L^2N), что с учетом незначительных затрат памяти делает сложностной анализ в режиме скользящего окна эффективным инструментом выявления локальных структурных закономерностей в длинных текстах.

Заключение

Различные языковые системы характеризуются многообразными проявлениями повторности. Наряду с повторами в обычном смысле функционально значимыми часто оказываются симметричные повторы (палиндромы) и изоморфные фрагменты (участки текста, совпадающие друг с другом с точностью до переименования элементов алфавита). Степень насыщенности текста указанными типами повторов удобно характеризовать с помощью введенной авторами числовой характеристики (мера C_f), обобщающей понятие сложности конечной символьной последовательности, предложенное Лемпелем и Зивом.

Рассмотрены алгоритмические аспекты вычисления меры C_f для полной последовательности и ее отдельных фрагментов, зацепленных друг с другом (режим скользящего окна). Показано, что существенное расширение спектра учитываемых повторов не приводит к значительному повышению трудоемкости, которая имеет порядок L^2N, где L — величина, сопоставимая с длиной максимального повтора в случайных аналогах исследуемого текста, сохраняющих его частотный состав. Предлагаемые конструкции и алгоритмы обладают простой логикой и удобны для организации вычислений в режиме реального времени, т.е. по мере поступления очередных символов.

Литература

1. LEMPEL A., ZIV J. On the complexity of finite sequences // IEEE Trans. on Inf. Th. – 1976. – Vol. IT – 22, №1.– P. 75–81.

2. ГУСЕВ В.Д., НЕМЫТИКОВА Л.А. Сложностные характеристики генетических текстов // Труды 4-й Всероссийской конф. "Распознавание образов и анализ изображений". – 1998. – Ч.1. – Новосибирск. – С. 83–87.

3. КНУТ Д. Искусство программирования для ЭВМ. Т.3. – М.: Мир, 1978.

4. AOL Junichi, YAMAMOTO Yoneo, SHIMADA Yoneo. A method for improving string pattern matching machines // IEEE Trans. on Software Engineering. – 1984. – Vol. SE–10, №1. – P. 116–120.

5. ЗУБКОВ А.М., МИХАЙЛОВ В.Г. Предельные распределения случайных величин, связанных с длинными повторениями в последовательности независимых испытаний // Теория вероятностей и ее применения. – 1974. – Т. XIX, №1. – С. 173–181.

6. McCREIGHT E.M. A space–economial suffix tree construction algorithm // J. Assoc. Comput. Mach. – 1976. – Vol. 32, №2 . – P. 262–272.

7. BAKER B.S. Parameterized pattern matching: algorithms and applications // Journal of computer and system sciences. – 1996. – Vol. 52.– P. 28–42.

8. ГУСЕВ В.Д. Сложностные профили символьных последовательностей // Методы обработки символьных последовательностей и сигналов. – Новосибирск, 1989. – Вып. 132: Вычислительные системы. – С. 35–63.

9. ЧУПАХИНА О.М. Алгоритм построения сложностного профиля символьной последовательности // Методы обработки символьных последовательностей и сигналов. – Новосибирск, 1989. – Вып. 132: Вычислительные системы. – С. 64–91.

Поступила в редакцию
31 августа 2001 года