ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И ЭКСПЕРТНЫЕ СИСТЕМЫ
(Вычислительные системы)
1997 год Выпуск 160

УДК 519.688:519.713.1

ИСПОЛЬЗОВАНИЕ НЕДЕТЕРМИНИРОВАННЫХ КОНЕЧНЫХ АВТОМАТОВ ДЛЯ УСКОРЕНИЯ ПОИСКА В ТЕКСТОВЫХ БАЗАХ ДАННЫХ[1]

Л.А. Немытикова

Введение

Целью работы является разработка эффективных и легко реализуемых на персональных компьютерах среднего класса методов поиска в текстовых базах данных по *групповому частично специфицированному запросу* с *большим* числом образцов. Частично специфицированный образец — это цепочка символов, элементы которой заданы с точностью до принадлежности к определенным подмножествам исходного алфавита. Такие образцы естественным образом возникают в задачах классификации при построении консенсусной (обобщающей) последовательности для группы близких текстов.

В литературе в основном рассматривается случай единичного частично специфицированного образца, отдельные элементы которого заданы точно, а все другие несущественны (элементы типа "X") [1,2]. Указанная постановка обобщается в двух направлениях: а) допускается возможность неточного (приближенного) соответствия

[1]Работа выполнена в рамках проекта, поддержанного грантом Госкомитета по высшему образованию РФ.

между образцом и фрагментом анализируемого текста [3]; б) предполагается, что вместо одного образца может быть предъявлена одновременно группа образцов [4]. Настоящая работа имеет отношение к обеим постановкам, но основное внимание будет уделено задаче "б".

В [5] рассмотрен *общий* случай группового частично специфицированного запроса. Среди элементов образца могут быть точно заданные символы, подмножества исходного алфавита и неопределенные символы ("X" — любой символ алфавита). Для поиска по такому запросу использовался детерминированный конечный автомат Ахо–Корасик [4], который строился не по полным образцам, а по относительно коротким их фрагментам — частично специфицированным (в общем случае) ядрам, выбранным оптимальным образом. Поскольку автомат Ахо–Корасик предназначен для поиска по *детерминированному* групповому запросу, осуществлялась замена частично специфицированных в общем случае ядер эквивалентно представляющими их (в смысле полноты поиска) множествами точно заданных ядер (процедура "развертки"). В общем случае это приводило к многократному увеличению числа константных "ядерных" образцов на входе, особенно в ситуациях, когда частично специфицированное ядро содержало элементы типа "X".

В связи с этим в данной работе предлагается аналог автомата Ахо–Корасик в предположении, что элемент "X" добавлен к исходному алфавиту. Тогда на этапе построения автомата данный элемент рассматривается наравне с константными символами, а на этапе поиска ребра, помеченные символом X, образуют альтернативные пути. Это эквивалентно переходу от детерминированного конечного автомата к недетерминированному. Время поиска в таком автомате несколько возрастает, но объем памяти существенно сокращается. Таким образом предлагаемые конструкции позволяют осуществлять перекачку памяти в быстродействие (и наоборот), что существенно при реализации поиска на персональных ЭВМ.

1. Постановка задачи и обозначения

Пусть Σ — исходный (конечный) алфавит, $s = |\Sigma|$ — размер алфавита Σ; T — текст, составленный из элементов Σ; $T[i]$ — i-й символ текста T; $N = |T|$ — длина текста T; $D = \{\delta_1, \delta_2, \ldots, \delta_d\}$ — множество подмножеств алфавита Σ, фигурирующих в частично специфицированном запросе ($|\delta_k| \geq 2$, $1 \leq k \leq d$); $p = b_1 b_2 \ldots b_m$, $b_i \in \Sigma \cup D$, — частично специфицированный образец; $m = |p|$ — длина образца; $P = \{p_1, p_2, \ldots, p_n\}$ — групповой частично специфицированный запрос; n — число образцов в запросе.

Задача поиска по групповому частично специфицированному запросу P состоит в том, чтобы в заданном тексте T найти все фрагменты, согласованные хотя бы с одним образцом из P. Фрагмент $a_1 a_2 \ldots a_m$, $a_i \in \Sigma$, $1 \leq i \leq m$, будем считать согласованным с образцом $p = b_1 b_2 \ldots b_m$, $b_i \in \Sigma \cup D$, если для всех $1 \leq i \leq m$ имеет место: $a_i = b_i$, если $b_i \in \Sigma$, либо $a_i \in b_i$, если $b_i = \delta_k$, $\delta_k \in D$.

В дальнейшем нам понадобятся характеристики "размытости" конкретного образца и группового запроса в целом, введенные в [5]. Пусть $p = b_1 b_2 \ldots b_m$ — частично специфицированный образец. Обозначим через $|b_i|$ число элементов в подмножестве Σ, обозначаемом символом b_i. Очевидно, что $|b_i| = 1$, если $b_i \in \Sigma$ и $|b_i| \geq 2$, если $b_i \in D$. Если $b_i = X$ (любой символ алфавита), то $|b_i| = s$. Величину $|b_i|$ будем называть степенью неопределенности, характеризующей i-ю позицию образца.

Целочисленную величину $Q(p) = \prod\limits_{i=1}^{m} |b_i|$ назовем степенью неопределенности образца p. Если $Q = 1$, образец задан точно (все $b_i \in \Sigma$). Значения Q, большие 1, характеризуют количество точно заданных образцов, согласующихся с p. Полную совокупность таких образцов назовем множеством, эквивалентным заданному частично специфицированному образцу (или просто эквивалентным множеством). Процесс получения из p эквивалентного ему множества $E(p)$ назовем *разверткой* частично специфицированного образца p.

2. Общая схема решения

Рассмотрим для простоты случай, когда образец состоит из точно заданных символов и элементов типа "X". Предлагается два подхода к решению.

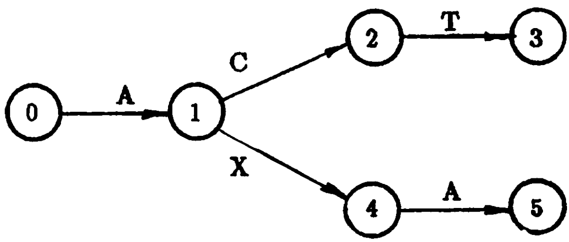
Первый ориентирован на то, чтобы в процессе поиска учесть многозначность в выборе пути, возникающую при прохождении точек ветвления, из которых выходят ребра, помеченные как элементами алфавита Σ, так и элементом X. Эта многозначность разрешается путем двукратного прохождения точки ветвления. Вначале выбирается ребро, помеченное элементом из Σ, и этот путь прослеживается до конца, т.е. до получения отказа либо обнаружения образца. Затем осуществляется возврат в точку ветвления, выбирается ребро, помеченное элементом X, и прослеживается второй маршрут.

Возврат в точку ветвления обеспечивается с помощью *функции отказов*, которая определяется несколько иначе, чем в автомате Ахо–Корасик. Более того, в рассматриваемом варианте не удается избежать и возвратов по тексту, которые задаются с помощью *"функции возвратов"*. Этот элемент отсутствовал в автомате Ахо–Корасик. Возвраты по тексту приводят к неоднократному просмотру отдельных символов текста. Их можно трактовать как плату за экономию памяти, исчисляемую в единицах трудоемкости.

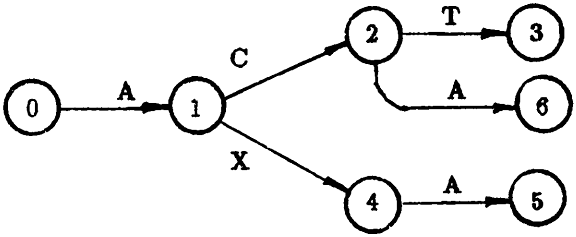
Если в первом подходе возвраты по тексту связаны с прохождением точек ветвления и носят нерегулярный характер, то во втором подходе они делаются сознательно и регулярно. Текст анализируется в режиме скользящего окна, сдвигающегося каждый раз на один символ. Ширина окна l равна размеру ядер, упакованных в автомат. При каждом положении окна решается вопрос о согласованности цепочки текста, выделяемой окном, с образцами из запроса. Поскольку цепочки пересекаются, отдельные элементы текста могут просматриваться неоднократно (l раз в наихудшем случае).

Описанный режим работы не требует использования функции отказов и, что менее очевидно, функции возвратов при анализе *конкретной l-элементной цепочки текста* (возврат делается при переходе к следующей цепочке). Чтобы исключить возвраты при анализе одной цепочки, осуществляется коррекция построенного недетерминированного автомата путем включения в него новых состояний и переходов. Цель этого - проходить точки ветвления *однократно*. Если "a" — управляющий символ, то из точки ветвления осуществляется переход по ребру, помеченному "a", а если таковое отсутствует, то по ребру, помеченному X. Тем самым меняется семантика элемента X : теперь это не произвольный элемент алфавита, а элемент подмножества, дополняющего реберные метки, связанные с точкой ветвления, до алфавита Σ.

При такой трактовке элемента X автомат становится подобным детерминированному. Достройка же его необходима для обеспечения полноты поиска, чтобы "не потерять" отдельные цепочки текста. Пусть, к примеру, имеем множество из двух образцов {ACT,AXA}, которым соответствует недетерминированный автомат



Из состояния 1 данного автомата возможен переход по символу "C" как в состояние 2, так и в 4. Чтобы не отслеживать обе ветви, будем под X понимать множество $\{\Sigma \setminus$ "C"$\}$. Тогда по символу "C" возможен переход только в состояние 2, но при этом не будут обнаруживаться цепочки вида АСА, согласованные с образцом АХА, где X имеет начальную трактовку. Чтобы этого не случилось, образец АСА вводится в автомат в явном виде:



Переход по X делается только в том случае, когда "C" не является входным символом. Данный прием соответствует частичной развертке образца АХА (в отличие от полной, использовавшейся в [5]).

3. Алгоритм 1 поиска по группе образцов, содержащих неопределенные позиции

Рассмотрим сначала частный случай группового запроса из образцов одинаковой длины l, содержащих точно заданные элементы и элементы типа "X" (не более одного на образец). Этот случай будет затем использован при решении общей задачи (раздел 5). Символ "X" будем трактовать как обычный элемент алфавита, т.е. "развертку" по данному элементу проводить не будем, а в автомат включим переходы по "X". Состояния, из которых допускается переход по "X", назовем *точками ветвления* (вне зависимости от того, есть ли переходы по другим символам).

3.1. Построение недетерминированного автомата. Так же как в "базовом" автомате Ахо–Корасик, работа предлагаемого автомата определяется функциями переходов, отказов, выходов, но дополнительно вводится функция "возвратов", которая определяет, на сколько позиций текста необходимо вернуться при "отказе". Опишем алгоритм по шагам.

ШАГ 0. **Инициализация**. Для произвольного состояния автомата t зададим следующие начальные значения:

– функция переходов $g(t, a) = \text{fail}$, т.е. не определена для всех $a \in \Sigma \cup X$;

– функция отказов $f(t)$ не определена;

– функция возвратов $fv(t) = 0$;

– функция выходов $o(t)$ — пустой список.

ШАГ 1. **Построение функции переходов и выходов**. Функцию переходов определяем и строим по множеству образцов аналогично [4]. Принцип построения — склеивание общих префиксов у образцов. Значение $g(t, a) = t'$ интерпретируется как переход из состояния t в состояние t' при наличии "a" в качестве управляющего символа. С символом "X" оперируем как с обычным элементом алфавита.

Отличия от алгоритма Ахо–Корасик.

а) Состояния нумеруем по уровню их "глубины", т.е. номер состояния на уровне q, всегда меньше номера состояния на $(q + 1)$–м уровне. Поскольку для конечных состояний автомата функцию переходов определять не нужно, такая нумерация позволяет избежать "дыр" в двумерном массиве $g(t, a)$, что существенно экономит память [5].

б) Строим список "*точек ветвления*", т.е. состояний, из которых определен переход по символу "X".

в) Строим массив предков (для каждого состояния определяем связанное с ним состояние предыдущего уровня).

Значение $o(t)$ задается списком образцов (или их номеров), для которых состояние t является конечным [4]. Заметим, что в случае образцов одинаковой длины список $o(t)$ для всех внутренних состояний автомата останется пустым, т.е. функция выходов определяется только для конечных состояний автомата.

ШАГ 2. **Построение функции отказов** для точек ветвления и состояний, путь в которые не проходит через точки ветвления, осуществляется так же, как в [4]. Функция отказов определяет, в какое состояние осуществляется переход из состояния t при входном символе "a", если $g(t, "a") =$ fail (этому условию, в частности, удовлетворяют все конечные состояния). Формально $f(t) = t'$, где цепочка символов, связывающая состояния "0" и t', есть максимальный (по всем образцам p) префикс, являющийся одновременно суффиксом цепочки, связывающей состояния "0" и t.

Потомки точек ветвления и состояний, функция отказов для которых указывает на точку ветвления, на этом шаге игнорируются.

ШАГ 3. **Построение функции отказов** для первых потомков точек ветвления.

Пусть t — точка ветвления (см. список, построенный на шаге 1,б), "a" — произвольный символ из Σ, такой что $g(t, a) =$ fail.

Для всех t и "a" выполняем: если $g(t, a) = j$, $g(t, X) = r$, то полагаем $f(j) = r$ (значение $f(r)$ уже получено на втором шаге).

ШАГ 4. **Построение функции отказов и возвратов** для оставшихся состояний (т.е. вторых и последующих потомков точек ветвления).

В порядке *возрастания* номеров состояний (t):

а) находим предка состояния t из массива, построенного на шаге 1,в. Для предка r значение $f(r)$ уже определено, так как $r < t$ (см. шаг 1,а), а данный шаг выполняется в порядке возрастания номеров;

б) определяем функции отказов и возвратов:

$f(t) := f(r)$;

$fv(t) := fv(r) + 1$ (функция возвратов определяет на сколько позиций текста следует вернуться в процедуре поиска).

На рис.1 представлен пример автомата, построенного с помощью алгоритма 1 для образцов {gtac, gXct, gagt, ctag, acta} (пунктирными линиями показана функция отказов, отметки на них соответствуют функции возвратов;

функция отказов для остальных состояний равна нулю — начальному состоянию автомата).

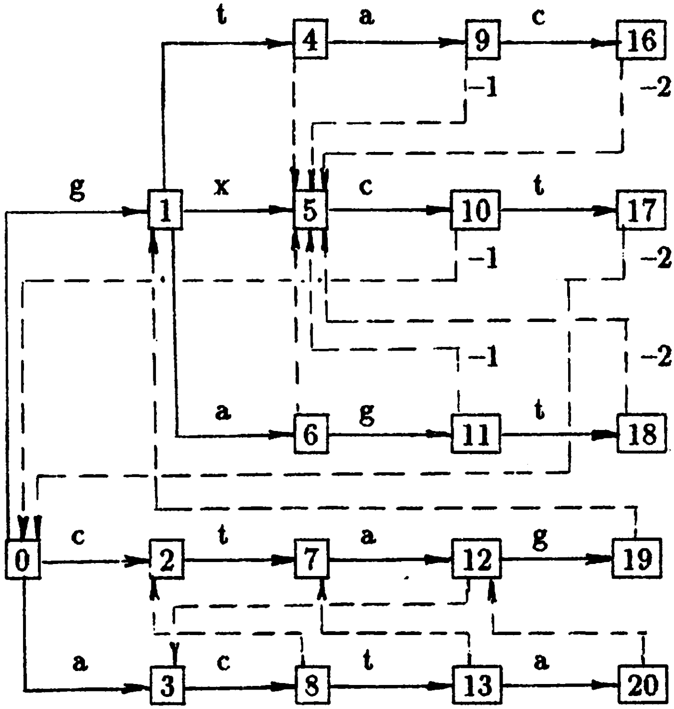


Рис. 1.

3.2. Процедура поиска образцов в тексте. Поиск начинается с первого символа текста и с нулевого состояния автомата.

Пусть t — текущее состояние автомата, i — анализируемая позиция текста, содержащая символ $a = T[i]$. Тогда:

– если t–е состояние внутреннее и $g(t, a) \neq \mathrm{fail}$ (переход по i-му символу текста определен), то: а) $t := g(t, a)$ и б) $i := i + 1$ (идем по функции переходов, сдвигаясь по тексту на один символ вправо);

- если t-е состояние внутреннее и $g(t,a)$ = fail (переход по i-му символу текста не определен), но $g(t,X) \neq$ fail (определен переход по "X"), то осуществляем этот переход: а) $t := g(t,X)$ и б) $i := i+1$;

- если t-е состояние внутреннее, $g(t,a)$ = fail, $g(t,X)$ = = fail (не определен переход ни по $a = T[i]$, ни по "X"), то делаем переход по функции отказов (с учетом функции возвратов): а) $t := f(t)$ и б) $i := i - fv(t)$;

- если t-е состояние — лист (конечное состояние), то

а) фиксируем все "выходы" по функции $o(t)$;

б) $t := f(t)$ (делаем переход по функции отказов);

в) $i := i - fv(t)$ (если $fv(t) \neq 0$, возвращаемся на $fv(t)$ позиций по тексту).

3.3. Оценка суммарной трудоемкости. Трудоемкость построения автомата Ахо-Корасик по образцам пропорциональна их суммарной длине ($n \cdot l$ в случае n образцов одинаковой длины l). Алгоритм, описанный выше, имеет ряд отличий от алгоритма Ахо-Корасик, но порядок трудоемкости не меняется.

Процедура поиска в случае, когда образцы не содержат "X" (либо траектория поиска не проходит через точки ветвления), не отличается от процедуры Ахо-Корасик. Трудоемкость ее составляет $O(N)$, где N — длина текста.

Наихудшим является случай, когда каждый образец содержит элементы типа "X", что может привести к многократным возвратам по тексту. Например, если gact — фрагмент текста, а среди образцов фигурируют {gact, gXct, gaXt, gacX} (каждый из них согласован с фрагментом текста), то для поиска потребуется 4+1+2+ + 3=10 шагов ($1+2+\cdots+l$). Если для каждого фрагмента текста найдется аналогичная группа образцов, поиск займет $N \cdot l \cdot (l+1)/2$ шагов, где каждый шаг — переход в новое состояние с учетом функций g, f и fv. Таким образом, трудоемкость процедуры поиска в наихудшем случае — $O(N \cdot l^2)$. Соответственно суммарная трудоемкость поиска по групповому запросу вместе с предобработкой в наихудшем случае составляет $O(N \cdot l^2 + n \cdot l)$ (при $n \geq l$).

4. Алгоритм 2 поиска по группе образцов, содержащих неопределенные позиции

Вновь рассматриваем частный случай группового запроса, когда образцы имеют одинаковую длину l и состоят из точно заданных элементов и элементов типа "X". Предполагаем, что в одном образце присутствует не более одного "X". Этот случай при малых значениях l является наиболее типичным, и он будет затем использован при решении общей задачи (см. ниже).

Поиск образцов будем вести по каждой l–элементной цепочке текста, поэтому функции отказов и возвратов, минимизировавшие в предыдущем случае число повторно просматриваемых символов текста, здесь не понадобятся. Основная цель — избежать двукратного прохождения точек ветвления, т.е. преобразовать недетерминированный автомат в детерминированный.

В процессе такого преобразования нам понадобится вспомогательная функция "связей" $fs(t)$. Состояние t назовем "*связанным*" с состоянием t' ($fs(t) = t'$), если цепочка символов, соответствующая пути из начального состояния 0 автомата в состояние t "согласована" с цепочкой, связывающей состояния 0 и t'. Если $b_1 b_2 \cdots b_k$ и $c_1 c_2 \cdots c_k$, $k \leq l$, — соответствующие цепочки символов (префиксы образцов), то $b_i = c_i$ или $c_i = X$, $1 \leq i \leq k$. Заметим, что хотя бы одно из c_i равно X, иначе $t = t'$. Функция $fs(t)$ определяется только для тех t, для которых "связи" существуют. Из определения следует, что связи устанавливаются только между состояниями, находящимися на одинаковом расстоянии от начального. Функция связей многозначна: одному значению t может соответствовать несколько значений t'. Эта функция используется только на этапе преобразования автомата, в процедуре поиска она не участвует.

4.1. Построение автомата.

ШАГ 0. **Инициализация**. Пусть t — любое состояние автомата, "a" — произвольный символ алфавита $\Sigma \cup X$. Задаем следующие начальные значения:

- функция переходов не определена: $g(t, a) = \text{fail}$;
- функция выходов $o(t)$ — пустой список;
- функция связей $fs(t)$ не определена.

ШАГ 1. **Построение функции переходов и выходов.** Повторяем шаг 1 алгоритма 1, но не вычисляем массив предков (1,в).

ШАГ 2. **Построение функции связей** для первых потомков точек ветвления, формирование "очереди связей". Пусть t — точка ветвления, $g(t, a) = t1$, $g(t, b) = t2$, $a, b \in \Sigma$; $g(t, "X") = t3$. Устанавливаем "связи": $fs(t1) = t3$, $fs(t2) = t3$, образуем из них очередь. Эти связи удовлетворяют определению, данному выше, так как цепочки символов, соответствующие пути от начального состояния к точке ветвления, совпадают, а последние символы связаны отношением вложенности. Таким образом, цепочки символов (префиксы образцов), заканчивающиеся в $t1$ и $t3$ ($t2$ и $t3$) согласованы.

ШАГ 3. **"Продолжение связей"**. Пока очередь не станет пустой, исключаем из нее первую связь, а в конец очереди записываем *все возможные* продолжения этой связи. Пусть, например, зафиксирована связь $fs(t1) = t2$ и действует ограничение: не более одного "X" на образец. Тогда на любом пути из $t2$ "X" уже не встретится. Для всех $a \in \Sigma$ таких, что $g(t2, a) \neq \text{fail}$ (пусть для определенности $f(t2, a) = w$), есть только три возможности:

а) если $g(t1, a) = v$, то устанавливаем связь $fs(v) = w$ и ставим ее в "хвост" очереди;

б) если $g(t1, a) = \text{fail}$ и $g(t1, "X") = \text{fail}$, то вводим дополнительное состояние v, достраиваем функцию переходов, полагая $g(t1, a) = v$, устанавливаем связь $fs(v) = w$ и ставим ее в очередь. Например, если мы имеем образцы: gcg и gXa, то на шаге 3,б в автомат будет добавлен образец gca, согласованный с gXa и имеющий общее начало с образцом gcg;

в) если $g(t1, a) = \text{fail}$, но $g(t1, "X") = u$, вводим дополнительное состояние v, достраиваем функцию переходов, полагая $g(t1, a) = v$, устанавливаем связи $fs(v) = w$ и $fs(v) = u$ и ставим их в очередь. Для образцов gcX и gXa, например, существует образец gca, согласованный с обоими. Он и включается в автомат на данном шаге.

Одновременно корректируется функция выходов. Установление каждой новой "связи" $fs(r) = r'$ сопровождается добавлением выходов r'-состояния к списку "выходов" r-го состояния.

На рис. 2 представлен автомат, построенный по данному алгоритму на примере той же группы образцов, что и на рис. 1.

4.2. Процедура поиска образцов в тексте. Текст сканируется окном ширины l, сдвигающимся каждый раз на один символ. Отождествление очередной цепочки длины l, выделяемой окном, с образцами из запроса начинается путем передачи управления в начальное состояние автомата. Пусть в некоторый момент времени автомат находится в состоянии t и анализируется r-й элемент q-й цепочки ($1 \leq q \leq N - l + 1$, $1 \leq r \leq l$), что соответствует символу текста $a = T[q + r - 1]$. Тогда:

- если $r \leq l$ (t-е состояние внутреннее) и $g(t, a) \neq \text{fail}$ (переход по символу "a" разрешен), то а) $t := g(t, a)$ и б) $r := r + 1$, что эквивалентно переходу к следующему символу текста);

- если $r \leq l$ и $g(t, a) = \text{fail}$ (переход по символу "a" не определен), но $g(t, "X") \neq \text{fail}$ (определен переход по "X"), то осуществляем этот переход: а) $t := g(t, "X")$ и б) $r := r + 1$. Фактически "переход по X" осуществляется в этом автомате уже не по любому символу алфавита, а только по тем из них, которые дополняют множество реберных меток узла t до полного алфавита Σ. В этом смысле достроенный автомат можно трактовать как детерминированный;

- если $r \leq l$, $g(t, a) = \text{fail}$ и $g(t, "X") = \text{fail}$ (не определен переход ни по "a", ни по "X"), то рассматриваемая цепочка текста не соответствует ни одному из образцов. Пере-

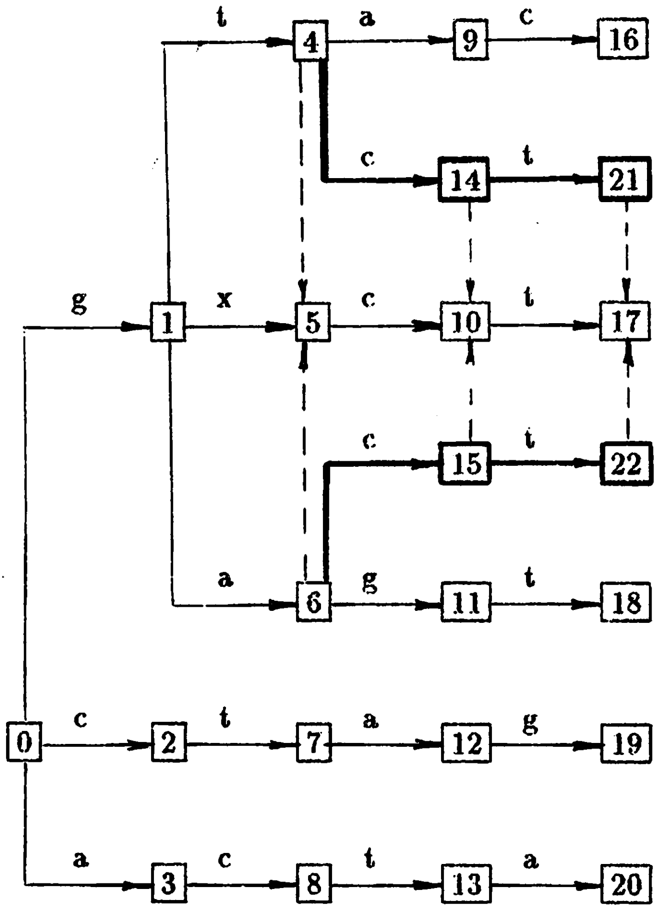


Рис. 2. Автомат, построенный по алгоритму 2 для образцов {grac, gXct, gagt, ctag, acta}. Достроенные состояния показаны жирными линиями. Пунктирными линиями отмечены "связи".

ходим к анализу следующей цепочки, вернувшись в начальное состояние автомата: а) $t := 0$ и б) $r := 1 (q := q+1)$;

- если $r = l+1$ и, соответственно, t-е состояние — лист (конечное состояние), то образец найден:

а) фиксируем позицию текста и все "выходы" $o(t)$;

б) $t := 0$ (переходим в начальное состояние автомата);

в) $r := 1 (q := q+1)$ (переходим к следующей l-ке текста).

4.3. Оценка суммарной трудоемкости. Трудоемкость построения автомата (шаг 1) пропорциональна суммарной длине образцов, т.е. $n \cdot l$ в случае n образцов одинаковой длины l. При выполнении шага 3 в автомат добавляются образцы, получающиеся путем подстановки вместо "X" символов алфавита Σ. В наихудшем случае возможна полная "развертка" образцов. Трудоемкость шагов 1-3 тогда составит $n \cdot l \cdot \overline{Q}$, где $\overline{Q}$ — средняя степень неопределенности, приходящаяся на один образец.

Для поиска каждой l-ки требуется не более l шагов (в случае, когда фрагмент текста соответствует хотя бы одному образцу). Таким образом, трудоемкость процедуры поиска в наихудшем случае — $O(N \cdot l)$.

Суммарная трудоемкость поиска по групповому запросу (вместе с предобработкой) в наихудшем случае составляет $O(N \cdot l + n \cdot l \cdot \overline{Q})$.

5. Алгоритм поиска по групповому частично специфицированному запросу (общий случай)

В алгоритме, описанном в [5], использовался автомат Ахо–Корасик, строившийся по "развернутым" ядрам. В предыдущих разделах описаны алгоритмы, позволяющие не производить развертку неопределенной позиции (типа "X"), а непосредственно включать "X" в автомат как некоторый выделенный символ. Это позволяет использовать ту же стратегию, что и в [5], но не избегать ядер, содержащих "X".

ПРИМЕР. Пусть задан образец:

F-[TV]-X-[DER]-[FY]-[IL]-X-E-[FIKRS]-[NS]-[AQRS]-[DKMR]-R

Здесь черточки разделяют позиции образца, все символы (кроме X) — элементы аминокислотного алфавита ($s = 20$), частично специфицированные позиции заданы перечислением допустимых (для каждой позиции) символов в квадратных скобках. Предположим, что мы хотим выбрать фрагмент длины 4 с минимальной степенью неопределенности (ядро). Если считать $|X| = 20$, то ядром данного образца будет фрагмент [NS]-[AQRS]-[DKMR]-R, степень неопределенности которого равна 32 (именно такое ядро было бы выбрано в алгоритме, описанном в [5]). Если же X рассматривать как выделенный символ алфавита, полагая $|X| = 1$, то ядром будет фрагмент [FY]-[IL]-X-E со степенью неопределенности 4.

Приведенный пример носит в значительной мере иллюстративный характер. На практике при выборе оптимального ядра нужно учитывать дополнительные обстоятельства:

а) не совсем корректно считать $|X| = 1$, поскольку включение элемента X в алфавит приводит к некоторому замедлению алгоритма. При прочих равных условиях всегда выгоднее выбрать ядро, где вместо X стоит элемент алфавита Σ;

б) следует избегать появления X в качестве первого элемента ядра, что с одной стороны равносильно уменьшению длины ядра на единицу, а с другой приводит к тому, что начальное состояние автомата становится точкой ветвления и процедура поиска замедляется;

в) невыполнение предпосылки о наличии в ядре не более чем одного элемента типа "X" приводит к существенному усложнению поиска.

Для учета соображений "а"–"в" целесообразно скорректировать понятие степени неопределенности образца (или ядра), приведенное в разделе 1. Пусть $z = b_k b_{k+1} \dots b_{k+l-1}$ — ядро длины l, выделяемое из образца p, $1 \leq k \leq |p| - l + 1$. Степень неопределенности i-й позиции ядра z зададим следующим образом:

$d_i = |b_i|$, если $b_i \neq X$, $k \leq i \leq k + l - 1$;

$d_i = c$, если $b_i = X$ и $b_j \neq X$ для всех $i \neq k$, $j \neq i$, где c — некоторая константа, большая 1 (см. условие "а"). Из эмпирических соображений для c можно рекомендовать диапазон значений от 1 до 2;

$d_i = |\Sigma|$, если $i = k$ и $b_i = X$ (см. условие "б");

$d_i = |\Sigma|$, если $b_i = X$ и $b_j = X$, $j \neq i$, $k \leq j \leq k + l - 1$, $k + 1 \leq i \leq k + l - 1$ (см. условие "в").

Тогда степень неопределенности всего ядра z:

$$R(z) = \prod_{i=k}^{k+l-1} d_i.$$

Выбор ядер с минимальным значением $R(z)$ позволяет существенно уменьшить затраты памяти и время предобработки. Следует заметить, что характеристика $R(z)$ уже не имеет столь прозрачной интерпретации, как $Q(z)$, в том смысле, что она не соответствует числу константных фрагментов, эквивалентно представляющих частично специфицированный фрагмент z.

Для получения оценок трудоемкости понадобится еще одна величина, характеризующая число фрагментов в "развертке" ядра z:

$$H(z) = \prod_{i=k}^{k+l-1} d_i,$$

где $d_i = |b_i|$, если $b_i \neq X$, $k \leq i \leq k + l - 1$ и $d_i = 1$, если $b_i = X$. (Элементы типа "X" не подвергаются развертке, а остальные частично специфицированные позиции — подвергаются.)

5.1. Построение автомата.

ШАГ 1. **Выбор размера ядра**. Выбираем размер ядра l, исходя из мощности алфавита Σ, степени неопределенности образцов в запросе и имеющихся ресурсов оперативной памяти. Значение $l = 3$ давало оптимальный результат в алгоритмах, описанных в [5], для алфавитов средней мощности ($|\Sigma| = 20 \div 50$). Учитывая большую вероятность попадания "X" в ядро (а это неопределенная позиция), значение l рекомендуется выбирать равным 4 и выше.

ШАГ 2. **Выбор ядра**. В каждом образце $p_j = b_1 b_2 \dots b_m$, $1 \leq j \leq n$, выбираем ядро z_j длины l в виде фрагмента $b_k b_{k+1} \dots b_{k+l-1}$ с минимальным значением $R(z_j)$. Положение оптимального ядра в образце p (номер позиции k) фиксируем в j-й ячейке массива J размерности n.

ШАГ 3. **Развертка ядер**. Осуществляем развертку каждого из ядер z_j, $1 \leq j \leq n$, совмещая ее с построением автомата по множествам $E(z_j)$ (элементы типа "X" разверткой не затрагиваются). Построение автомата ведем с помощью алгоритмов 1 или 2, описанных в разделах 3.1 и 4.1.

5.2. Поиск по тексту.

ШАГ 4. **Обнаружение ядер**. Пусть T — анализируемый текст. Пропускаем его через автомат, пользуясь процедурами, описанными в разделах 3.2 или 4.2 (в зависимости от того, какой автомат выбран на шаге 3). Фиксируем символы текста, переводящие автомат в одно из конечных состояний. Каждый из них является последним элементом ядра, характеризующего совокупность образцов, связанных с соответствующим конечным состоянием.

ШАГ 5. **Расширение ядер**. Рассматриваем последовательно ядра, выделенные в тексте T на шаге 4. Пусть очередному ядру длины l соответствует конечное состояние t. Проверяем, согласуются ли лево- и правосторонние расширения ядра в тексте T с образцами, представленными в списке $o(t)$. Подробное описание этого шага смотрите в [5].

5.3. Оценка суммарной трудоемкости. В [5] получены оценки трудоемкости аналогичного алгоритма, в котором ядерный автомат строится с помощью алгоритма Ахо–Корасик (шаг 3) . Мы на этом шаге пользуемся алгоритмами, описанными в разделах 3.1 и 4.1. Оценки их трудоемкости для наихудшего случая приведены в разделах 3.3 и 4.3. Для остальных шагов алгоритма имеют место оценки из [5].

Пусть $s = |\Sigma|$. Если на шаге 3 используется алгоритм 1, суммарная трудоемкость шагов 2 и 3 (трудоемкость предобработки) составляет $O\left(\sum_{i=1}^{n} |p_i| + n \cdot l \cdot \overline{H}\right)$. Трудоемкость поиска (шаги 4 и 5) составляет в наихудшем случае $O(N \cdot l^2 + N \cdot n \cdot \overline{Q}/s^l)$, а суммарная трудоемкость поиска и предобработки в наихудшем случае — $O[N(l^2 + n \cdot \overline{Q}/s^l) + \sum_{i=1}^{n} |p_i| + n \cdot \overline{H} \cdot l]$.

Если на шаге 3 используется алгоритм 2, суммарная трудоемкость шагов 2 и 3 (предобработка) составляет в наихудшем случае $O\left(\sum_{i=1}^{n} |p_i| + n \cdot l \cdot \overline{Q}\right)$ (здесь $\overline{Q}$ — значение, соответствующее оптимальному $\overline{R}$). Трудоемкость поиска (шаги 4 и 5) составляет в наихудшем случае $O(N \cdot l + N \cdot n \cdot \overline{Q}/s^l)$, а суммарная трудоемкость поиска и предобработки в наихудшем случае — $O[N(l + n \cdot \overline{Q}/s^l) + \sum_{i=1}^{n} |p_i| + n \cdot \overline{Q} \cdot l]$.

6. Обсуждение

Алгоритм 1 описан для случая, когда все образцы имеют одинаковую длину, чтобы подчеркнуть специфику его использования при решении общей задачи (см. раздел 5). В общем случае ограничение на длину образцов не является принципиальным. Алгоритм остается корректным и для образцов разной длины. В алгоритме 2 поиск в тексте ведется по фрагментам равной длины, т.е. одинаковая длина образцов здесь необходима.

Ограничение на число элементов типа "X" (не более одного на ядро) является желательным, но не обязательным, в обоих алгоритмах. Оно упрощает описание и получение оценок трудоемкости. Однако при невыполнении данного ограничения запрос становится слишком "расплывчатым" (неопределенным), ему удовлетворяет большое число фрагментов текста, что автоматически ведет к повышению трудоемкости.

Алгоритм 2 проигрывает первому по памяти и времени предобработки, однако в наихудшем случае выигрывает по времени поиска. Поэтому его целесообразно применять (как часть алгоритма из раздела 4 или самостоятельно) в случаях, когда образцов не очень много (из-за ограничений по памяти), но степень их неопределенности высока: почти каждый содержит элементы типа "X".

Если частично специфицированный запрос не содержит элементов типа "X", то алгоритм 1 эквивалентен алгоритму из [5]. Если запрос общего вида (с элементами типа "X"), то нужно учитывать, что с одной стороны использование алгоритмов 1 и 2 вместо [5] предполагает увеличение размера ядра, что увеличивает память и время предобработки, а с другой стороны уменьшает степень неопределенности ядра $H(z)$, что существенно уменьшает память и время предобработки. Различий во времени поиска с помощью алгоритмов 1,2 и [5] на практике не заметно, но диапазон применимости алгоритмов 1 и 2 шире. Все алгоритмы апробированы на реальных групповых запросах, содержащих порядка 10^3 – 10^4 образцов, возникающих при анализе генетических текстов (ДНК- и аминокислотных последовательностей). На персональных компьютерах среднего класса обработка идет практически в реальном масштабе времени.

Заключение

В [5] рассматривалась задача поиска в текстовых базах данных по групповому частично специфицированному запросу большой мощности. Предполагалось, что элементы поискового образца могут быть заданы с точностью до принадлежности к произвольному подмножеству исходного алфавита. Схема решения задачи предполагала выделение на первом этапе опорных ядер в каждом из образцов (коротких фрагментов с минимальной степенью неопределенности), замену их эквивалентным (с точки зрения полноты поиска) множеством "константных" ядер, построение по ним распознающего конечного автомата (типа Ахо-Корасик) и расширение обнаруживаемых в тексте ядер до размеров образца. Существенное неудобство при таком подходе создавали сильно неопределенные образцы, для которых не удавалось выбрать ядро, не содержащее элементов типа "X", характеризующих несущественные позиции.

В данной работе в развитие [5] предложены два алгоритма решения задачи поиска по групповому запросу, элементы которого заданы точно или несущественны. В отличие от [5] используются недетерминированные конечные автоматы. Элементы типа "X" непосредственно включаются в автомат, образуя точки ветвления, обработка которых определяет стратегию поиска.

Предложенные алгоритмы имеют самостоятельное значение. Помимо этого они могут быть использованы в составе алгоритма из [5], делая допустимым выбор ядер, содержащих константные символы и элементы типа "X". Это существенно расширяет возможности указанного алгоритма, экономит память и время предобработки при незначительном увеличении времени поиска.

Автор выражает благодарность Гусеву В.Д. за постановку задачи и обсуждение.

Литература

1. FITCHEV M., PATERSON M. String matching and other products //Proc. 7th SIAM-AMS Symp. on Complexity of Computation. 1974. — P. 113-125.

2. PINTER Ron Y. Efficient string matching with don't-care patterns //Combinatorial algorithms on words (ed. by A.Apostolico and Z.Galil).— Springer Verlag, 1985.— NATO ASI Series, Vol. F12. — P. 11-29.

3. WU Sun, MANBER U., MYERS Y. A Subquadratic algorithm for approximate limited expression matching //Algorithmica. — 1996. — № 15. — P. 50-67.

4. AHO A.V., CORASICK M.J. Efficient string matching: an aid to bibliographic search //Communications of the ACM. — 1975. — Vol. 18, № 6. — P. 333-340.

5. ГУСЕВ В.Д., НЕМЫТИКОВА Л.А. Алгоритмы поиска в текстовых базах данных по групповому частично специфицированному запросу //Искусственный интеллект и экспертные системы. — Новосибирск, 1996. — Вып. 157: Вычислительные системы. — С. 12–39.

Поступила в редакцию
10 сентября 1997 года