Задача о наименьшей надстроке[1]

Илья Васильев, Никита Погадаев

Ноябрь 2022 - Январь 2023

Аннотация

Задача о наименьшей надстроке — известная классическая задача, имеющая применения в таких сферах как сравнение текстовых файлов или биоинформатике (в частности генетике и молекулярной биологии). На вход поступает множество строк (над ограниченным алфавитом), требуется найти такую строку, что любая строка из поступившего множества является подстрокой в найденной, при этом требуется минимизировать длину найденной строки. Чтобы свести данную задачу поиска к задаче с бинарным ответом, мы немного меняем формулировку: на вход поступает множество строк и верхняя оценка k, нам же нужно ответить на вопрос, существует ли надстрока (в смысле, описанном выше) длины, не больше k.

На сегодняшний день мы не знаем эффективных точных алгоритмов для её решения. Наилучший известный приближённый жадный алгоритм для этой задачи находит решение с 2.5-приближением, но есть основания полагать, что жадный алгоритм может находить решение с 2-приближением, но формального доказательства нет.

В данной статье мы планируем дать краткий обзор данной задачи и ее сложности, описать и имплементировать простой жадный алгоритм, а также протестировать и собрать статистику на коротких строках и проверить на них гипотезу о том, что жадный алгоритм дает 2-приближение. Основной код и псевдо-код будет представлен на основе языка C++, тестирование и обработка данных будет выполняться с помощью python.

План

1. Введение

Будем называть описанную выше строку S надстрокой над монжеством M строк, если для любой строки s измножества M существуют индексы i, j, т. ч. S[i..j] = s. Самые последние результаты по приближению к точному значению равны $2\frac{11}{23}$ due to Mucha[2], and $2\frac{11}{30}$ due to Paluch[3], [4]. В данном контексте речь идет о коэффициенте сходимости, который мы определим в пункте ниже. Мы хотим проверить гипотезу о 2-сходимости на маленьких тестах, данная гипотеза гласит, что жадный алгоритм дает эпмирически провереный коэффициент сходимости равный двум (как было описано выше максимальное приближение, то есть миниммальный коэффициент чуть больше двойки).

2. Техническая часть

В данном контексте речь идет о коэффициенте сходимости, который определяется как результат деления полученного результата на точный результат, полученный, к примеру, наивным алгоритмом на маленьких тестах. Блюм доказывает, что жадный алгоритм дает коэффициент сходимости 4 (доказательство [6]), Каплан и Шафрир [7] улучшают коэффициент до 3.5. Известно, что жадная гипотеза о кэффициенте 2 верна для случая, когда все входные строки имеют длину не более 3, и недавно было показано, что оно выполняется в случае строк длины 4 [8].
Жадный алгоритм также дает 2-аппроксимацию другой метрики, так называемого сжатия. Сжатие определяется как сумма длин всех входных строк минус длина найденной надстроки (следовательно, это количество символов, сэкономленных по отношению к точно найденной надстроке с помощью навиного алгоритма, полученной в результате объединения входных строк). Мы собираемся проверить оба метрических подхода для проверки гипотезы. Мы сгенерируем несколько случайных множеств из слов, длин небольше 3 и 4, при этом фиксируя количество слов в выборке для каждого теста. Для того, чтобы наивный алгоритм работал не слишком долго, ограничим алфавит до 2 символов (0, 1). Также мы приведем графики зависимости значений коэффициента от размера выборки. Формально и у наивного алгоритма и у жадного сигнатура будет следующей:
Input: set of strings M.
Output: a superstring for M.

3. Основная часть

Докажем, что задача поиска сводится к NP-полной задаче в соответствующей формулировке: существует ли надстрока S длина не более чем k для данного множества строк M (на вход подается M и k).
Для начала надо понять, лежит ли данная перефразированная задача в NP, является ли она полной. Ответ на эти вопросы утвердительный и доказательство содержится в статье [9].

Задача лежит в NP, так как в качестве сертификата достаточно просто привести просто итоговую надстроку (ее длина должна быть не более чем суммарная длин всех строк, если она больше чем k или суммарная длина строк, отвергаем сертификат), а затем за O(N^3) тривально (проверяя каждую строку из M на вхождение в сертификат за O(N^2)) вхождение каждой строки из M в наш сертификат.
Также нужно понять, является ли задача NP-hard. Основная идея состоит в том, что мы будем сводить задачу о вершинном покрытии к нашей задаче [9], строя множество строк E' из e + 1 (над алфавитом (0, 1)), где e - мощность множества ребер в графе. Особым образом кодируя ребра и вершины в графе, мы сводим нахождение множества вершин, к поиску надстроки для множества E' строк, соответсвующих ребрам графа.

Далее мы должны построить жадный алгоритм, который будет давать 4-сходимость, но на самом деле оценка уже была улучшена в результатах выше. Для того, чтобы описать алгоритм, небходимо ввести понятие overlap: overlap$(s_i, s_j) = max_{len(s)}(s : s - suff(s_i)\ and\ pref(s_j))$. Жадный алгоритм довольно просто и лаконичен: он будет выбирать две строки с максимальным overlap и конкатенировать первую строку s_i со строкой s_j без префикса в виде overlap(s_i, s_j), затем удалять s_i и s_j и добавлять вместо них новую полученную строку.

Input: set of strings M.
Output: a superstring for M.
.while M contains at least two strings do
. extract from M two strings with the maximum overlap
. add to M the shortest superstring of these two strings
.return the only string from M

Укконен [5] показывает, что для фиксированного алфавита жадный алгоритм может быть реализован за линейное время. Следует отметить, что жадный алгоритм не является детерминированным, поскольку мы не указываем, как разрывать связи в случае, когда имеется много пар строк с максимальным перекрытием, поэтому жадный алгоритм может создавать разные надстроки для одних и тех же входных данных (с точки зрения перестановки самих строк в множестве M).

Чтобы подвердить гипотезу о 2-сходимости, мы должны проверить результаты, полученные жадным алгоритмом с помощью тривиального алгоритма. Для того, чтобы перебор не работал слишком долго, мы ограничимся алфавитом из двух символов (0, 1), а также гипотезу, согласно результатам выше, будем проверять на строках длины 3 и 4, для количества строк от 1 до 8(16). Тогда известно, что цикл де Брёйна для строк не более чем длины 3 будем 8, а для 4 - 16, соотвественно перебрать нужно будет не более чем 2^8 надстрок в случае длин не более чем 3, и не более чем 2^{16} надстрок в случае длин не более чем 4 [10].

Графики зависимости коэффицентов сходимости и коэффициентов сходимости сжатия в зависимости от количества строк в зависимости от длин приведены ниже.

```
Alphabet = {a,b}. Length = 1. Number = 2.  =>  1
Alphabet = {a,b}. Length = 2. Number = 4.  =>  1.2
Alphabet = {a,b}. Length = 3. Number = 8.  =>  1.2
Alphabet = {a,b}. Length = 4. Number = 16.  =>  1.26316
Alphabet = {a,b}. Length = 5. Number = 32.  =>  1.25
Alphabet = {a,b}. Length = 6. Number = 64.  =>  1.27536
Alphabet = {a,b}. Length = 7. Number = 128.  =>  1.26119
Alphabet = {a,b}. Length = 8. Number = 256.  =>  1.27757
Alphabet = {a,b,c}. Length = 1. Number = 3.  =>  1
Alphabet = {a,b,c}. Length = 2. Number = 9.  =>  1.2
Alphabet = {a,b,c}. Length = 3. Number = 27.  =>  1.24138
Alphabet = {a,b,c}. Length = 4. Number = 81.  =>  1.2619
Alphabet = {a,b,c}. Length = 5. Number = 243.  =>  1.24696
Alphabet = {a,b,c,d}. Length = 1. Number = 4.  =>  1
Alphabet = {a,b,c,d}. Length = 2. Number = 16.  =>  1.17647
Alphabet = {a,b,c,d}. Length = 3. Number = 64.  =>  1.21212
Alphabet = {a,b,c,d}. Length = 4. Number = 256.  =>  1.22008
Alphabet = {a,b,c,d,e}. Length = 1. Number = 5.  =>  1
Alphabet = {a,b,c,d,e}. Length = 2. Number = 25.  =>  1.15385
Alphabet = {a,b,c,d,e}. Length = 3. Number = 125.  =>  1.1811
```

Рис. 1: Разные алфавиты

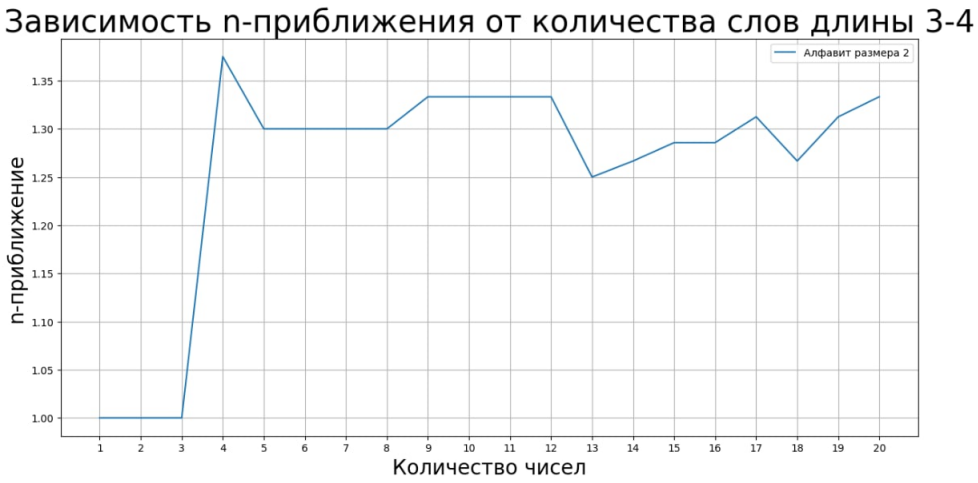

Рис. 2: График коэффициента сходимости

4. Ссылки

1. Collapsing Superstring Conjecture, Alexander Golovnev, Alexander S. Kulikov, Alexander Logunov, Ivan Mihajlin, Maksim Nikolaev

2. Marcin Mucha. Lyndon Words and Short Superstrings. In SODA 2013, pages 958–972. SIAM, 2013

3. Katarzyna Paluch. Better approximation algorithms for maximum asymmetric traveling salesman and shortest superstring. arXiv preprint arXiv:1401.3670, 2014.
4. Alexander Golovnev, Alexander S. Kulikov, and Ivan Mihajlin. Approximating shortest superstring problem using de Bruijn graphs. In CPM 2013, pages 120–129. Springer, 2013
5. Esko Ukkonen. A linear-time algorithm for finding approximate shortest common superstrings. Algorithmica, 5(1-4):313–323, 1990.
6. vrim Blum, Tao Jiang, Ming Li, John Tromp, and Mihalis Yannakakis. Linear approximation of shortest superstrings. In STOC 1991, pages 328–336. ACM, 1991.
7. Haim Kaplan and Nira Shafrir. The greedy algorithm for shortest superstrings. Inf. Process. Lett., 93(1):13–17, 2005
8. Alexander S. Kulikov, Sergey Savinov, and Evgeniy Sluzhaev. Greedy conjecture for strings of length 4. In CPM 2015, pages 307–315. Springer, 2015.
9. The shortest common supersequence problem over binary alphabet is NP-complete. Kari-Jouko Räihä, Esko Ukkonen, Volume 16, Issue 2, 1981, Pages 187-198.
10. Последовательность де Брёйна.