

Задача о минимальной надстроке

Жуков Владислав 499

Определение 1. Пусть задано множество строк $A = \alpha_1, \alpha_2, \dots, \alpha_n$ над конечным алфавитом. Требуется найти такую строку ω , что все строки из множества A являются подстроками ω и длина ω минимальна. Назовем эту задачу "задачей о надстроке" или SSP.

Определение 2. $IN(v), OUT(v)$ - обозначают количество входящих и исходящих ребер соответственно из вершины v .

Определение 3. (Задача об ограниченном направленном Гамильтоновом пути)
Задача направленного Гамильтонова пути:

Дан ориентированный граф G , есть ли в нем путь проходящий по всем вершинам ровно по одному разу. Про эту задачу известно, что он NP-полная.

Задача ограниченного направленного Гамильтонова пути:

Это задача направленного Гамильтонова пути со следующими ограничениями: Есть назначенная стартовая вершина s и конечная вершина t , такие, что $IN(s) = OUT(t) = 0$
Все вершины кроме конечной вершины t , имеют исходящую степень большую 1.

Определение 4. $overlap(s_1, s_2)$ Это длина наибольшего x , такого что $s_1 = ax$ и $s_2 = xb$ для некоторых строк a и b . Проще говоря, это длина максимально возможного перекрытия двух строк.

Теорема 1. SSP с бесконечным алфавитом является NP-полной.

Доказательство: Пусть $G = (V, E)$ это граф из задачи ограниченного направленного Гамильтонова пути, где V это множество целых чисел от 1 до n (1 это начальная вершина и n это конечная вершина) и $|E| = m$. Мы построим строки для G над алфавитом $\Sigma = V \cup B \cup S$, где $B = \{\bar{v} | v \in V - \{n\}\}$ множество "запрещенных" символов а S - множество специальных символов. Запрещенные символы являются "локальными" для каждой вершины, тогда как специальные являются глобальными символами для всего графа G . Для каждой вершины $v \in V - \{v\}$ мы сопоставим множество A_v содержащее $2OUT(v)$ строк. Пусть $R_v = \{\omega_0, \dots, \omega_{OUT(v)-1}\}$ это множество вершин соединенных с v . Тогда, $A_v = \{\bar{v}\omega_i\bar{v} | \omega_i \in R_v\} \cup \{\omega_i\bar{v}\omega_{i \oplus 1} | \omega_i \in R_v\}$, где \oplus обозначает сложение по модулю $OUT(v)$.

Для каждой вершины $v \in V - \{1, n\}$ создаем множество C_v из одного элемента, содержащее строку $v\#\bar{v}$ называемую соединителем или коннектором. Введем множество, которое содержит терминальные строки $T = \{\% \# \bar{1}, n\#\$\}$. Пусть S это объединение $A_j, 1 \leq j < n; C_j, 1 \leq i < n$ и T . Утверждается, что G имеет направленный Гамильтонов путь в том и только в том случае, если S имеет надстроку длины $2m + 3n$.

Предположим, что в G есть направленный Гамильтонов путь. Пусть v, ω_i ребро из этого пути. Для начала построим надстроку длины $2OUT(v) + 2$ для A_v вида $\bar{v}\omega_i\bar{v}\omega_{i \oplus 1}\bar{v} \dots \bar{v}\omega_i$, называемую ω_i -стандартной надстрокой для A_v . Эта надстрока сформирована "схлопыванием" перекрытий строк A_v в порядке

$$\bar{v}\omega_i\bar{v}, \omega_i\bar{v}\omega_{i \oplus 1}, \bar{v}\omega_{i \oplus 1}\bar{v}, \dots, \bar{v}\omega_{i \oplus OUT(v)}\bar{v}, \omega_{i \oplus OUT(v)}\bar{v}\omega_i$$

, где каждая последующая пара имеет перекрытие длины 2. Отметим, что множество ω_i -стандартных надстрок для A_v переходят друг в друга в соответствии с циклическими перестановками целых чисел от 0 до $OUT(v) - 1$. Пусть u_1, u_2, \dots, u_n обозначает направленный Гамильтонов путь где $u_1 = 1, u_n = n$ и обозначим стандартную u_j надстроку для A_{u_i} как $STD(\bar{u}_i, u_j)$. Мы можем построить надстроку для S как схлопывание стандартных надстрок и строк из S в конкретном порядке:

$$\% \# \bar{1}, STD(\bar{1}, u_2), u_2\#\bar{u}_2, STD(\bar{u}_2, u_3), u_3\#\bar{u}_3, \dots, \bar{u}_{n-1}\#\bar{u}_{n-1}, STD(\bar{u}_{n-1}, n), n\#\$$$

Надстрока имеет длину $\sum_{i=1}^{n-1} (2OUT(i) + 2) + (n - 2) + 4 = 2m + 3n$.

Чтобы доказать обратное утверждение, мы покажем, что $2m + 3n$ это нижняя граница размера надстроки S и затем покажем, что эта нижняя граница может быть достигнута только в случае если надстрока кодирует Гамильтонов путь. Всего мы имеем $2m + n$

строк, в сумме их длина $3(2m+n)$. Наибольшее "сжатие" дает порядок, в котором каждая строка кроме первой имеет перекрытие длины 2 с обеих сторон. Этот порядок должен дать надстроку длины $3(2m+n) - 2(2m+n-1) = 2m+n+2$. Однако, $n-2$ коннектора могут иметь перекрытие только длины 1 с обеих сторон, т.к. ни одна строка не начинается и не заканчивается с символа $\#$. К тому же терминальные строки могут перекрываться максимум на 1 символ только с одной стороны. Соблюдая эти условия, мы имеем нижнюю границу на длину надстроки в $(2m+n+2) + 2(n-2) + 2 = 2m+3n$ для S . Отметим, что она начинается с $\% \# \bar{1}$ и заканчивается $n \# \$$. Рассмотрим два вхождения $\#$ в такую надстроку. Обозначим за x то, что находится между этими двумя знаками $\#$. Первый символ из x должен быть запрещенным, а последний не запрещенным, поскольку они являются подстрокой соединителя. Если в x нет соединителей, то тогда все подстроки кроме первой и последней должны иметь перекрытие 2 с обеих сторон. Первая строка должна быть $\bar{v}u_j\bar{v}$, следующая $u_j\bar{v}u_{j\oplus 1}$ и так далее. Более того, все строки в A_v кроме двух последних должны иметь перекрытие длины 2 с обеих сторон, так каждая последующая строка должна быть "добита" уникальной строкой, которая перекрывается с ней на 2 символа. Таким образом все строки в A_v должны появляться в конкретном порядке, и если x содержит одну строку из A_v , то он обязан содержать их все. Таким образом, x - это стандартная надстрока для A_v . Применяя рассуждения выше ко всем вхождениям пар $\#$ мы получаем $n-1$ различную стандартную строку. Мы можем восстановить Гамильтонов путь смотря на символы следующие за каждым вхождением $\#$, причем запрещенные и не запрещенные символы каждого соединителя отвечают одной и той же вершине в G . Отметим, что в силу расположения $\% \# \bar{1}$ и $b \# \$$, мы получаем путь из 1 в n

Построение приближенного алгоритма и доказательство 4-приближенности

Определение 5. Префиксным графом для множества строк S назовем полный ориентированный граф, в котором вершинами являются строки из S , где ребро (s_i, s_j) имеет вес $pr(s_i, s_j)$

Определение 6. Обозначим за $\langle s_{i_1}, s_{i_2} \dots s_{i_n} \rangle$ строку:

$$pr(s_{i_1}, s_{i_2})pr(s_{i_2}, s_{i_3}) \dots pr(s_{i_{n-1}}, s_{i_n})s_{i_n}$$

Пусть S это множество строк из задачи о минимальной надстроке. Мы попытаемся построить другое множество R , такое что:

- 1) Строки в R не сильно перекрываются,
- 2) Каждая строка $s \in S$ имеет надстроку в R , так что любая надстрока для R является надстрокой для S ,
- 3) Оптимальное решение OPT_R для R не сильно хуже оптимального решения OPT_S для S .

Как только мы построили такое R мы можем найти приближенное решение для R сведением к асимметричному наибольшему пути в TSP

Построим множество для R .

- 1) Найдем минимальное покрытие циклами \mathcal{C} в графе префиксов.
- 2) Для каждого цикла $C \in \mathcal{C}$, построим представление $r(C)$ содержащее все строки для C как подстроки. Пусть $C = \{s_{i_1}, \dots, s_{i_k}\}$ мы можем взять $r(C) = \langle s_{i_1}, \dots, s_{i_k} \rangle$
- 3) R это множество всех представлений.

Лемма 1. Суммарная длина $\omega(\mathcal{C})$ всех циклов минимального покрытия \mathcal{C} ограничена сверху величиной $\leq OPT$ оптимального решения.

Доказательство. Верно поскольку суммарная длина минимального покрытия циклами графа префиксов не больше чем длина минимального TSP пути графе, которая является оценкой снизу для OPT . (Поскольку OPT имеет вид $\langle s_{i_1}, \dots, s_{i_n} \rangle$ и его длина больше чем цикл на этих же вершинах в префиксном графе)

Лемма 2. $OPT_R \leq 2OPT_S$

Доказательство:

Вспомним, что все строки $r \in R$ имеют вид $r = \langle s_{i_1}, \dots, s_{i_k} \rangle$. Рассмотрим немного более длинные $\hat{r} = \langle s_{i_1}, \dots, s_{i_k}, s_{i_1} \rangle$. Мы покажем, что $OPT_{\hat{R}} \leq 2OPT_S$

Каждая строка $\hat{r} \in \hat{R}$ начинается и кончается с одной и той же строки, обозначим ее за $s(\hat{r})$. Пусть $S(\hat{R})$ это набор таких строк $s(\hat{r})$. Очевидно $OPT_{S(\hat{R})} \leq OPT(S)$. Но отметим, что оптимальное решение для $S(\hat{R})$ может быть найдено из решения для \hat{R} . Просто заменим каждую $s(\hat{r})$ на соответствующую строку r . Замена $s(r)$ на r увеличивает длину решения на длину цикла C соответствующего r , таким образом все замены суммарно увеличивают размер решения на $\omega(C)$, который оценивается сверху по предыдущей лемме $\leq OPT$.

Лемма 3. Пусть c_1, c_2 это два цикла минимального покрытия C для графа префиксов и пусть $r(c_1)$ и $r(c_2)$ это представления для этих циклов, как определено выше. Тогда

$$ov(r(c_1), r(c_2)) < \omega(c_1) + \omega(c_2)$$

Перед тем как доказать эту лемму нам нужно сделать несколько технических определений и наблюдений. Для любого цикла $c = s_{i_1}, \dots, s_{i_k} \in C$ в префиксном графе S , введем следующее обозначение:

$$s(c) = pr(s_{i_1}, s_{i_2})pr(s_{i_2}, s_{i_3}) \dots pr(s_{i_{k-1}}, s_{i_k})pr(s_{i_k}, s_{i_1})$$

Обозначим за s^∞ строку вида $ssss\dots$

Лемма 4. Для любого цикла $c = s_{i_1}, \dots, s_{i_k}$ в префиксном графе для S , каждая s_{i_j} является подстрокой $[s(c)]^\infty$. Также, $r(c) = \langle s_{i_1}, \dots, s_{i_k} \rangle$ является подстрокой $[s(c)]^\infty$

Доказательство

Вспомним, что для любых s, t мы имеем $s = pr(s, t)ov(s, t)$, поэтому s является подстрокой (префиксом) $pr(s, t)t$. Таким образом s_{i_j} это префикс для строки $pr(s_{i_j}, s_{i_{j+1}})s_{i_{j+1}}$, которая в свою очередь является префиксом для $pr(s_{i_j}, s_{i_{j+1}})pr(s_{i_{j+1}}, s_{i_{j+2}})s_{i_{j+2}}$ и так далее. Продолжая таким же образом по циклу c мы доказываем утверждение для s_{i_j} . По тем же причинам доказательство работает для $r(c)$. Обратное утверждение также верно

Лемма 5. Если все строки $\hat{S} \subseteq S$ являются подстроками t^∞ , тогда существует цикл длины $|t|$ в графе префиксов для S , такой что он содержит все эти строки.

Доказательство.

Если строка s является подстрокой t^∞ , то она появляется в t^∞ каждые $|t|$ символов. Таким образом задается циклический порядок на строках из \hat{S} и легко видеть, что этот порядок дает цикл длины $|t|$ в префиксном графе.

продолжение доказательства леммы 3

Пусть $x, |x| \geq \omega(c_1) + \omega(c_2)$ это перекрытие $r(c_1)$ и $r(c_2)$. Тогда x является подстрокой $[s(c_1)]^\infty$ (поскольку $r(c_1)$ является) и $[s(c_2)]^\infty$ (поскольку $r(c_2)$ является). Пусть x_1 это префикс x длины $\omega(c_1)$ и x_2 это префикс x длины $\omega(c_2)$. Очевидно, x это префикс x_1^∞ и x_2^∞ . Теперь, поскольку $|x| \geq \omega(c_1) + \omega(c_2) = |x_1| + |x_2|$, мы имеем $x_1x_2 = x_2x_1$. Но это значит, что $x_1^kx_2^k = x_2^kx_1^k$ для любого k (по индукции), поэтому $x_1^\infty = x_2^\infty$

Любая строка в c_1 является подстрокой x_1^∞ (по лемме 4, поскольку x_1 является циклическим сдвигом $s(c_1)$), по этому она также является подстрокой x_2^∞ . Также любая подстрока в c_2 является подстрокой x_2^∞ . Таким образом по лемме 2, все строки в циклах c_1 и c_2 сожержатся в одном цикле длины $\omega(c_2)$, что противоречит предположению, что c_1 и c_2 являются минимальными циклическими покрытиями.

Из леммы 3 и леммы 1 вытекает следующее

Лемма 6. Суммарное перекрытие любого решения для множества представлений $R \leq 2OPT_S$

4-приближенный алгоритм для имплементации:

- 1) Для множества строк S построим множество представлений R
- 2) Вернем конкатенацию всех строк из R

Теорема 2. Описанный выше алгоритм является 4-приближенным алгоритмом

Доказательство. Рассмотрим оптимальное решение для множества R . Длина OPT_R решения $\leq 2OPT_S$. Также суммарное перекрытие этого оптимального решения $\leq 2OPT_S$

по лемме 5. Но общая длина $|R|$ строк в R в точности равняется сумме длин OPT_R оптимального решения и его суммарного перекрытия, поэтому $|R| \leq 4OPT_S$

4 приближенный алгоритм

Построим граф $G = (V, E)$, где $V = 1..n$, $E = \{(i, j, overlap(s_i, s_j)) | i, j = 1..n, i \neq j\}$ - последнее множество, это множество троек (начальная вершина, конечная вершина, вес). Затем для данного графа G найдем покрытие циклами минимального суммарного веса. Это и будет 4-приближенный алгоритм для задачи.

Вычислим жадное назначение для данного графа G

Будем хранить его в массиве из n чисел, обозначим его за A .

объявим все ребра незачеркнутыми

повторять пока остаются незачеркнутые ребра.

1) Выберем ребро (i, j) максимального веса среди незачеркнутых

2) Зачеркнем все ребра выходящие из i и входящие в j

3) $A[i] = j$

Наконец найдем покрытие минимального суммарного веса

0) Повторяем пока есть непосещенные вершины.

1) Возьмем вершину непосещенную вершину i . Отметим как посещенную. Положим $s = i$

2) while True:

Далее если $A[i]$ совпадает с s , то добавим цикл в результат, закончить цикл, перейти к пункту (0)

иначе $i = A[i]$

Запуска описанного алгоритма, проверка 2-приближенности жадного алгоритма для маленьких строк

Далее приведен Ipython notebook. К сожалению nbpdfconverter не поддерживает кириллицу, поэтому пояснения приведены на английском.

task45_Zhukov_vlad

November 16, 2016

0.0.1 See src.py, test.py files for algorithm code

```
In [50]: import src
import test
```

1 Let's test algorithm

See test.py

```
In [51]: import unittest
suite = unittest.TestLoader().loadTestsFromTestCase(test.Tests)
unittest.TextTestRunner(verbosity=2).run(suite)
```

```
test_strings_overlap (test.Tests) ... ok
```

```
-----
Ran 1 test in 0.001s
```

OK

```
Out[51]: <unittest.runner.TextTestResult run=1 errors=0 failures=0>
```

1.1 Example of usage

```
In [52]: strings = ['cde', 'abc', 'eab', 'fgh', 'ghf', 'hed']
algo = src.superstring4(strings)
print('superstring4: ' + str(algo.solve()))
print('greedy: ' + \
      str(src.greedy_min_max_contain_string(strings)))
print('answer: ' + \
      str(src.min_max_contain_string(set(strings))))
      #input must be set of strings
```

```
superstring4: cdeabcfghfhed
greedy:       cdeabchedfghf
answer:       fghfhedcdeabc
```

```
In [57]: from itertools import combinations
         from itertools import combinations_with_replacement
         from itertools import permutations
         from itertools import product

         def short_string_test(n_words, words):
             for c in combinations(words, n_words):
                 l1 = len(src.greedy_min_max_contain_string(c))
                 l2 = len(src.min_max_contain_string(set(c)))
                 res.append((1.0 * l1) / l2)
             return res
```

2 Let's "make sure" that approximation ratio is equal 2 for short strings(a.r.>=2)

It takes some time to find right answers

```
In [62]: letters = 'abcd'
         tmp = [map(''.join, product(letters, repeat=length))\
                 for length in range(1, 4)]
         words = [x for n in tmp for x in n]
         res = []
         sst = short_string_test(3, words)
         print(max(sst))
```

1.4

For sentences with two words, where words consist of 1, 2, 3 letters approximation ratio is $\sim \leq$
1.5

Let's implement test described in: <http://www.mimuw.edu.pl/~muchacha/teaching/aa2008/ss.pdf>
(2.2 The greedy algorithm)

```
In [44]: def generate_worst_test(k):
         return ['a' + 'b' * k, 'b' * k + 'c', 'b' * (k + 1)]
         test = generate_worst_test(4)
         res = list()
         for c in [generate_worst_test(i) for i in range(4, 300)]:
             l1 = src.greedy_min_max_contain_string(c)
             l2 = src.min_max_contain_string(set(c))
             res.append((1.0 * len(l1)) / len(l2))
         print(max(res))
```

1.99

So we can see on $\{ab^k, b^k c, b^{k+1}\}$ tests algorithm's approximation ratio converges to 2. We have a little bit better algorithm than in article(in article assumes that strings can not contain each other) that merges strings in one, if one contains another

Имплементация алгоритмов. src.py

```
from copy import deepcopy
from numpy import zeros, ix_, array, roll, argmax
#GREEDY
#####
#####2 approximation#####
#####
def strings_overlap(s1, s2):
    if (s1 in s2): return len(s1)
    if (s2 in s1): return len(s2)
    return max([i for i in range(min(len(s1), len(s2)) + 1) if s1[len(s1)-i:] == s2[:i]])
def strings_merge(s1, s2):
    if (s1 in s2): return s2
    if (s2 in s1): return s1
    overlap = strings_overlap(s1, s2)
    return s1 + s2[overlap:]
def greedy_min_max_contain_string(strings):
    s = {i for i in strings}
    pairs = [(i, j, strings_overlap(i, j)) for i in s for j in s if not i is j]
    while len(pairs) > 1:
        max_pair = max(pairs, key=lambda x: x[2])
        tmp = strings_merge(max_pair[0], max_pair[1])
        s.remove(max_pair[0])
        s.remove(max_pair[1])
        s.add(tmp)
        pairs = [(i, j, strings_overlap(i, j)) for i in s for j in s if not i is j]
    return [i for i in s][0]

def min_max_contain_string(strings):
    #strings ' must be set
    s = strings
    pairs = [(i, j, strings_overlap(i, j)) for i in s for j in s if not i is j]
    results = list()
    if len(s) == 1:
        return [i for i in s][0]
    for p in pairs:
        new_s = deepcopy(s)
        tmp = strings_merge(p[0], p[1])
        new_s.remove(p[0])
        new_s.remove(p[1])
        new_s.add(tmp)
        results.append(min_max_contain_string(new_s))
    return min(results, key=lambda x: len(x))

s_strings = {'hello', 'world', 'lol'}

def get_max_ind(ar):
    pos = argmax(ar)
    return (int(pos / ar.shape[1]), int(pos % ar.shape[1]))
#####
#####4 approximation#####
#####
class superstring4:
    def __init__(self, strings):
        self.n = len(strings)
        self.strings = strings
        self.ov = zeros((self.n, self.n))
        for i in range(self.n):
            for j in range(self.n):
                if i != j:
                    self.ov[i][j] = strings_overlap(strings[i], strings[j])
    def greedy_assignment(self):
        rows = list(range(self.n))
        cols = list(range(self.n))
        asg = zeros(self.n)
        while len(rows) and len(cols):
            mi = get_max_ind(self.ov[ix_(array(rows), array(cols))])
            mi = (rows[mi[0]], cols[mi[1]])
            asg[mi[0]] = mi[1]
```

```

        if mi[0] in rows:
            rows.remove(mi[0])
        if mi[1] in cols:
            cols.remove(mi[1])
    return asg
def get_cycle(self, asg, start, used):
    res = list()
    j = int(start)
    while True:
        used[j] = 1
        res.append(j)
        j = int(asg[j]) #avoid warning
        if j == start:
            break;
    return res
def greedy_cover(self):
    used = zeros(self.n)
    asg = self.greedy_assignment()
    res = list()
    for i in range(self.n):
        if not used[i]:
            res.append(self.get_cycle(asg, i, used))
    return res

#works only if second not contains in first(in the middle of the string)
def pref(self, s1, s2):
    assert(not s2 in s1)
    ol = max([i for i in range(min(len(s1), len(s2)) + 1) if s1[len(s1)-i:] == s2[:i]])
    return s1[:len(s1) - ol]
def merge(self, strings):
    assert(len(strings) > 0)
    res = ""
    last = strings[0]
    for s in strings[1:]:
        if (s in last):
            last = s
            continue
        res = res + self.pref(last, s)
        last = s
    res = res + strings[-1]
    return res
def rotate_max_cycle(self, cycle):
    c = array(cycle)
    res = list()
    for i in range(len(cycle)):
        res.append(self.merge(roll(c, i)))
    return min(res, key=lambda x: len(x))
def solve(self):
    cover = self.greedy_cover()
    res = ""
    for c in cover:
        s = [self.strings[int(i)] for i in c]
        res = res + self.rotate_max_cycle(s)
    return res

```

test.py

```

from src import strings_overlap, strings_merge
import unittest
pairs = [['aba', 'baaaa', 'abaaaa', 2],
         ['abracadabra', 'abracadabra', 'abracadabra', len('abracadabra')],
         ['asdd', 'ddqw', 'asddqw', 2]]
class Tests(unittest.TestCase):
    def test_strings_overlap(self):
        for p in pairs:
            self.assertTrue(strings_overlap(p[0], p[1]), p[3])
            self.assertTrue(strings_merge(p[0], p[1]), p[2])

```

Список литературы

- [1] J Gallant, D Maier, J Astorer *On finding minimal length superstrings*, Journal of Computer and System Sciences, 1980 - Elsevier
- [2] Jonathan S. Turner *APPROXIMATION ALGORITHMS FOR THE SHORTEST COMMON SUPERSTRING PROBLEM*, Computer Science Department Washington University, St. Louis
- [3] Marcin Mucha *A Tutorial on Shortest Superstring Approximation* December 17, 2007