Wyszukiwanie wzorca z wieloznacznikami i kilkoma błędami

Na podstawie "Pattern matching with don't cares and few errors" – Clifford, Efremenko, Porat, Rothschild[1]

Grzegorz Gawryał

Maj 2022

1 Wstęp

W problemie wyszukiwania wzorca z wieloznacznikami i kilkoma błędami dany jest tekst t długości n oraz wzorzec w, długości m, składające się ze słów nad alfabetem $\Sigma \cup \{?\}$. Dla każdej pozycji $i \in [n-m+1]$ chcemy stwierdzić, czy odległość Hamminga między słowami $t[i \dots i+m-1]$ oraz w wynosi co najwyżej k, przy czym symbol specjalny ? pasuje do każdego innego symbolu (w szczególności do ?). Praca korzysta z obserwacji poczynionych w pracy "Simple deterministic wildcard matching" P. Clifforda i R. Clifforda, podając główny algorytm randomizowany, a następnie modyfikując go na dwa sposoby, przyśpieszając go sprytniej wykorzystując dotychczasowe informacje oraz derandomizując go, stosując wyniki uzyskane dla znanego problemu z teorii informacji.

2 Algorytmy i dowody

2.1 Wynik pracy "Simple deterministic wildcard matching" P. Clifforda i R. Clifforda

W tej pracy autorzy rozważali problem wyszukiwania wzorca z wieloznacznikami i bez błędów ($k = 0$). Niech $f : \Sigma \cup \{?\} \to \{0, \dots, |\Sigma|\}$ będzie dowolną funkcją różnowartościową taką, że $f(?) = 0$. Autorzy zauważyli, że w pasuje do podsłowa $t[i \dots i+m-1]$ wtw, gdy

$$A_0(i) = \sum_{j=1}^{m} f(w_j) f(t_{i+j-1})(f(w_j) - f(t_{i+j-1}))^2 = 0$$

Można zauważyć, że ta suma będzie zerowa wtw, gdy każdy jej składnik będzie zerowy, czyli $f(w_j) = 0,\ f(t_{i+j-1}) = 0$, więc któryś z tych symboli jest wieloznacznikiem, lub $f(w_j) = f(t_{i+j-1})$, czyli symbole do siebie pasują.

Powyższą sumę dla każdego i można obliczyć poprzez klasyczny splot $(+, *)$, wykonując go na ciągach:

- $\Big(f(w_1)^3, \ldots, f(w_m)^3\Big)$ oraz $\Big(f(t_{i+m-1}), \ldots, f(t_i)\Big)$
- $\Big(-2f(w_1)^2, \ldots, -2f(w_m)^2\Big)$ oraz $\Big(f(t_{i+m-1})^2, \ldots, f(t_i)^2\Big)$
- $\Big(f(w_1), \ldots, f(w_m)\Big)$ oraz $\Big(f(t_{i+m-1})^3, \ldots, f(t_i)^3\Big)$

a następnie sumując otrzymane wektory. Całość zajmuje czas $O(n \log m)$, wykorzystując FFT.

2.2 Algorytm dla $k \leqslant 1$

Lekko zmodyfikujmy poprzednią sumę:

$$A_1(i) = \sum_{j=1}^{m} (i + j - 1) f(w_j) f(t_{i+j-1}) (f(w_j) - f(t_{i+j-1}))^2$$

Zauważmy, że jeśli w różni się na dokładnie jednej pozycji x od podsłowa $t[i \ldots i+m-1]$, to $A_0(i) = f(w_x) f(t_{i+x-1}) (f(w_x) - f(t_{i+x-1}))^2$ oraz $A_1(i) = (i+x-1) f(w_x) f(t_{i+x-1}) (f(w_x) - f(t_{i+x-1}))^2$. Dla każdego indeksu i zachodzi więc dokładnie jeden z przypadków:

- $A_0(i) = 0$ – wtedy odpowiednie słowa pasują do siebie bez błędów,
- $A_0(i) \neq 0$ – wówczas liczymy $B(i) = \frac{A_1(i)}{A_0(i)}$. Jeśli odpowiednie słowa różnią się na dokładnie jednej pozycji, to będzie to indeks $y = B(i)$. Sprawdzamy więc, czy $A_0(i) = f(w_y) f(t_{i+y-1}) (f(w_y) - f(t_{i+y-1}))^2$ – jeśli tak, to będzie to jedyny błąd, a jeśli nie, to znaczy, że mamy więcej błędów.

Analogicznie jak wyżej, wektor A_1 możemy obliczyć za pomocą 3 splotów, otrzymując tę samą złożoność. Powyższy algorytm może zwrócić wektor B mówiący, na której pozycji znajduje się jedyny błąd, o ile istnieje.

2.3 Większa liczba błędów

Główna ideą pracy jest zastępowanie niektórych symboli we wzorcu wieloznacznikami, a następnie uruchamianiu algorytmu dla $k \leqslant 1$. Wybierając odpowiednio dużą liczbę razy odpowiednią "maskę" dla wzorca, tzn. zbiór indeksów, które zamieniamy na wieloznacznik, będziemy otrzymywali indeksy różnych błędów, ostatecznie znajdując wszystkie, o ile dla danego offsetu jest ich co najwyżej k.

Pierwszy algorytm będzie po prostu losował niezależnie dla każdej pozycji $j \in [m]$, czy zastąpić symbol $w[j]$ wieloznacznikiem z prawdopodobieństwem $\frac{k-1}{k}$ (zamaskować go), czy go pozostawić. Ten proces powtórzony zostanie $\Theta(k \log n)$ razy, utrzymując dla każdego offsetu i znalezione pozycje błędów $E(i)$ (np. w hashsecie, który utrzymuje zbiory wielkości co najwyżej k). Na końcu należy jeszcze wykonać proces sprawdzenia – aby stwierdzić, czy znalezione zostały wszystkie błędy, wystarczy obliczyć tablicę A_0 dla niezamaskowanego wzorca w i sprawdzić, czy

$$A_0(i) = \sum_{j \in E(i)} f(w_j) f(t_{i+j-1})(f(w_j) - f(t_{i+j-1}))^2$$

Powyższa równość będzie zachodzić wtw, gdy w zbiorze E znajdą się wszystkie błędy dla offsetu i.

Aby obliczyć prawdopodobieństwo pomyłki algorytmu, załóżmy, że dla offsetu i mamy $d \leqslant k$ błędów. Dla każdego z tych błędów prawdopodobieństwo, że go znajdziemy w jednej iteracji wynosi $\frac{1}{k}(\frac{k-1}{k})^{d-1} \geqslant \frac{1}{k}(1 - \frac{1}{k})^k \geqslant \frac{1}{k}e^{-\frac{1}{k}k} = \frac{1}{ke}$. Zatem, prawdopodobieństwo, że go nie znajdziemy nigdy, wynosi co najwyżej $(1-\frac{1}{ke})^{\Theta(k \log n)} \leqslant e^{-\frac{1}{ke}\Theta(k \ln n)} = (e^{\ln n})^{(-c)} = n^{-c}$, dla pewnej stałej c. Wykorzystując union bound, sumując po wszystkich $nk \leqslant n^2$ błędach, prawdopodobieństwo pomyłki algorytmu będzie dalej w postaci n^{-c}.

Złożoność całego algorytmu to $\Theta(k \log n n \log m) = \Theta(nk \log n \log m)$ – główna część algorytmu jest w tej złożoności, a sprawdzenie na końcu zajmuje $O(n \log m + nk)$. W repozytorium nosi on nazwę `nonrecursive_randomised`. Algorytm przy przy pechowym losowaniu może zwrócić niepoprawną odpowiedź oraz autorzy nie podają metody, jak zweryfikować poprawność wyników, zachowując ten sam czas działania (jeśli dla danego offsetu liczba błędów jest większa niż k, to prawdopodobieństwo zamaskowania wszystkich błędów poza jednym spada).

2.4 Szybsza wersja – algorytm rekurencyjny

W poprzednim algorytmie w kolejnych iteracjach nie wykorzystujemy faktu, że znamy już niektóre błędy i możemy je poprawić w algorytmie dopasowania

z jednym błędem.

Algorytm będzie działał rekurencyjnie w $\log k$ wywołaniach i w każdym wywołaniu, od $s = 0$ do $\lfloor \log k \rfloor$ będziemy chcieli zmniejszyć liczbę pozostałych do wykrycia błędów podwójnie. W fazie s ustalimy prawdopodobieństwo pozostawienia symbolu na $\frac{1}{k_s} = \frac{2^s}{k}$ i przez $\Theta(\log n + k_s)$ iteracji wykonamy to samo, co w poprzednim algorytmie, z wyjątkiem tego, że w algorytmie dla $k \leqslant 1$ od $A_0(i)$ oraz $A_1(i)$ odejmiemy składniki pochodzące od pozycji znanych już błędów.

Pozycje znanych błędów będziemy trzymać w dodatkowej tablicy hash-setów $E'[1 \dots m]$ takiej, że $E'[j]$ zawiera pozycje i takie, że w oraz $t[i \dots i + m - 1]$ nie pasują do siebie na pozycji j. Łącznie w E' będzie co najwyżej nk elementów, ponieważ dla każdej pozycji i pamiętamy co najwyżej k błędów, czyli w E', które jest komplementarne do E z poprzedniego algorytmu będzie ich tyle samo.

Poprawianie tablic w algorytmie 1 będzie wyglądać następująco: dla każdej pozycji $j \in [m]$ sprawdzimy, czy w danej iteracji zewnętrznego algorytmu zamaskowana została pozycja j (czyli $w'[j] =?$) i wyłącznie jeżeli nie, to wtedy przeiterujemy się po $i \in E'[j]$, poprawiając $\forall i \in E'[j]$: $A_0(i)$ `-=` $f(w_j)f(t_{i+j-1})(f(w_j) - f(t_{i+j-1}))^2$ oraz A_1 w analogiczny sposób.

Pesymistycznie takie poprawianie zajmie czas $O(m+nk)$, ale korzystając z tego, że maskujemy część pozycji losowo, pokażemy, że w oczekiwaniu zajmie to czas $O(m + \frac{nk}{k_s})$. Ustalmy fazę s i konkretną iterację w niej. Niech $X_{j,i}$ będzie zmienną losową oznaczającą, że w poprawianiu uwzględnimy błąd na pozycji j we wzorcu dla offsetu i w tekście. Oczekiwany czas poprawiania wynosi $\mathbb{E}[\sum X_{j,i}] \leqslant nk\mathbb{E}[x_{j,i}] = nk\frac{1}{k_s}$. Korzystając z nierówności Chernoffa można pokazać, że taka liczba porównań zachodzi nie tylko w oczekiwaniu, ale również z dużym prawdopodobieństwem.

Dla fazy s wykonamy łącznie średnio $\Theta((\log n+k_s)(\frac{nk}{k_s}+n\log m))$ operacji, zatem łącznie będzie ich średnio:

$$\Theta\Big(\sum_{s=0}^{\log k}(\log n + k_s)\Big(\frac{nk}{k_s} + n\log m\Big)\Big) = \Theta\Big(\sum_{s=0}^{\log k}\Big(\log n + \frac{k}{2^s}\Big)\Big(\frac{nk2^s}{k} + n\log m\Big)\Big) =$$
$$= \Theta\Big(\sum_{s=0}^{\log k} n2^s\log n + nk + n\log m\log n + n\log m\frac{k}{2^s}\Big) =$$
$$= \Theta\Big(nk\log n + nk\log m\log k\Big)$$

W celu zakończenia dowodu należy wykazać, że prawdopodobieństwo zwrócenia niepoprawnego wyniku przez jest małe. Dla uproszczenia załóżmy, że dla każdego offsetu i mamy dokładnie k błędów. Wystarczy udowodnić,

że zgodnie z tym, co było napisane wyżej, w fazie s znajdziemy $\frac{k_s}{2}$ nowych błędów.

Ustalmy offset i. Prawdopodobieństwo, że w pojedynczej iteracji znajdziemy ustalony błąd (którego nie wykryliśmy w poprzednich fazach – te błędy usuwamy w fazie poprawiania) wynosi, podobnie jak w poprzednim algorytmie co najmniej $\frac{1}{ek}$, zatem prawdopodobieństwo znalezienia jakiegoś, z union bounda, to co najmniej $\frac{1}{e}$. Dalej skorzystamy z następującej nierówności:

Lemma 2.1 (Nierówność Chernoffa)**.** Dla $\mu = \mathbb{E}[Z_\iota]$ oraz niezależnych zmiennych Z_ι o tym samym rozkładzie zachodzi:

$$\Pr\left(\frac{Z_1 + \cdots + Z_l}{l} \leqslant (1-\delta)\mu\right) \leqslant e^{-l\mu\delta^2/2}$$

Dla zmiennych $Y_1, \ldots, Y_{\Theta(\log n + k_s)}$ takich, że Y_ι jest zmienną wskaźnikową mówiącą, czy w danej iteracji znaleźliśmy jakiś błąd (być może znaleziony w jakiejś innej iteracji w tej samej fazie) oraz dla $\delta = \frac{1}{2}$ mamy $\mu = \frac{1}{e}$ mamy, że po t iteracjach

$$\Pr\left(Y_1 + \cdots + Y_l \leqslant \frac{t}{2e}\right) \leqslant e^{-t/8e}$$

Czyli, z dużym prawdopodobieństwem znajdziemy co najmniej $\frac{t}{2e}$ błędów, niekoniecznie różnych. Dalej pokażemy, że co najmniej połowa z powyższych znalezionych błędów jest różna. Przypuśćmy, że znalezione błędy będą tylko ze zbioru $\frac{k_s}{2}$ pozycji. Szansa, że tak będzie to co najwyżej $2^{-t/2e}$. Prawdopodobieństwo, że znajdziemy mniej niż $\frac{k_s}{2}$ różnych błędów to co najwyżej $2^{-t/2e}\binom{k_s}{k_s/2}$, czyli prawdopodobieństwo zdarzenia przeciwnego jest duże, tzn. w postaci $1 - n^c$.

2.5 Algorytm deterministyczny

Wróćmy do głównego algorytmu, w złożoności $\Theta(nk \log n \log m)$. Zamiast losować maski dla wzorca, możemy wcześniej wygenerować rodzinę pozostawionych indeksów spoza maski $\mathcal{F} = \{F_1, \ldots, F_s\}$ i wykorzystać ją w algorytmie, wykonując s iteracji. Jeśli odpowiednio utworzymy $\mathcal{F}$, to będziemy w stanie zagwarantować poprawność algorytmu.

Chcemy, aby dla każdego offsetu $i \in [n-m+1]$, niezależnie od tego, gdzie znajdują się błędy (zakładając, że jest ich co najwyżej k), istniały zbiory w $\mathcal{F}$ takie, że dla każdego błędu maskujemy wszystkie pozycje oprócz niego.

Będzie tak, jeśli dla każdego zbioru pozycji błędów $A = \{a_1, \ldots, a_d\} \subseteq [m]$, gdzie $d \leqslant k$, oraz dla każdego elementu $a_x \in A$ w $\mathcal{F}$ istniał zbiór pozostawionych indeksów F_y taki, że $F_y \cap A = \{a_x\}$, czyli zamaskowane zostaną wszystkie błędy poza a_x.

Okazuje się, że istnieje deterministyczny algorytm tworzenia takiej rodziny silnie wybieralnych zbiorów $\mathcal{F}$ (strongly selective family), tworzący rodzinę wielkości $\Theta(\min(m, k^2 \log m))$ w czasie $\Theta(km \log m)$, opisany w pracy [2]. Pozwoli on rozwiązać nasz problem deterministycznie w $\Theta(km \log m + \min(m, k^2 \log m)n \log m) = \Theta(nk^2 \log^2 m)$

Appendix A: Tworzenie rodziny silnie wybieralnych zbiorów

Załóżmy, że dane są m, k i chcemy utworzyć rodzinę $\mathcal{F}$ spełniającą warunki z poprzedniego rozdziału. Okazuje się, że problem jest silnie powiązany z kodami korekcji błędów (error correction codes - ECC).

Dla $\text{ECC}(n, l, d)_q$ chcemy umieć kodować w sposób jednoznaczny słowa długości l w słowa długości n, oba nad alfabetem wielkości q tak, aby po przejściu przez zaszumiony kanał, który może zmienić znaki na co najwyżej d pozycjach, otrzymane słowo dało się poprawnie odkodować.

Załóżmy, że mamy $\text{ECC}(n, \log_q m, \delta n)_q$. Mamy więc $q^{\log_q m} = m$ różnych słów, które można zakodować. Oznaczmy kody tych słów jako $w_1, \ldots, w_m$. Pokażemy, że poniższy algorytm pozwala z utworzyć rodzinę $\mathcal{F}$ o parametrach $m, \lceil \frac{1}{1-\delta} \rceil$, wielkości nq:

Require: $\text{ECC}(n, \log_q m, \delta n)_q$, codewords $w_1, \ldots, w_m$, each of length n
Ensure: $\text{SSF}(m, \lceil \frac{1}{1-\delta} \rceil)$
  for $i = 1$ **to** m **do**
    for $p = 1$ **to** n **do**
      insert i into $F_{p, w_i[p]}$
    end for
  end for

W sumie będziemy mieć co najwyżej nq różnych zbiorów F, więc wielkość outputu się zgadza. Aby pokazać, że zbiory F mają wymaganą własność, niech $r = \lceil \frac{1}{1-\delta} \rceil$ oraz $A = \{i_1, \ldots, i_r\}$ będą dowolnymi różnymi indeksami w $[m]$. Bso wystarczy pokazać, że istnieje zbiór F', który wybiera i_1, tzn $F' \cap A = \{i_1\}$. Z własności kodu, dla dowolnego $j \neq 1$ liczba pozycji, na których słowa w_{i_1} oraz w_{i_j} się zgadzają, to co najwyżej $(1 - \delta)n$ (w przeciwnym wypadku odległość Hamminga po zakodowaniu byłaby mniejsza i słowa byłyby nie do rozróżnienia po przesłaniu przez kanał). Sumując po wszystkich takich j, liczba pozycji, na których w_{i_1} zgadza się z którymś z wi_j to co najwyżej

$(r-1)(1-\delta)m < m$, więc istnieje pozycja p, na której to słowo się różni od wszystkich pozostałych. Zatem, tym zbiorem F' będzie $F_{p,w_{i_1}[p]}$.

W ECC często stosuje się kody liniowe – zakładamy, że alfabetem jest $\mathbb{F}_q$, słowa są wektorami nad tym ciałem, a funkcja kodowania G jest macierzą nad $\mathbb{F}_q$ o rozmiarach $n \times l$, tzn zakodowane słowo y to Gy. Dla kodów linowych istnieje ważne ograniczenie Gilberta-Varshamova (GV bound), które mówi jak duże musi być n w stosunku do l, aby dla danego d skonstruowanie takiej macierzy było możliwe. To ograniczenie jest dokładne i okazuje się, że wystarczy losować macierz G o wymiarach wyznaczonych przez to ograniczenie (każdą komórkę jednostajnie i niezależnie z $\mathbb{F}_q$), i z dużym prawdopodobieństwem będzie ona umożliwiała poprawne kodowanie, przesyłanie przez kanał oraz dekodowanie.

W dowodzie tego, że losowa macierz G będzie dobra, pokazujemy, że z dużym prawdopodobieństwem dla każdego słowa kodowego w_i prawdopodobieństwo, że będzie miało ono nie więcej niż δn zer jest duże, na podstawie czego można udowodnić, że kodowanie spełnia wymogi.

W celu derandomizacji tej metody wystarczy zamiast losowania po kolei, idąc po wierszach, elementy macierzy G, za każdym razem wybierać element $x \in \mathbb{F}_q$, który minimalizuje wartość oczekiwaną liczby złych wydarzeń, czyli liczby słów kodowych, które będą miały więcej niż δn zer. Początkowo ta wartość oczekiwana jest mniejsza niż 1, dlatego po wykonaniu ostatniego kroku będzie również musiała taka być, czyli będzie wynosić 0. Sam algorytm nie jest bardzo trudny, ale w celu jego efektywnej implementacji i uzyskania pożądanej złożoności trzeba uważnie wczytać się w szczegóły techniczne pracy[2], dlatego nie został on tutaj przedstawiony.

Literatura

[1] R. Clifford, K. Efremenko, E. Porat, A. Rothschild, Pattern matching with don't cares and few errors, Journal of Computer and System Sciences, Volume 76, Issue 2, 2010, Pages 115-124, ISSN 0022-0000, https://doi.org/10.1016/j.jcss.2009.06.002.

[2] E. Porat and A. Rothschild, "Explicit Nonadaptive Combinatorial Group Testing Schemes," in IEEE Transactions on Information Theory, vol. 57, no. 12, pp. 7982-7989, Dec. 2011, doi: 10.1109/TIT.2011.2163296.