INSTYTUT BADAŃ SYSTEMOWYCH
POLSKIEJ AKADEMII NAUK

TECHNIKI INFORMACYJNE TEORIA I ZASTOSOWANIA

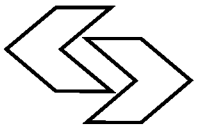
Wybrane problemy
Tom 1(13)

poprzednio

ANALIZA SYSTEMOWA W FINANSACH I ZARZĄDZANIU

Pod redakcją
Jerzego HOŁUBCA

Warszawa 2011



INSTYTUT BADAŃ SYSTEMOWYCH
POLSKIEJ AKADEMII NAUK

TECHNIKI INFORMACYJNE TEORIA I ZASTOSOWANIA

Wybrane problemy
Tom 1(13)

poprzednio

ANALIZA SYSTEMOWA W FINANSACH I ZARZĄDZANIU

Pod redakcją
Jerzego HOŁUBCA

Warszawa 2011

Wykaz opiniodawców artykułów zamieszczonych
w niniejszym tomie:

Dr hab. inż. Przemysław GRZEGORZEWSKI, prof. PAN

Prof. dr hab. inż. Jerzy HOŁUBIEC

Dr inż. Tatiana JAWORSKA

Dr hab. inż. Wiesław KRAJEWSKI, prof. PAN

Dr hab. inż. Maciej KRAWCZAK, prof. PAN

Dr hab. Michał MAJSTEREK

Dr hab. inż. Andrzej MYŚLIŃSKI, prof. PAN

Prof. dr hab. inż. Witold PEDRYCZ

Dr hab. inż. Ryszard SMARZEWSKI, prof. KUL

Prof. dr hab. inż. Andrzej STRASZAK

Dr Dominik ŚLĘZAK

Prof. dr hab. inż. Stanisław WALUKIEWICZ

ISBN 9788389475336

METODY PRZETWARZANIA DANYCH POMIAROWYCH DO POSTACI ŹRÓDŁOWEJ

Paweł Pylak
Studia Doktoranckie IBS PAN
Katolicki Uniwersytet Lubelski Jana Pawła II

In this article we present two algorithms transforming preprocessed real data acquired from ADC to the source integer form. Such a transformation permits one to achieve significant reduction of the space required to store the data. The average time complexity of the algorithms is $O(n \log_2 n)$.

Key words*: data compression algorithms, lossless compression*

1. Wprowadzenie

W zagadnieniu bezstratnej kompresji danych przemysłowych, czyli danych pochodzących z różnych czujników zamontowanych na urządzeniach, w pomieszczeniach, itp., zdarza się, że dane te występują w postaci liczb rzeczywistych. Może to stanowić punkt wyjścia do wstępnej eliminacji nadmiarowości. Najczęściej dane te są pozyskiwane z przetworników analogowo-cyfrowych (AC), które poziom napięcia z czujnika próbkują i przekształcają na postać cyfrową (digitalizują). Tak otrzymane wartości są w postaci liczb całkowitych 2 bajtowych (przetworniki 16-bitowe) lub 3 bajtowych (przetworniki 24-bitowe). Jednak sterowniki kart akwizycji danych lub oprogramowanie wyższego poziomu te surowe dane przekształcają w dane prezentujące konkretne parametry, np. przyśpieszenie, ciśnienie, temperaturę, itd. wyrażone w odpowiednich jednostkach. Takie dane są już najczęściej w postaci liczb rzeczywistych 4 lub 8-bajtowych (w języku C++: float lub double). Autor miał do czynienia z taką sytuacją podczas diagnostyki przekładni mechanicznych, gdzie olbrzymie ilości danych diagnostycznych pochodzących pierwotnie z przetworników AC zapisywane były przez oprogramowanie w postaci zbiorów liczb rzeczywistych 8-bajtowych (double).

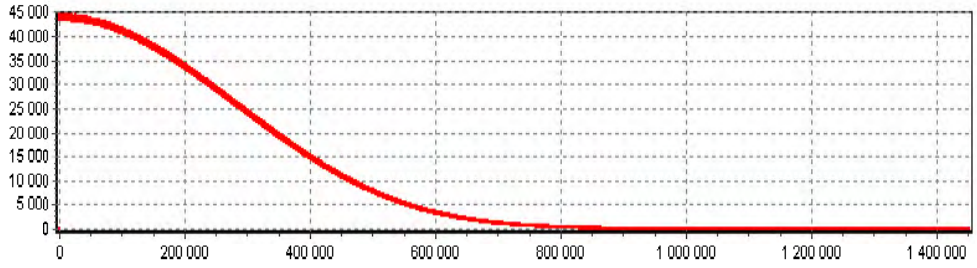
W projekcie tym zarówno karty, jak i oprogramowanie akwizycyjne dostarczała jedna z najbardziej liczących się na rynku firm międzynarodowych.

Celem tego artykułu jest zaproponowanie i przedstawienie algorytmów dających możliwość przekształcenia takich danych rzeczywistych z powrotem na ich pierwotną postać całkowitoliczbową oraz na mały zbiór współczynników, który w przyszłości pozwoli ten proces odwrócić. Należy tu na wstępie nadmienić, że dane przetworzone na postać całkowitoliczbową i z powrotem nie są identyczne binarnie ze zbiorem wejściowym, ale są identyczne logicznie. Dzieje się tak, gdyż dane te są poddawane obróbce numerycznej, a występujące przy tym błędy zaokrągleń mogą wprowadzać pewne niedokładności na mało znaczących pozycjach dziesiętnych, co oczywiście czyni te liczby różnymi w sensie zapisu binarnego.

2. Algorytmy

Wszystkie omówione tu algorytmy wymagają do pracy danych zgodnych w sensie struktury z rzeczywistymi danymi przemysłowo-diagnostycznymi, czyli dane te powinny mieć:

- odpowiednią liczebność – w zależności od rozdzielczości przetwornika AC (od ilości unikalnych wartości),
- wykres rozkładu różnic wartości unikalnych powinien być w okolicy zera stosunkowo wysoko ponad osią i powinien maleć wraz z oddalaniem się od zera. Czyli im mniejsza różnica tym jest wyższe prawdopodobieństwo jej wystąpienia. Przykład łamanej częstości [3, s. 8] takiego rozkładu dla pewnego realnego zbioru danych (200000 próbek) jest pokazany na Rys. 1.



Rys. 1. Przykład łamanej częstości rozkładu różnic wartości unikalnych

2.1. Algorytm 1

Pierwszy z algorytmów ma za zadanie dla podanego ciągu $\{y_i, i = 0,\dots,N-1\}$ liczb rzeczywistych znaleźć, o ile jest to możliwe, takie liczby $a \in \mathbb{R}_+, b \in \mathbb{R}, x_i \in \mathbb{N}$, że prawdziwe będą równania

$$y_i = ax_i + b, i = 0,\dots,N-1.$$

Dodatkowo, żądamy, aby liczby x_i były najmniejsze z możliwych.

Sposób działania Algorytmu 1

Najpierw, aby zoptymalizować operacje, bez zmniejszenia ogólności rozważań, poddaje się dane sortowaniu i eliminacji powtórek. Otrzymuje się nowy ciąg liczb $\{y'_i, i = 0,\dots,M-1\}, M \le N, y'_j < y'_k, j < k,$ dla którego przyporządkowujemy ciąg szukanych x'_i takich, że

$$y'_i = ax'_i + b, i = 0,\dots,M-1.$$

Po sprawdzeniu podstawowego warunku na liczbę elementów w otrzymanym ciągu (>=2), zakłada się, dla uproszczenia obliczeń, że $x'_0 = 0$, czyli, że $b = y'_0$. Następnie szuka się najmniejszej co do wartości bezwzględnej różnicy $y'_j - y'_k, j,k \in \{0,\dots,M-1\}, k < j,$ która z dużym prawdopodobieństwem będzie różnicą pewnej pary *y*-ów dla sąsiadujących *x*-ów lub przynajmniej stosunkowo bliskich *x*-ów. Tak więc niech ta różnica równa się

$$dy' = y'_j - y'_k = ax'_j + b - ax'_k - b = a(x'_j - x'_k).$$

dla pewnych $j,k \in \{0,\dots,M-1\}, k < j$, stąd

$$a = \frac{dy'}{x'_j - x'_k} = \frac{dy'}{z'}.$$

Teraz wiedząc, że $z' = x'_j - x'_k$ jest liczbą naturalną i powinno być stosunkowo małe podstawia się za z' kolejne liczby naturalne od 1 do pewnej z góry określonej granicy *P*:

$$a_p = \frac{dy'}{p}, p = 1,\dots,P, \qquad (1)$$

Dla każdej z tych wartości $i' = p$ i wyliczonego a_p sprawdza się czy prawdziwe jest

$$x'_i = \frac{y'_i - b}{a_p} \in \mathbb{N}, i = 1, \ldots, M-1.$$

Jeśli tak, to szukane $a = a_p$ i algorytm jest przerywany.

W praktyce, jeśli a_p nie jest szukanym a, to bardzo szybko – już dla początkowych *i* okazuje się, że $\frac{y'_i - b}{a_p} \notin \mathbb{N}$.

Zauważmy, że:

Twierdzenie 1

Znalezione przez Algorytm 1 rozwiązania $x'_i, i = 0, \ldots, M-1$ są najmniejszymi możliwymi naturalnymi rozwiązaniami.

Konkludując można wysunąć wniosek, że podany tu algorytm ma złożoność obliczeniową O(n)+O(p), gdzie p zależy nie od ilości danych, ale od specyfiki źródła danych. Dla pewnych danych ze wspominanego projektu była to np. liczba 138. Dodatkowo powinno się tu uwzględnić złożoność $O(n \log_2 n)$ wstępnego sortowania.

2.2. Algorytm 2

Drugi z przedstawionych w tym artykule algorytmów ma za zadanie dla danego posortowanego ciągu parami różnych liczb całkowitych $\{y_i : i = 0, \ldots, M-1\}$ znaleźć, o ile jest to możliwe, takie liczby $a \in \mathbb{R}_+, a > 3, b \in \mathbb{R}, x_i \in \mathbb{N}$, że prawdziwe będą równania

$$y_i = \lfloor ax_i + b \rfloor, \; i = 0, \ldots, M-1. \tag{2}$$

Założenie $a > 3$ wynika z faktu, że w celu poprawy wydajności algorytm posługuje się pewnymi uproszczeniami w części związanej z wyznaczaniem x_i. Dla $a \in (2; 3]$ algorytm działa w jakichś 70% przypadków testowych zbiorów danych. Natomiast w pozostałych 30% przypadków oraz dla wartości $a \in (1; 2]$ potrzebny jest inny algorytm o prawdopodobnie większej złożoności obliczeniowej $O(n^2)$. Jest on jeszcze w obszarze badań autora. Dla $a \in (0; 1)$ funkcja (2) przestaje być odwracalna.

Dane, co do których można zastosować ten algorytm mogą powstać w co najmniej dwóch sytuacjach:

a) Dane źródłowe są skalowane, a następnie zapisywane jako liczby całkowite, gdyż np. twórca systemu uznał, że części ułamkowe niewiele wnoszą do diagnostyki;
b) Do obliczeń (skalowania i przesuwania sygnału) w urządzeniu lub sterowniku użyto liczb rzeczywistych w zapisie stałoprzecinkowym. Takie dane wymagają wówczas zarówno pierwszego, jak i drugiego algorytmu do odwrócenia obliczeń. Z taką sytuacją spotkał się autor we wspomnianym projekcie. W tym przypadku a z reguły jest duże (> 100).

Sposób działania Algorytmu 2

Pierwsza część algorytmu służy do wyznaczenia $x_i \in \mathbb{N}$. W tym celu, jak poprzednio, po sprawdzeniu podstawowego warunku na liczbę elementów w otrzymanym ciągu (>=2), szukamy najmniejszej co do wartości bezwzględnej różnicy $y_j - y_k, j, k \in \{0, \ldots, M-1\}, k < j$, która z bardzo dużym prawdopodobieństwem będzie różnicą pewnej pary *y*-ów dla sąsiadujących *x*-ów. Znaleziona wartość

$$dy_{min} = y_{j'} - y_{k'},$$

dla pewnych $j', k' \in \{0, \ldots, M-1\}, k' < j'$, pozwala zatem na wstępne przybliżone oszacowanie współczynnika a. Przy założeniu, że

$$x_{j'} - x_{k'} = 1, \tag{3}$$

mamy

$$a \cong \frac{y_{j'} - y_{k'}}{x_{j'} - x_{k'}} = \frac{dy_{min}}{1} = dy_{min},$$

Następnie sprawdzamy czy wśród różnic $y_j - y_k, j, k \in \{0, \ldots, M-1\}, k < j$, występuje różnica o wartości $dy_{min1} = dy_{min} + 1$. Jeśli tak, to startowy przedział wartości a definiujemy w następujący sposób:

$$(a^0_{min}; a^0_{max}) = (dy_{min}; dy_{min} + 1).$$

W przeciwnym razie zbiór ten musimy zdefiniować jako trochę szerszy:

$$(a^0_{min}; a^0_{max}) = (dy_{min} - 1; dy_{min} + 1),$$

Wynika to z własności funkcji *floor* [2].

W głównej pętli algorytmu staramy się znaleźć takie pierwsze y_r, przy pomocy którego możliwe będzie wyznaczenie wartości x_i dla wszystkich $y_i, i \in \{0, \ldots, M-1\}$.

A mianowicie dla pewnego ustalonego chwilowo y_r próbujemy poruszając się w obie strony ($y_j, j = r+1, \ldots, M-1$ i $y_m, m = r-1, \ldots, 0$) i zawężając w kolejnych iteracjach dopuszczalny zbiór wartości parametru a, tj. przedział $(a^i_{min}; a^i_{max}) \subseteq (a^{i-1}_{min}; a^{i-1}_{max})$, wyznaczać wartości x dla kolejnych y. W *i*-tej iteracji jeśli prawdziwe jest wyrażenie logiczne $(j < M) \wedge \left((m < 0) \vee (y_r - y_m \geq y_j - y_r)\right)$, to wybieramy do przetwarzania element y_j, a w przeciwnym wypadku y_m. Jeśli wybrany został y_j, to wykonujemy następujące operacje:

a) Jeśli $(y_j - y_{j-1} = dy_{min}) \vee (y_j - y_{j-1} = dy_{min1})$, to $x_j = x_{j-1} + 1$ i przechodzimy do punktu d).
b) Niech $x_{min} = \left\lceil \frac{y_j - y_r}{a^{i-1}_{max}} \right\rceil$, $x_{max} = \left\lfloor \frac{y_j - y_r}{a^{i-1}_{min}} \right\rfloor$. Jeśli $x_{min} = x_{max}$, to $x_j = x_{min}$ i przechodzimy do punktu d).
c) Jeśli nieprawdziwe były wyrażenia logiczne z punktów a) i b), to oznacza, że *r* jest niewłaściwe. Wówczas należy zwiększyć r o jeden i przejść do kolejnej iteracji głównej pętli;
d) Aktualizujemy przedział dopuszczalnych wartości a:

$$(a^i_{min}; a^i_{max}) = \left(\max\left(\frac{y_j - y_r - 1}{x_j}, a^{i-1}_{min}\right); \min\left(\frac{y_j - y_r + 1}{x_j}, a^{i-1}_{max}\right)\right).$$

Zwiększamy wartość *j* o jeden.

W przypadku y_m .operacje są analogiczne.

Jeśli dla pewnego *r* w wewnętrznej iteracji doszliśmy do momentu, że $(j \geq M) \wedge (m < 0)$, wtedy oznacza to fakt znalezienia wartości x_i dla wszystkich $y_i, i \in \{0, \ldots, M-1\}$. W tym momencie x_i dla $i < r$ będą ujemne, więc (bez żadnych konsekwencji) powinniśmy przesunąć wszystkie x_i o x_0:

$$x'_i = x_i - x_0, i = 0, \ldots, M-1,$$

W ten sposób zbiór $x'_i, i \in \{0, \ldots, M-1\}$ jest rozwiązaniem zadania postawionego przed algorytmem.

Jeśli dla żadnego *r* wewnętrzna iteracja nie zakończyła się sukcesem, wtedy albo dane nie spełniają zależności (2) albo są za mało precyzyjne, aby można było stosować powyższy algorytm. Jednak we wszystkich znanych autorowi rzeczywistych przypadkach zbiorów danych diagnostycznych nigdy taka sytuacja nie nastąpiła.

Uwaga: jeśli warunek (3) nie jest spełniony (co dla rozważanych w artykule zbiorów danych jest bardzo mało prawdopodobne), wtedy można zamiast $x_{j'} - x_{k'} = 1$ przyjąć, że $x_{j'} - x_{k'} = 2, 3, \ldots$ i po niewielkich korektach ponowić wszystkie obliczenia.

W ten sposób zakończyliśmy pierwszą część prezentowanego algorytmu.

Pozostało jeszcze wyznaczenie współczynników a oraz b. Z uwagi na (2) omawiane zagadnienie można przedstawić jako układ nierówności:

$$y_i \leq ax_i + b < y_i + 1, i = 0, \ldots, M-1. \tag{4}$$

Może to sugerować wykorzystanie metod programowania liniowego [1], np. metody simpleks do rozwiązania naszego zagadnienia. Jednakże, ze względu na dużą liczbę nierówności (rzędu kilkuset tysięcy, a nawet milionów) oraz dużą złożoność obliczeniową i pamięciową tych metod zrezygnowano w prezentowanym algorytmie z użycia klasycznych metod na rzecz algorytmu uwzględniającego specyfikę zagadnienia.

Na wstępie rozpatrzmy dwie nierówności spośród (4):

$$y_k \leq ax_k + b < y_k + 1 \text{ oraz } y_n \leq ax_n + b < y_n + 1$$

dla dowolnych $n, k, n < k$. Rozkładając na pojedyncze nierówności mamy

$$y_k \leq ax_k + b, \tag{5}$$

$$ax_k + b < y_k + 1, \tag{6}$$

$$y_n \leq ax_n + b, \tag{7}$$

$$ax_n + b < y_n + 1. \tag{8}$$

Teraz dodając stronami (5) z ((8) oraz (6) z (7) otrzymujemy

$$y_k + ax_n + b < ax_k + b + y_n + 1 \quad \text{oraz} \quad ax_k + b + y_n < y_k + 1 + ax_n + b,$$

stąd:

$$a > \frac{y_k - y_n - 1}{x_k - x_n} \quad \text{oraz} \quad a < \frac{y_k - y_n + 1}{x_k - x_n}. \tag{9}$$

Na podstawie (9) zdefiniujmy przedział dopuszczalnych wartości a wyznaczony z nierówności $n, k, n < k$:

$$A_{n,k} = \left(a_{min}^{n,k}; a_{max}^{n,k}\right) = \left(\frac{y_k - y_n - 1}{x_k - x_n}; \frac{y_k - y_n + 1}{x_k - x_n}\right).$$

Okazuje się, że:

Twierdzenie 2

Elementy zbioru

$$A = \bigcap_{k=0}^{M-1} \bigcap_{n=0}^{k-1} A_{n,k},$$

są współczynnikami a spełniającymi (4), a więc i (2). Innymi słowy, dla każdego $a \in A$ istnieje takie $b \in \mathbb{R}$, że spełnione są nierówności (4).

Następnie dla ustalonego współczynnika a określamy granice dla współczynnika b. Na podstawie (4) mamy

$$b_{\min}(a) = \max_{i=0,\dots,M-1} y_i - ax_i,$$

$$b_{max}(a) = \min_{i=0,\dots,M-1} y_i - ax_i + 1.$$

Zatem poszukiwane b dla danego a spełnia nierówność

$$b_{\min}(a) \leq b < b_{max}(a). \tag{10}$$

Jednak z uwagi na fakt, że wyznaczenie przedziału A wymaga $O(n^2)$ operacji, a w praktyce wystarczy znalezienie jednego dowolnego $a \in A$ i odpowiadającego mu jednego b, stąd nie będziemy wyznaczać całego A, a do

wyznaczenia poszukiwanych współczynników a i b zastosujemy poniższą metodę.

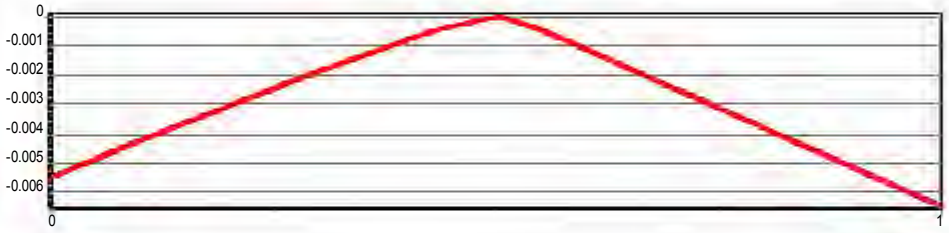
Zaczynamy od wstępnego oszacowania przedziału dopuszczalnych wartości a:

$$A^0 = \bigcap_{k=M-\lceil\sqrt{M}\rceil}^{M-1} \bigcap_{n=0}^{\min(\lceil\sqrt{M}\rceil,k)-1} A_{n,k}$$

W przedziale tym na pewno zawarty jest przedział A.

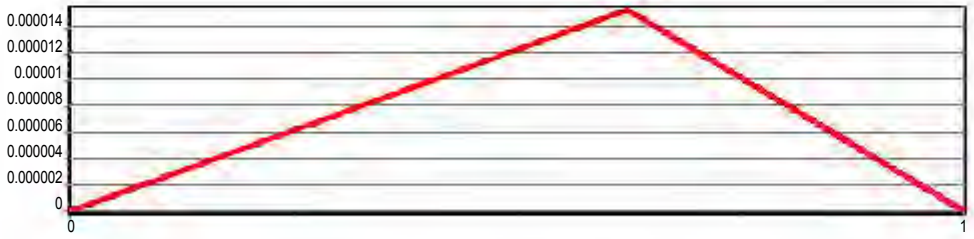
Następnie zdefiniujmy funkcję $B(a) = b_{max}(a) - b_{\min}(a)$.

Dla przykładowych danych wykres funkcji $B(a)$, $a \in A^0$, wygląda jak na Rys. 2.



Rys. 2. Wykres funkcji B(a) dla A°

Natomiast wykres funkcji $B(a)$, $a \in A$, dla tych samych danych ma postać jak na Rys. 3.



Rys. 3. Wykres funkcji B(a) dla A

W obydwu przypadkach oś X przedstawia badany przedział A^0 i A odpowiednio, ale dla celów prezentacyjnych jest on przedstawiony jako odcinek [0;1].

Zauważmy, że

Twierdzenie 3

$B(a) > 0$ wtedy i tylko wtedy, gdy $a \in A$.

Naszym celem jest znalezienie jakiegokolwiek a, dla którego $B(a) > 0$. Skorzystamy z faktu, że funkcja $B(a)$ ma maksimum dla pewnego punktu $a_{max} \in A$ oraz jest rosnąca dla $a < a_{max}$, a malejąca dla $a > a_{max}$.

Parametr a_{max}, dla którego funkcja $B(a)$ osiąga maksimum jest punktem, w którym parametr b osiąga największy zakres poprawnych wartości – mamy największą swobodę w jego wyborze. Teraz wykorzystując zmodyfikowany algorytm bisekcji poszukujący maksimum, w czasie logarytmicznym docieramy do pierwszego a, dla którego $B(a) > 0$, czyli do pierwszego $a \in A$. Następnie biorąc dowolne b z przedziału $(b_{min}(a); b_{max}(a))$ otrzymujemy szukaną parę współczynników a i b.

Złożoność obliczeniowa przedstawionego powyżej algorytmu dla przeciętnego przypadku jest rzędu $O(n \log_2 n)$. Dokładniej: wyznaczenie x_i – przeciętnie $O(n \log_2 n)$, pesymistycznie $O(n^2)$, wyznaczenie przedziału A^0 – $O(n)$, bisekcja używająca funkcji $B(a)$ – $O(n \log_2 n)$. Natomiast złożoność pamięciowa jest dla wszystkich części algorytmu rzędu $O(n)$.

3. Dowody twierdzeń

3.1. Dowód twierdzenia 1

Niech $a \in \mathbb{R}_+, b \in \mathbb{R}, x'_i \in \mathbb{N}, i = 0, \ldots, M-1$ będą znalezionymi przez algorytm rozwiązaniami takimi, że $y'_i = ax'_i + b, i = 0, \ldots, M-1$.

Załóżmy teraz, że istnieje inne rozwiązanie, czyli

$$a \in \mathbb{R}_+, b \in \mathbb{R}, x''_i \in \mathbb{N}, i = 0, \ldots, M-1,$$

spełniające równania

$$y'_i = a''x''_i + b'', i = 0, \ldots, M-1.$$

oraz $x''_0 = 0$, $b'' = y'_0$ i jednocześnie takie, że

$$x''_i < x'_i, i = 1, \ldots, M-1. \tag{11}$$

Stąd widać, że $b'' = y'_0 = b$ oraz

$$y'_i = ax'_i + b = a''x''_i + b, i = 0, \ldots, M-1.$$

czyli

$$ax'_i = a''x''_i.$$

Dalej korzystając z nierówności (11) otrzymujemy

$$\frac{a}{a''} = \frac{x''}{x'_i} < 1,$$

a stąd, wiedząc, że $a, a'' > 0$, mamy

$$a'' > a.$$

Teraz pamiętając, że zarówno a, jak i a'' zostały wyliczone ze wzoru (1) dla pewnych p oraz p' odpowiednio możemy napisać

$$\frac{dy'}{p''} = a'' > a = \frac{dy'}{p}.$$

Stąd już łatwo wynika, że

$$p > p''.$$

Skoro jednak p poszukiwane było poprzez podstawianie wszystkich liczb naturalnych od 1 aż do p, to gdyby istniało $p'' \in \mathbb{N}$ takie, że $p'' < p$, to wówczas byłoby ono znalezione przed p jako rozwiązanie problemu. Stąd nie może istnieć takie p'', czyli również nie istnieje a'', a w konsekwencji nie istnieją $x''_i \in \mathbb{N}, i = 0, \ldots, M-1$.

□

3.2. Dowód twierdzenia 2

Ustalmy pewne $a \in A$. Stąd wiemy, że dla tego a spełnione są nierówności (9) dla dowolnych $n, k = 0, \ldots, M-1, n < k$. Mamy udowodnić, że przy tych założeniach istnieje $b \in \mathbb{R}$ takie, że zachodzi (4). Wiemy, że dla ustalonego a nierówności (4) są równoważne z (10). Wykażemy więc, że istnieje b takie, że

$$b_{\min}(a) \leq b < b_{max}(a),$$

czyli, że

$$b_{\min}(a) < b_{max}(a),$$

i rozpisując

$$\max_{i=0,\ldots,M-1} y_i - ax_i < \min_{i=0,\ldots,M-1} y_i - ax_i + 1.$$

To samo można zapisać w inny sposób, tzn. że prawdziwe jest

$$\bigwedge_{n,k=0,\ldots,M-1} y_n - ax_n < y_k - ax_k + 1.$$

Załóżmy najpierw, że $n < k$, wówczas mamy

$$y_n - ax_n < y_k - ax_k + 1,$$
$$ax_k - ax_n < y_k - y_n + 1,$$

a stąd

$$a < \frac{y_k - y_n + 1}{x_k - x_n},$$

co z (9) wiemy, że jest prawdą.

Załóżmy teraz, że $n > k$. Analogicznie

$$y_n - ax_n < y_k - ax_k + 1,$$
$$y_n - y_k - 1 < ax_n - ax_k,$$

a stąd

$$a > \frac{y_n - y_k - 1}{x_n - x_k},$$

co również wynika z (9).

Jeśli natomiast $n = k$, to dostajemy wyrażenie zawsze prawdziwe $0 < 1$.

Tak więc wykazaliśmy, że $b_{\min}(a) < b_{max}(a)$. Czyli można wybrać np. $b = \frac{b_{\min}(a)+b_{max}(a)}{2}$. Takie b będzie pomiędzy $b_{\min}(a)$ i $b_{max}(a)$, a więc prawdziwa będzie nierówność

$$b_{\min}(a) \leq b < b_{max}(a).$$

Czyli dla dowolnego $a \in A$ istnieje $b \in \mathbb{R}$ takie, że zachodzi (4).

□

3.3. Dowód twierdzenia 3

Jeśli $a \in A$, to wiemy z dowodu twierdzenia 2, że $b_{\min}(a) < b_{max}(a)$ czyli, że

$$B(a) = b_{max}(a) - b_{\min}(a) > 0.$$

W drugą stronę. W jakimś zakresie powtórzymy rozumowanie z dowodu twierdzenia 2. Jeśli $B(a) > 0$, to

$$b_{\min}(a) < b_{max}(a),$$

czyli

$$\max_{i=0,\dots,M-1} y_i - ax_i < \min_{i=0,\dots,M-1} y_i - ax_i + 1.$$

To samo inaczej

$$\bigwedge_{n,k=0,\dots,M-1} y_n - ax_n < y_k - ax_k + 1,$$

więc

$$a < \frac{y_k - y_n + 1}{x_k - x_n}, n, k = 0, \dots, M-1, n < k.$$

oraz

$$a > \frac{y_k - y_n - 1}{x_k - x_n}, n, k = 0, \dots, M-1, n < k.$$

Stąd oraz z definicji zbiorów $A_{n,k}$ i A widzimy, że $a \in A$.

□

4. Wnioski

Podsumowując warto jeszcze raz podkreślić, że oba zaprezentowane algorytmy dają możliwość znacznej redukcji zajmowanej przez dane przemysłowo-diagnostyczne pamięci (bez utraty informacji) już na etapie wstępnego przygotowania do właściwej bezstratnej kompresji.

Stopień tej redukcji zależy od sposobu zapisu danych źródłowych (float, double, itp.) oraz od ilości bitów przetwornika AC. Np. dla liczb typu double i przetwornika 16-bitowego będzie to czterokrotne zmniejszenie objętości danych.

Zaznaczmy ponadto, że dla przykładowych danych z 200000 liczb typu double czas wykonania obu algorytmów na CPU 2.1GHz (z wykorzystaniem jednego rdzenia) to około 0,04s.

Literatura

[1] F. S. Hillier, G. J. Lieberman (1986), *Introduction to Operations Research*, *4th Ed.*, Holden-Day Inc., Oakland, California.

[2] D. E. Knuth, R. L. Graham, O. Patashnik (2002), *Matematyka konkretna*, PWN, Warszawa.

[3] W. Krysicki, J. Bartos, W. Dyczka, K. Królikowska, M. Wasilewski (1999), *Rachunek prawdopodobieństwa i statystyka matematyczna w zadaniach*, PWN, Warszawa.

ISBN 9788389475336

J. Hołubiec, red. – TECHNIKI INFORMACYJNE-TEORIA I ZASTOSOWANIA, tom 1 (13)