Liniowe dopasowanie dwóch sekwencji

Olga RUSINEK
Karol SIWEK
Mateusz SUCHOCKI

Zastosowanie i znaczenie bioinformatyczne

Podobieństwo porównywanych sekwencji może świadczyć o:

- podobnej funkcji sekwencji
- podobnej strukturze białek
- wspólnej historii ewolucyjnej sekwencji

Dopasowanie dwóch sekwencji

- Globalne dopasowanie dwóch sekwencji otrzymujemy przez wstawienie pustych miejsc(spacji, tyld) zarówno w środek jak i na końcu sekwencji S1 i S2, następnie umieszczenie tych sekwencji jedna nad drugą w ten sposób, że każdy znak (litera lub spacja) w jednej sekwencji posiada odpowiedni znak lub spację w przeciwległej sekwencji.

  - A L A M A K O T _ K A
  - M _ A K A R O N I K _

Definicja formalna:

- Rozważamy słowa nad alfabetem Σ. Dodajemy nowy symbol spacji Σ' = Σ ∪ {_}. Dla słów $u, w \in \Sigma^*$ ich **liniowym dopasowaniem** nazywamy parę słów $u^*, w^* \in (\Sigma')^*$ spełniających warunki:
- Usunięcie spacji z u* i w* daje w wyniku odpowiednio u i w,
- |u*| = |w*|
- $\forall_{i \le |u^*|}\ u^*[i] \neq '\_' \vee u^*[i] \neq '\_'$

Przykład: u=ALAMAKOTKA, w=MAKARONIK

```
u* = A L A M A K O T _ K A
w* = M _ A K A R O N I K _
```

Odległość edycyjna „ważona”

- **Odległością edycyjną** *(edit distance)* między dwoma sekwencjami jest minimalna liczba operacji edycji (działań prostych) – wstawiania, usuwania i zamiany symboli, konieczna do transformacji pierwszej sekwencji z drugą. Symbole pasujące nie są liczone.

- **Odległość edycyjna ważona** – operacje: wstawiania, usuwania i zamiany mogą mieć różne wagi.

Odległość edycyjna- przykład

- u = ALAMAKOTKA, w= MAKARONIK
- R – zamiana *(replacement)*
- I – wstawienie *(insertion)*
- D – usunięcie *(deletion)*
- M – zgodność *(matching)*

```
R D M R M R M R I M D
A L A M A K O T _ K A
M _ A K A R O N I K _
```

Odległość edycyjna- przykład

- Napis `RDMRMRMRIMD` jest **transkryptem edycji**, czyli opisem transformacji jednej sekwencji w drugą.

- Przyjmując dla wszystkich operacji edycji równe wagi(1), transformacja napisu *u* w napis *w* ma koszt `7` :
  - 1 wstawienie,
  - 2 usunięcia,
  - 4 zamiany.

Ważona odległość edycyjna

- Problem ważonej operacyjnie odległości edycyjnej polega na znalezieniu transkryptu edycji o najmniejszej całkowitej wadze operacji, przy ustalonych wagach na poszczególne działania proste.

- Przykład:
  - Transkrypt: `RDMRMRMRIMD`
  - Wagi:
    - `D=1, I=1, R=2, M=0`
  - Koszt: `2*D + 1*I + 5*R + 4*M = 13`

Programowanie dynamiczne

- Programowanie dynamiczne jest techniką lub strategią projektowania algorytmów, stosowaną przeważnie do rozwiązywania zagadnień optymalizacyjnych. Jest alternatywą dla niektórych zagadnień rozwiązywanych za pomocą algorytmów zachłannych.
- 3 zasadnicze podproblemy programowania dynamicznego:
  - Równanie rekurencyjne i funkcja celu
  - Obliczenia tabelaryczne
  - Powrót (traceback)

Algorytm wielomianowy

- Ponieważ problem znalezienia optymalnego dopasowania (minimalnej odległości edycyjnej) metodą zachłanną wymagałoby dużej liczby rekurencyjnych zagnieżdżeń, lepszym rozwiązaniem będzie zastosowanie algorytmu opartego na programowaniu dynamicznym. Zapisywanie wartości kolejnych podproblemów w tablicy pozwala znacznie oszczędzić czas.

Algorytm wielomianowy cd.

- **`D(i,j)`** – odległość edycyjna między $S_1[1..i]$ i $S_2[1..j]$ dla dwóch sekwencji S1 i S2, wyznacza najmniejszą liczbę operacji edycji potrzebnych do transformacji pierwszych *i* symboli napisu S1 do pierwszych *j* symboli napisu S2.
- Zakładając $|S_1|=n$ oraz $|S_2|=m$, odległość edycyjna między S1 a S2 wynosi dokładnie `D(n,m)`.

Algorytm wielomianowy –równanie rekurencyjne

- Warunki początkowe:
  - `D(i,0)=i,`
  - `D(0,j)=j`
    - Utworzenie na pustym napisie napisu o długości *j* wymaga *j* operacji wstawiania, zaś utworzenie z napisu długości *i* napisu pustego wymaga *i* operacji usunięcia.
- Równanie rekurencyjne :
  - `D(i,j)=min[D(i-1,j)+1, D(i,j-1)+1, D(i-1, j-1)+t(i,j)]`
- Funkcja celu -> minimalizacja kosztu

Algorytm wielomianowy – poprawność równania rekurencyjnego

- Dowód poprawności równania rekurencyjnego:
  - `D(i,j)=min[D(i-1,j)+1, D(i,j-1)+1, D(i-1, j-1)+t(i,j)]`
- Weźmy pod uwagę ostatni symbol transkryptu:
  - I – wstawienie symbolu S2(j) na koniec transformowanego S1
    - `D(i,j)=D(i,j-1)+1`
  - D – usunięcie S1(i) z S1
    - `D(i,j)=D(i-1,j)+1`
  - R – zamiana S1(i) na S2(j)
    - `D(i,j)=(i-1, j-1)+1`
  - M – zgodność: S1(i)=S2(j)
    - `D(i,j)=(i-1,j-1)`
- t(i,j):
  - `0` dla M
  - `1` dla R

Algorytm wielomianowy – poprawność równania rekurencyjnego – cd.

- I – wstawienie symbolu S2(j) na koniec transformowanego S1
- Poprzedni krok musi więc zawierać minimalną liczbę operacji potrzebną do transformacji S1[1..i] do S2[1..j-1] (jeśli się tak nie dzieje, oznacza to, że przedstawiona transformacja z S1[1..i] do S2[1..j] nie jest optymalna).
- Wstawienie symbolu S2(j) jest kolejną operacją edycji, stąd wzór:
  - `D(i,j)=D(i,j-1)+1`

Algorytm wielomianowy – poprawność równania rekurencyjnego – cd.

- D – usunięcie S1(i) z końca transformowanego napisu S1
- Poprzedni krok musi więc zawierać minimalną liczbę operacji potrzebną do transformacji S1[1..i-1] do S2[1..j] (jeśli się tak nie dzieje, oznacza to, że przedstawiona transformacja z S1[1..i] do S2[1..j] nie jest optymalna).
- Usunięcie symbolu S1(i) jest kolejną operacją edycji, stąd wzór:
  - `D(i,j)=D(i-1,j)+1`

Algorytm wielomianowy – poprawność równania rekurencyjnego – cd.

- R – Zamiana znaków S1(i) z S2(j)
- Poprzedni krok musi więc zawierać minimalną liczbę operacji potrzebną do transformacji S1[1..i-1] do S2[1..j-1] (jeśli się tak nie dzieje, oznacza to, że przedstawiona transformacja z S1[1..i] do S2[1..j] nie jest optymalna).
- Zamiana jest kolejną operacją edycji, stąd wzór
  - `D(i,j)=D(i-1,j)+1`
- M – Zgodność S(i)=S2(j)
- Przepisujemy symbole bez operacji edycji, stąd wzór:
  - `D(i,j)=D(i-1,j)`

Algorytm wielomianowy –obliczenia tabelaryczne

- W programowaniu dynamicznym używa się tablic, aby zapisać wyniki poszczególnych podproblemów, unikając w ten sposób namnażających się wywołań rekurencyjnych, które czyniłyby algorytm nieefektywnym – liczba wywołań rośnie wykładniczo od *m* i *n*.
- Ponieważ mamy tylko *(n+1)*(m+1)* instancji *i* i *j*, tabela o rozmiarach *(n+1)*(m+1)* zminimalizuje nam liczbę obliczeń.

Algorytm wielomianowy –obliczenia tabelaryczne

- Tworzymy tabelę dla S1=vintner oraz S2=writers
- Korzystamy z wartości początkowych:
  - `D(i,0)=i`
  - `D(0,j)=j`

	i		w	r	i	t	e	r	s
j		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1							
i	2	2							
n	3	3							
t	4	4							
n	5	5							
e	6	6							
r	7	7							

Algorytm wielomianowy –obliczenia tabelaryczne

- Obliczamy `D(1,1)`
  - `D(i-1,j)  +1= 2`
  - `D(i,j-1)  +1= 2`
  - `D(i-1,j-1)+1=1`

	i		w	r	i	t	e	r	s
j		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1							
i	2	2							
n	3	3							
t	4	4							
n	5	5							
e	6	6							
r	7	7							

  - Użytą operacją jest operacja zamiany, więc wartość komórki obliczamy ze wzoru :
- `D(i,j)=D(i-1,j-1)+1=1`

	i		w	r	i	t	e	r	s
j		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1	1						
i	2	2							
n	3	3							
t	4	4							
n	5	5							
e	6	6							
r	7	7							

Algorytm wielomianowy –obliczenia tabelaryczne

- Obliczamy `D(1,2)`
  - `D(0,2)+1= 2+1 = 3`
  - `D(1,1)+1= 1+1 = 2`
  - `D(0,1)+1= 1+1 = 2`

	i		w	r	i	t	e	r	s
j		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1	1						
i	2	2							
n	3	3							
t	4	4							
n	5	5							
e	6	6							
r	7	7							

- W tym wypadku możemy skorzystać z operacji zamiany lub usunięcia symbolu, w wyniku czego otrzymujemy liczbę 2.

	i		w	r	i	t	e	r	s
j		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1	1						
i	2	2	2						
n	3	3							
t	4	4							
n	5	5							
e	6	6							
r	7	7							

Algorytm wielomianowy –obliczenia tabelaryczne

- Obliczamy `D(2,1)`
  - `D(1,1)+1= 1+1 = 2`
  - `D(2,0)+1= 2+1 = 3`
  - `D(1,0)+1= 1+1 = 2`

	i		w	r	i	t	e	r	s
j		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1	1						
i	2	2	2						
n	3	3							
t	4	4							
n	5	5							
e	6	6							
r	7	7							

- W tym wypadku możemy skorzystać z operacji zamiany lub wstawienia symbolu, w wyniku czego otrzymujemy liczbę 2.

	i		w	r	i	t	e	r	s
j		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1	1	2					
i	2	2	2						
n	3	3							
t	4	4							
n	5	5							
e	6	6							
r	7	7							

Algorytm wielomianowy –obliczenia tabelaryczne

- Obliczamy `D(3,2)`
  - `D(2,2)+1= 2+1 = 3`
  - `D(3,1)+1= 3+1 = 3`
  - `D(2,1)+0= 2+0 = 2`

	i		w	r	i	t	e	r	s
j		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1	1	2	3				
i	2	2	2	2					
n	3	3							
t	4	4							
n	5	5							
e	6	6							
r	7	7							

  - Ponieważ symbole S1(3) oraz S2(2) są równe, nie wykonujemy żadnych operacji edycyjnych – korzystamy ze wzoru:
- `D(i-1,j-1)+0= 2`

	i		w	r	i	t	e	r	s
j		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1	1	2	3				
i	2	2	2	2	2				
n	3	3							
t	4	4							
n	5	5							
e	6	6							
r	7	7							

Algorytm wielomianowy –obliczenia tabelaryczne

- Obliczamy `D(3,7)`
  - `D(2,7)+1= 6+1 = 7`
  - `D(3,6)+1= 6+1 = 7`
  - `D(2,6)+1= 6+1 = 7`

	i		w	r	i	t	e	r	s
j		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1	1	2	3				
i	2	2	2	2	2				
n	3	3	3	3	3				
t	4	4	4	4	4				
n	5	5	5	5	5				
e	6	6	6	6	6				
r	7	7	7	6					

- W tym wypadku możemy skorzystać z operacji zamiany, wstawienia lub usunięcia symboli, w wyniku czego otrzymujemy liczbę 7.

	i		w	r	i	t	e	r	s
j		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1	1	2	3				
i	2	2	2	2	2				
n	3	3	3	3	3				
t	4	4	4	4	4				
n	5	5	5	5	5				
e	6	6	6	6	6				
r	7	7	7	6	7				

Algorytm wielomianowy –obliczenia tabelaryczne

- Obliczamy `D(7,7)`
  - `D(6,7)+1= 4+1 = 5`
  - `D(7,6)+1= 6+1 = 7`
  - `D(6,6)+1= 5+1 = 6`

	i		w	r	i	t	e	r	s
j		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1	1	2	3	4	5	6	7
i	2	2	2	2	2	3	4	5	6
n	3	3	3	3	3	3	4	5	6
t	4	4	4	4	4	3	4	5	6
n	5	5	5	5	5	4	4	5	6
e	6	6	6	6	6	5	4	5	6
r	7	7	7	6	7	6	5	4	

- W tym wypadku korzystamy z operacji wstawienia symbolu, ze wzoru:
  - `D(i-1,j)+1= 5`

	i		w	r	i	t	e	r	s
j		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1	1	2	3	4	5	6	7
i	2	2	2	2	2	3	4	5	6
n	3	3	3	3	3	3	4	5	6
t	4	4	4	4	4	3	4	5	6
n	5	5	5	5	5	4	4	5	6
e	6	6	6	6	6	5	4	5	6
r	7	7	7	6	7	6	5	4	5

Algorytm wielomianowy –traceback

- Z powstałej tabeli wynika, że odległość edycyjna pomiędzy *vintner* a *writers* wynosi D(7,7)=5.

	i		w	r	i	t	e	r	s
j		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1	1	2	3	4	5	6	7
i	2	2	2	2	2	3	4	5	6
n	3	3	3	3	3	3	4	5	6
t	4	4	4	4	4	3	4	5	6
n	5	5	5	5	5	4	4	5	6
e	6	6	6	6	6	5	4	5	6
r	7	7	7	6	7	6	5	4	5

- Jeżeli odpowiednio zmodyfikujemy tabelę, będziemy mogli odtworzyć wszystkie optymalne transkrypty edycyjne. W tym celu należy dodać do odpowiednich komórek wskaźniki na ich poprzedników.

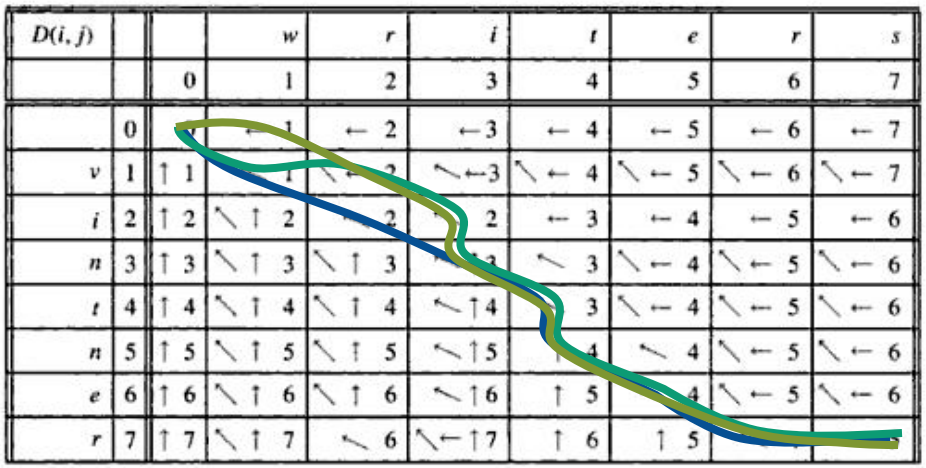
D(i, j)			*w*	*r*	*i*	*t*	*e*	*r*	*s*
		0	1	2	3	4	5	6	7
	0	0	← 1	← 2	←3	← 4	← 5	← 6	← 7
v	1	↑ 1	↖ 1	↖← 2	↖←3	↖← 4	↖← 5	↖← 6	↖← 7
i	2	↑ 2	↖ ↑ 2	↖ 2	↖ 2	← 3	← 4	← 5	← 6
n	3	↑ 3	↖ ↑ 3	↖ ↑ 3	↖↑3	↖ 3	↖← 4	↖← 5	↖← 6
t	4	↑ 4	↖ ↑ 4	↖ ↑ 4	↖↑4	↖ 3	↖← 4	↖← 5	↖← 6
n	5	↑ 5	↖ ↑ 5	↖ ↑ 5	↖↑5	↑ 4	↖ 4	↖← 5	↖← 6
e	6	↑ 6	↖ ↑ 6	↖ ↑ 6	↖↑6	↑ 5	↖ 4	↖← 5	↖← 6
r	7	↑ 7	↖ ↑ 7	↖ 6	↖←↑7	↑ 6	↑ 5	↖ 4	← 5

Algorytm wielomianowy –traceback

$D(i,j)$			w	r	i	t	e	r	s
		0	1	2	3	4	5	6	7
	0	0	← 1	← 2	← 3	← 4	← 5	← 6	← 7
v	1	↑ 1	↖ 1	↖ ← 2	↖ ← 3	↖ ← 4	↖ ← 5	↖ ← 6	↖ ← 7
i	2	↑ 2	↖ ↑ 2	↖ 2	↖ 2	← 3	← 4	← 5	← 6
n	3	↑ 3	↖ ↑ 3	↖ ↑ 3	↖ 3	↖ 3	↖ ← 4	↖ ← 5	↖ ← 6
t	4	↑ 4	↖ ↑ 4	↖ ↑ 4	↖ ↑ 4	↖ 3	↖ ← 4	↖ ← 5	↖ ← 6
n	5	↑ 5	↖ ↑ 5	↖ ↑ 5	↖ ↑ 5	↑ 4	↖ 4	↖ ← 5	↖ ← 6
e	6	↑ 6	↖ ↑ 6	↖ ↑ 6	↖ ↑ 6	↑ 5	4	↖ ← 5	↖ ← 6
r	7	↑ 7	↖ ↑ 7	↖ 6	↖ ← ↑ 7	↑ 6	↑ 5	[illegible]	[illegible]

Algorytm wielomianowy –traceback

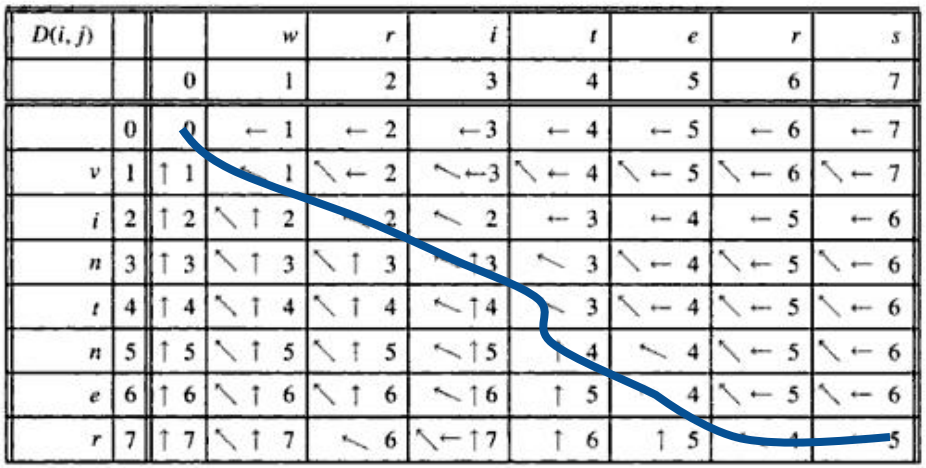
- Idąc za strzałkami, znajdujemy 3 różne trasy z punktu `(7,7)` do `(0,0)`.
- Strzałki oznaczają:
  - ← I – wstawienie
  - ↑ D – usunięcie
  - ↖ R,M – zamiana (jeżeli symbole są różne) lub zgodność



D(i, j)			w	r	i	t	e	r	s
		0	1	2	3	4	5	6	7
	0		← 1	← 2	←3	← 4	← 5	← 6	← 7
v	1	↑ 1			↖←3	↖ ← 4	↖ ← 5	↖ ← 6	↖ ← 7
i	2	↑ 2	↖ ↑ 2		2	← 3	← 4	← 5	← 6
n	3	↑ 3	↖ ↑ 3	↖ ↑ 3		↖ 3	↖ ← 4	↖ ← 5	↖ ← 6
t	4	↑ 4	↖ ↑ 4	↖ ↑ 4	↖ ↑4	3	↖ ← 4	↖ ← 5	↖ ← 6
n	5	↑ 5	↖ ↑ 5	↖ ↑ 5	↖ ↑5		↖ 4	↖ ← 5	↖ ← 6
e	6	↑ 6	↖ ↑ 6	↖ ↑ 6	↖ ↑6	↑ 5		↖ ← 5	↖ ← 6
r	7	↑ 7	↖ ↑ 7	↖ 6	↖← ↑7	↑ 6	↑ 5		

Algorytm wielomianowy –traceback

- Przykładowe odtworzenie transkryptu:
- ↖ ↖ ↖ ↖ ↑ ↖ ↖ ←
- R R R M D M M I
- Stosując podany transkrypt otrzymujemy:
  - RRRMDMMI
  - vintner_
  - writ_ers

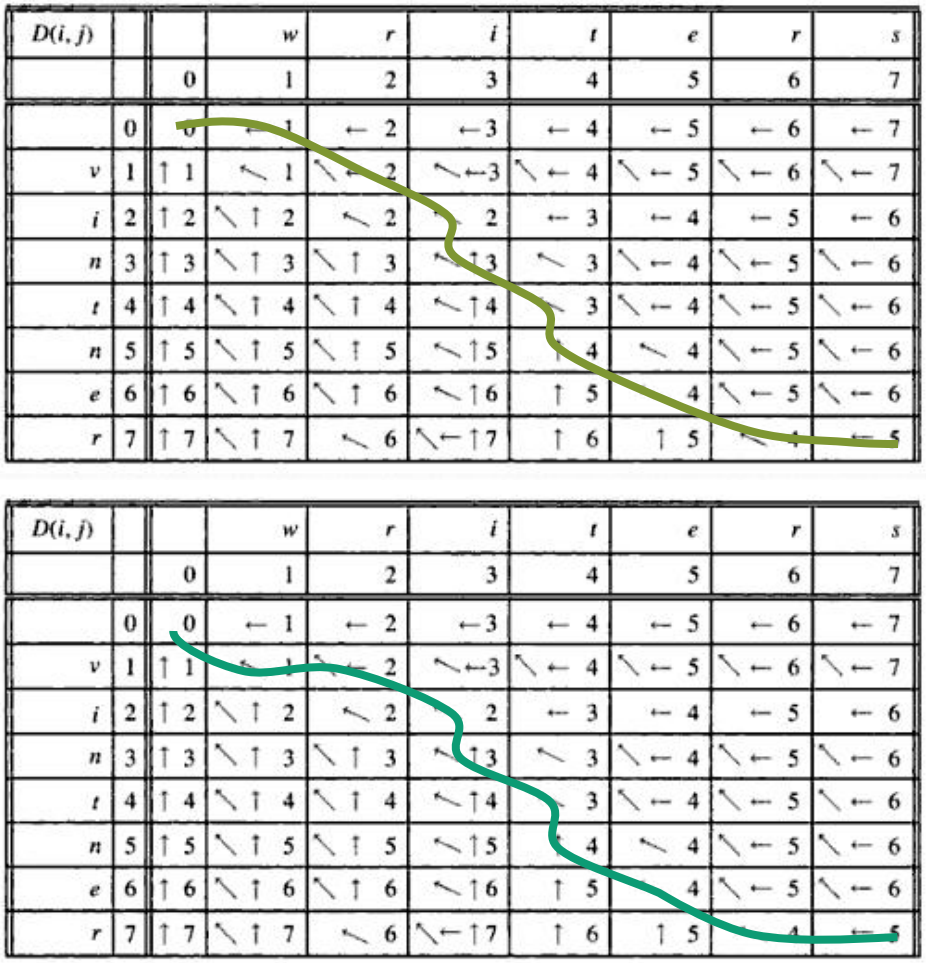


Kolejne symbole wskazują:

- R – zamiana v na w
- R – zamiana i na r
- R – zamiana n na i
- M – i=i, brak edycji
- D – usunięcie n
- M – e=e, brak edycji
- M – r=r, brak edycji
- I – wstawienie s

Algorytm wielomianowy –traceback

- Kolejne 2 transkrypty:
- IRMDMDMMI
  - `IRMDMDMMI`
  - `_vintner_`
  - `wri_t_ers`
- RIMDMDMMI
  - `RIMDMDMMI`
  - `v_intner_`
  - `wri_t_ers`



Algorytm wielomianowy –podsumowanie

- Każda ścieżka opisuje jednoznacznie jeden transkrypt, każdy transkrypt można jednoznacznie opisać za pomocą tylko jednej ścieżki.
- Algorytm znajduje wszystkie optymalne dopasowania, w tym wypadku były to:
  - `vintner_` `_vintner_` `v_intner_`
  - `writ_ers` `wri_t_ers` `wri_t_ers`
  - Złożoność algorytmu opartego na programowaniu dynamicznym ze wskaźnikami:
    - Czasowa: ***O(n+m)***
    - Pamięciowa: ***O(nm)***

Interpretacja grafowa algorytmu

- Tabela jest prostą metodą dopasowania dwóch sekwencji przy założeniu, że każda edycja ma tą samą wagę. Zakładając różne wagi poszczególnych operacji, sytuacja staje się bardziej skomplikowana.
- Użyteczną metodą reprezentacji rozwiązań programowania dynamicznego jest graf edycji.

Interpretacja grafowa algorytmu

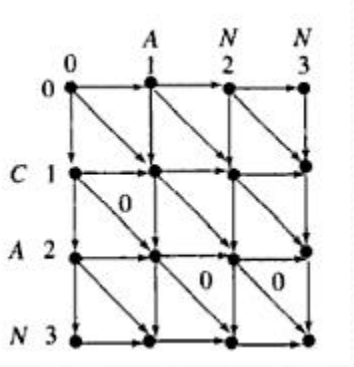
- Dla dwóch sekwencji S1 i S2 o długościach odpowiednio n i m, **ważony graf edycji** *(weighted edit graph)* posiada `(n+1)*(m+1)` węzłów, poetykietowanych jako `(i,j)` `(0≤i≤n, 0≤j≤m)`.
- Wagi poszczególnych krawędzi zależą od konkretnego problemu (najczęściej – od wag konkretnych operacji edycji).
- Znalezienie optymalnego dopasowania polega na znalezieniu najkrótszej ścieżki w grafie.
- ***Transkrypt edycji dla S1 i S2 ma minimalną liczbę operacji edycji wtedy i tylko wtedy, gdy pokrywa najkrótszą ścieżkę z `(0,0)` do `(n,m)` w grafie edycji.***

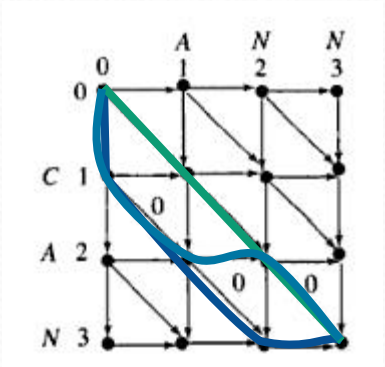
Interpretacja grafowa algorytmu

- Pary `(0,0) .. (n,m)` tworzą wierzchołki digrafu.
- Krawędzie postaci
  - (i,j) -> (i-1,j-1), waga d(u[i],w[j])
  - (i,j) -> (i,j-1) waga d('-', w[j])
  - (i,j) -> (i-1,j) waga d(u[i], '-')
- Dopasowanie: dowolna ścieżka z (n,m) do (0,0).
- Najtańsze dopasowanie: najkrótsza taka ścieżka

Interpretacja grafowa algorytmu- przykład

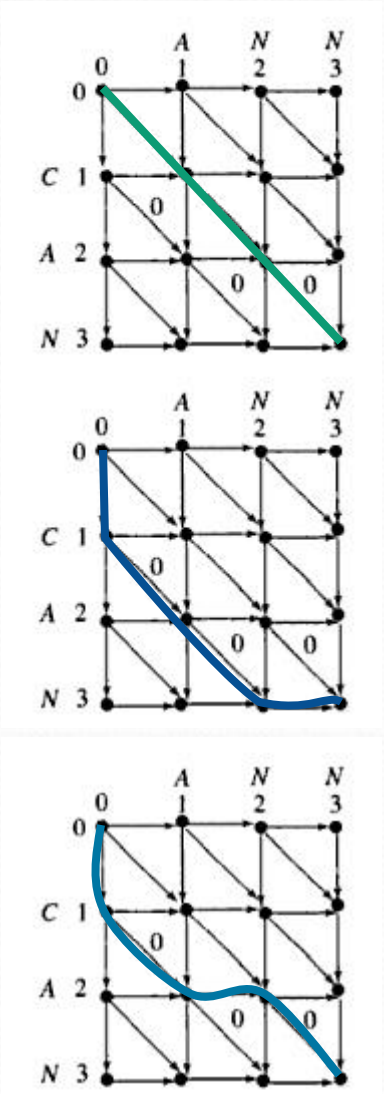
- Porównamy dwa napisy:
  - S1 = CAN
  - S2= ANN
  - Wszystkie krawędzie oprócz oznaczonych `0` (matching) mają wagę `1`.
  - Odnajdujemy najkrótsze ścieżki – wszystkie mają długość 2.
  - Tworzymy transkrypt:
  - → I –wstawienie
  - ↓ D –usunięcie
  - ↘ R,M – zamiana lub pasujące





Interpretacja grafowa algorytmu- przykład

- ↘ ↘ ↘ - RRM
- **RRM**
- **CAN**
- **ANN**
- ↓ ↘ ↘ → - DMMI
- **DMMI**
- **CAN_**
- **_ANN**
- ↓ ↘ → ↘ - DMIM
- **DMIM**
- **CA_N**
- **_ANN**



Ważona odległość edycyjna

- **Operacyjnie *(operation-weight)***
  - Operacje mają różne wagi, przykładowo:
    - `d=1` - I lub D (wstawienie lub usunięcie)
    - `r=2` - R (zamiana)
    - `e=0` - m (brak edycji)
  - Zmodyfikowane warunki początkowe:
    - `D(i,0)=i*d`
    - `D(0,j)=j*d`

- **Alfabetowo *(alphabet-weight)***
  - Niektóre zamiany mogą mieć większy koszt niż inne, np.: przy replikacji DNA zamiana A(adenina) na T(tymina) ma większy koszt niż zamiana A na G(guanina)
  - Operacje wstawiania i usuwania również mogą mieć różne koszta, w zależności od tego, jaki znak jest wstawiany lub usuwany.
  - Przy porównywaniu protein, pojecie „odległość edycyjna” prawie zawsze oznacza odległość ważoną alfabetowo. Alfabetem jest zbiór aminokwasów ({A,T,C,G} dla DNA lub {A,U,C,G} dla RNA).

Funkcja podobienstwa liter i podobienstwo dopasowań

Czy każda zamiana liter w dwóch ciągach jest równie prawdopodobna?

- Def 1

Niech S1 S2 – słowa nad alfabetem ∑', ∑ niech będzie ∑' wzbogaconym o spację {'_'} a x,y będą znakami ze słów S1 S2,

Funkcję: s(x,y) nazywamy funkcje podobieństwa liter.

- Def2

Niech S1' oraz S2' będą słowami S1 oraz S2 po dodaniu spacji, a l ich długością. Podobieństwem dopasowań S1 i S2 nazywamy liczbę:

$$\sum_{i=1}^{l} s(S'_1(i), S'_2(i))$$

Podobieństwo sekwencji

- Odległość edycyjna jest jedną z metod określenia pokrewieństwa między sekwencjami.
- Drugą, częściej używaną metodą, jest określenie **podobieństwa sekwencji**, zamiast ich odległości.
- To podejście jest częściej stosowane w aplikacjach biologicznych, gdyż jest wygodniejsze i bardziej intuicyjne niż odległość edycyjna.

Podobieństwo sekwencji

- Niech S1 i S2 będą słowami nad alfabetem Σ, zaś Σ'=Σ∪{_}. Wtedy dla każdych dwóch symboli *x*, *y* w Σ', *s(x,y)* oznacza wartość (*score*-wynik) otrzymaną przez dopasowanie symbolu *x* do symbolu *y*.

- Dla danego dopasowania *A* dla *S1* i *S2*, niech *S1'* i *S2'* będą sekwencjami po dodaniu odpowiednich spacji i niech *l* oznacza długość (równą) tych sekwencji w *A*. Wartość dopasowania *A* jest zdefiniowana jako:
  - $\sum_{i=1}^{l} s(S_1'(i), S_2'(i))$

Podobieństwo sekwencji-przykład

- Niech Σ = {a,b,c,d}. Wtedy Σ' = {a,b,c,d,_}.
- Macierz przedstawia koszty dopasowania dla poszczególnych par znaków.
- Rozważmy sekwencje:
  - S1 = cacdbd
  - S2= cabbdb

  oraz jedno z ich dopasowań:
  - `c a c _ d b d`
  - `c a b b d b _`
- Koszt tego dopasowania liczymy jako:
  - `0 + 1 – 2 + 0 + 3 + 3 – 1 =4`

s	a	b	c	d	-
a	1	-1	-2	0	-1
b		3	-2	-1	0
c			0	-4	-2
d				3	-1
-					0

Podobieństwo sekwencji-przykład

- W problemie podobieństwa sekwencji, macierz kosztów jest najczęściej konstruowana w ten sposób, że:
  - $s(x,y) \geq 0$ dla $x=y$, $x,y \in \Sigma'$
  - $s(x,y) < 0$ dla $x \neq y$, $x,y \in \Sigma'$
  - Na podstawie tego schematu, szukamy dopasowania o jak największej wartości.
  - Schemat ten uwydatnia zgodności i podobieństwa między znakami, wprowadzając jednocześnie kary za niedopasowania i wstawione spacje.
  - Do porównywania sekwencji DNA wyznaczone są specjalne macierze, uwzględniające również częstotliwości obserwowanych podstawień aminokwasów (prawdopodobieństwa mutacji).

s	a	b	c	d	-
a	1	-1	-2	0	-1
b		3	-2	-1	0
c			0	-4	-2
d				3	-1
-					0

Podobieństwo sekwencji

- Mając daną macierz kosztów nad alfabetem Σ' , **<u>podobieństwo dwóch sekwencji</u>** *S1* i *S2* jest zdefiniowane jako taka wartość dopasowania *A* z *S1* do *S2*, że wartość całkowitego wyrównania jest największa. Jest to również nazwane **<u>wartością optymalnego zrównania</u>** z *S1* do *S2*.
- Podobieństwo jest ściśle związane z ważoną alfabetowo odległością edycyjną, a jego wartość jest zależna od użytej macierzy kosztów.

s	a	b	c	d	-
a	1	-1	-2	0	-1
b		3	-2	-1	0
c			0	-4	-2
d				3	-1
-					0

Algorytm na optymalne dopasowanie

- Niech *V(i,j)* będzie wartością optymalnego dopasowania prefiksów *S1[1..i]* oraz *S2[1..j]*.
- Niech '_' oznacza spację wstawioną do napisu.
- Wartości początkowe:
  - $V(0,j) = \sum_{(1 \le k \le j)} s(\_, S2(k))$
  - $V(i,0) = \sum_{(1 \le k \le i)} s(S1(k), \_)$
- Wzór ogólny:
  - $$V(i,j) = \max[\ V(i-1,j-1) + s(S_1(i), S_2(j)),\ V(i-1,j) + s(S_1(i), \_),\ V(i,j-1) + s(\_, S_2(j))\ ]$$

- Jeżeli S1 i S2 mają długość *n* i *m*, wartość ich optymalnego dopasowania wynosi *V(n,m)*.
- Ustawiając wskaźniki (podobnie jak w problemie odległości edycyjnej), koszt czasowy algorytmu wynosi **O(nm).**
- Optymalnym rozwiązaniem jest dowolna ścieżka z komórki (n,m) do (0,0), otrzymana na podstawie wskaźników.

Zależność pomiędzy odległością edycyjną a maksymalnym podobieństwem

- Tw. Watermana:
  - Niech:
    - $s(a,b)$ będzie funkcją podobieństwa,
    - $\hat{g}(k)$ funkcją kary za operacje INDEL,
    - $d(a,b)$ funkcja odległości,
    - $g(k)$ waga operacji INDEL.
  - Jeśli istnieje stała c, taka że:
    - $s(a,b)=c-d(a,b)$ oraz
    - $\hat{g}(k)=g(k)-kc/2$,

    to dopasowanie jest **maksymalnym podobieństwem**, wtedy i tylko wtedy gdy jest to **optymalna odległość edycyjna**.

Zależność pomiędzy odległością edycyjną a maksymalnym podobieństwem

Dowód:

$$n+m=2*|M|+\sum_k k\,\Delta_k$$

M-zbiór dopasowanych znaków, Δ_k -liczba operacji INDEL długości k

$$D(a,b)=min\{\sum_M d(a,b)+\sum_k g(k)\Delta_k\}$$

$$=min\{\sum_M c+\sum_k k\,\Delta_k\, c/2-\sum_M s(a,b)+\sum_k \hat{g}(k)\Delta_k\}=$$

$$=min\{c(n+m)/2-\sum_M s(a,b)+\sum_k \hat{g}(k)\Delta_k\}=$$

$$=c(n+m)/2-max\{\sum_M s(a,b)-\sum_k \hat{g}(k)\Delta_k\}=$$

$$=c(n+m)/2-S(a,b)$$

Zależność pomiędzy odległością edycyjną a maksymalnym podobieństwem

- Wniosek:

$$d(a,b)+s(a,b)=c(n+m)/2=const$$

- Interpretacja:
  - Im większa odległość $d(a,b)$, tym mniejsze podobieństwo $s(a,b)$.
  - Oznacza to, że minimalna odległość implikuje maksymalne podobieństwo.

Przerwy

- Def. Przerwa to maksymalny, nieprzerwany ciąg spacji (_) w pojedynczym ciągu kodowym dla danego dopasowania.

np.

```
c  t  t  t  a  a  c  _  _  a  _  a  c
c  _  _  _  c  a  c  c  c  a  t  _  c
```

Przerwy

- Def. Przerwa to maksymalny, nieprzerwany ciąg spacji (_) w pojedynczym ciągu kodowym dla danego dopasowania.

np.

```
c  t  t  t  a  a  c  _  _  a  _  a  c
c  _  _  _  c  a  c  c  c  a  t  _  c
```

Dopasowanie z 7 spacjami i 4 przerwami

Przerwy - waga

- Wprowadzenie przerw ma wpływ na rozkład spacji, a co za tym idzie, całego dopasowania.
- Najprostsze podejście:
  - W_g – stała waga przerwy, niezależna od długości
  - s(x,_) = s(_, x) = 0, dla każdego x

$$\sum_{i=1}^{l} s(S_1'(i), S_2'(i)) - kW_g$$

k – ilość przerw, l – długość łańcucha

Przerwy – znaczenie bioinformatyczne

- Spacje – wstawienie(usunięcie) pojedynczego znaku
- Przerwa – wstawienie(usunięcie) ciągu znaków
  - Często oznacza pojedynczy przypadek mutacji – ważne!
  - Mutacje produkują z dużym prawdopodobieństwem podobne przerwy
  - Niektóre mechanizmy mutacyjne powodujące długie przerwy:
  - Nierówny crossing-over (dodanie do S1, usunięcie z S2)
  - Działanie retrowirusów
  - DNA slippage

Przerwy – znaczenie bioinformatyczne

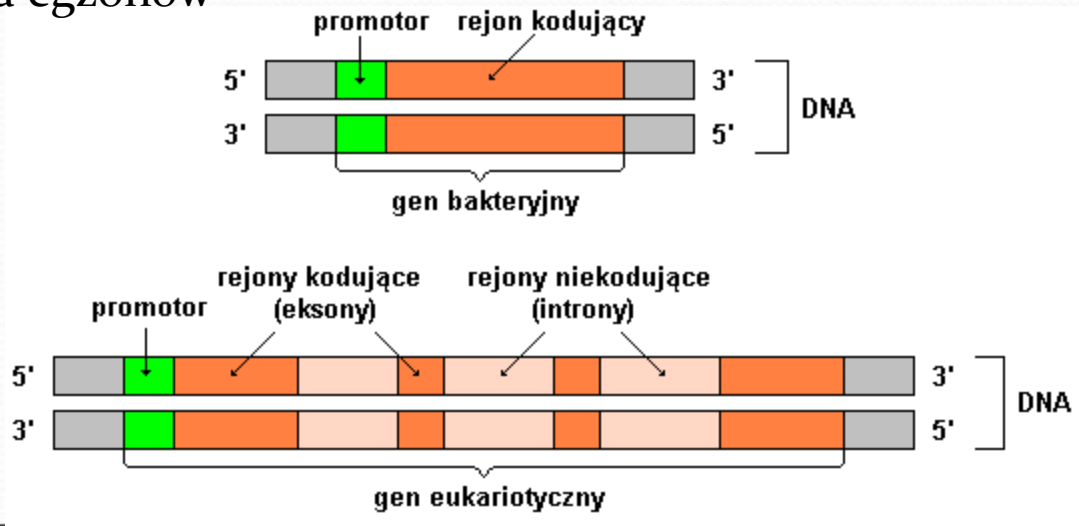
- Dodanie lub usunięcie długiego ciągu – powodujące przerwę występuje znacznie rzadziej niż zwykła zamiana pojedynczego znaku.
- Przerwy niosą ze sobą dużo informacji
- Wspólne przerwy dla pary łańcuchów mogą posłużyć do badania historii ewolucji.
- Dopasowywanie białek – białka są zbudowane z różnych kombinacji aminokwasów, których zbiór jest niewielki. Dlatego 2 białka mogą mieć podobne długie sekwencje, po czym się różnic w pewnym miejscu, które naturalnie pokaże nam przerwa. Dobre dopasowanie przerw daje możliwość dopasowania reszty łańcucha, w tym pojedynczych muacji.

Dopasowanie cDNA

- Jeden łańcuch dużo dłuższy od drugiego
- Kilka regionów o dużym podobieństwie .poprzeplatane długimi przerwami w krótszym z łańcuchów.
- Pasujące regiony mogą zawierać tylko niewielki procent spacji i niedopasowań.

cDNA

- Eukarioty, kod genetyczny zbudowany z naprzemiennie występujących egzonów i intronów.
- Egzony zawierają kod potrzebny do budowy białek
  - Niewielka liczba
- Introny zawierają przypadkowe sekwencje – śmieci
  - Dużo dłuższe od egzonów

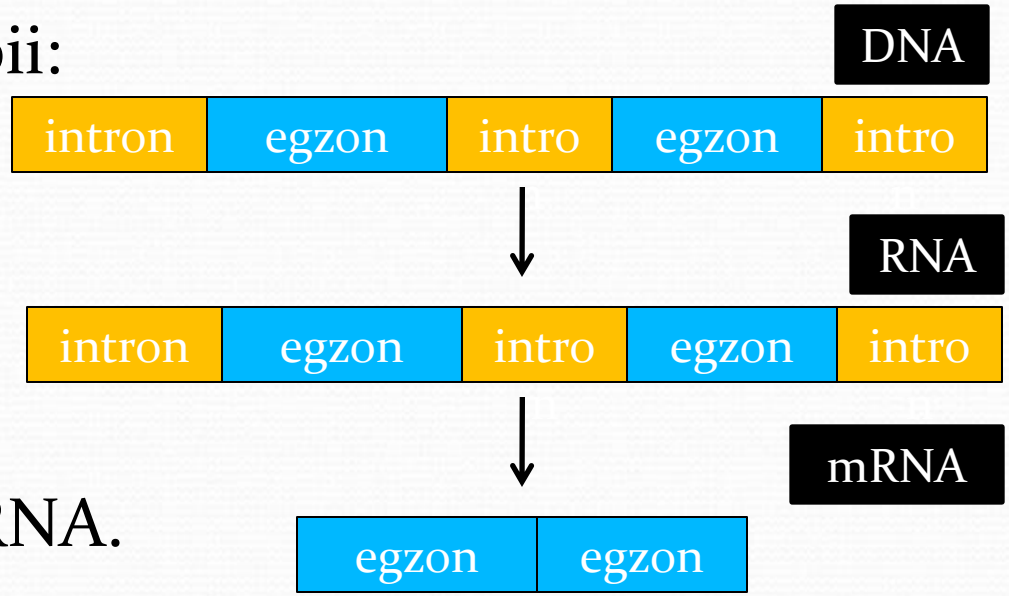


cDNA – powstanie

- Część pojedynczej nici DNA, odpowiadająca danemu genowi jest kopiowana.
- RNA – w uzyskanej kopii:
  - A jest zamieniane na U(uracyl)
  - T->A
  - C->G
  - G->C
- Introny są usuwane.
- Egzony są łączone - mRNA.
- mRNA jest dalej wykorzystywane do tworzenia białka

cDNA – powstanie

- Część pojedynczej nici DNA, odpowiadająca danemu genowi jest kopiowana.
- RNA – w uzyskanej kopii:
  - A jest zamieniane na U(uracyl)
  - T->A
  - C->G
  - G->C
- Introny są usuwane.
- Egzony są łączone - mRNA.
- mRNA jest dalej wykorzystywane do tworzenia białka



cDNA – powstanie c.d.

- Genom w komórkach
  - Komórki wyspecjalizowane (nie wykorzystują całego genomu)
- Cel: Znaleźć białka, które są wytwarzane w komórce, lokalizacja genów.
- Metoda:
  - Wyłapywanie mRNA z danej komórki.
  - cDNA: na podstawie mRNA tworzymy łańcuch odpowiadający DNA.
  - W porównaniu do oryginalnego genu, cDNA zawiera tylko egzony

Biblioteka cDNA

- Kolekcjonowanie genów
- Taksonomia komórek, której geny są skatalogowane

Pseudogeny

- Pseudogen to bliska kopie prawdziwego genu, która mutowała i nie może spełniać swoich funkcji.
- Pełnią ważną ewolucyjną rolę.
- Lokacja może bardzo się różnić od oryginalnego genu.
- Zazwyczaj zawiera introny i egzony.
- Problem sprowadza się do znajdowania powtarzających się podciągów w długim ciągu.

Przetworzone pseudogeny

- Zawierają tylko egzony od swoich przodków
- Przewiduje się, ze powstają w skutek odwrotnej transkrypcji mRNA na DNA (enzym Odwrotna Transkryptaza) i wklejone w losowe miejsce.
- Problem znajdowania jest podobny do znajdowania cDNA, ale trudniejszy.

Algorytm na „najlepsze dopasowanie” dla dowolnej funkcji kary za długość przerwy – dobór kar

- Wybór wagi ma krytyczny wpływ na efektywność algorytmu dopuszczającego przerwy.
- Zbyt duża kara oznaczałaby dopasowanie z kilkoma przerwami i niewielką liczbą podciągów.
- Małe kary pozwalają na bardziej podzielone dopasowania.
- Główne typy kar:
  - Stała
  - Afiniczna(liniowa)
  - Wypukła
  - Arbitralna

Algorytm na „najlepsze dopasowanie” dla stałej funkcji kary za długość przerwy

- W_g – stała kara, niezależna od długości przerwy
  - Spacje są wolne od kar: `s(x,_)=s(_,x)=0`, dla każdego x
- W_g (#przerwy)- kara za przerwy (*gap*)
- W_m (#zgodności) – waga zgodności(*matching*)
- W_{ms} (#niezgodności) – kara za niedopasowanie (*missmatching*)

- Znaleźć dopasowanie *A* maksymalizujące:

W_m(#zgodności) - W_{ms}(#niezgodności) - W_g(#przerwy)

$$\sum_{i=1}^{l} [s(S_1'(i), S_2'(i))] - W_g(\#\ przerwy\quad)$$

Algorytm na „najlepsze dopasowanie” dla afinicznej (liniowej) funkcji kary za długość przerwy

- Rozszerza model stały o zmienną W_s – ilość spacji w danej przerwie.
  - W_g – kara początkowa,
  - W_s – kara za rozszerzenie przerwy
- W_g+qW_s

- Znaleźć dopasowanie maksymalizujące:

W_m(#zgodności) - W_{ms}(#niezgodności) - W_g(#przerwy) – W_g(#spacje)

$$\sum_{i=1}^{l} [s(S_1'(i), S_2'(i))] - W_g(\#\ przerwy\ ) - W_s(\#\ spacje\ )$$

Algorytm na „najlepsze dopasowanie” dla afinicznej (liniowej) funkcji kary za długość przerwy

- Prawdopodobnie najczęściej używany model.

- Domyślnie $W_g = 10$, $W_s = 2$

Algorytm na „najlepsze dopasowanie” dla wypukłej funkcji kary za długość przerwy

- Niektóre zjawiska biologiczne są lepiej modelowane przez funkcję wagową, gdzie każda kolejna spacja dodana do przerwy mniej wpływa na karę.
- Przykład: $W_g + \log_e q$, gdzie q to długość przerwy

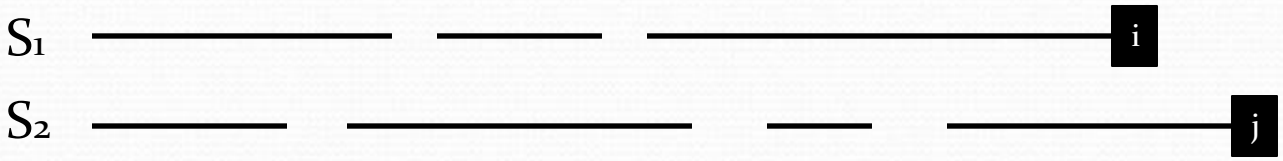
Algorytm na „najlepsze dopasowanie” dla arbitralnej funkcji kary za długość przerwy

- Karę za przerwę opisuje funkcja arbitralna: ω(q), gdzie q – długość.

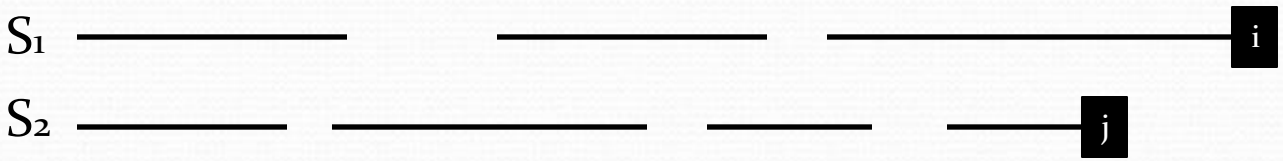
- Pozostałe modele są przypadkami modelu arbitralnego.

Algorytm na „najlepsze dopasowanie” dla dowolnej funkcji kary za długość przerwy – przypadki dopasowań

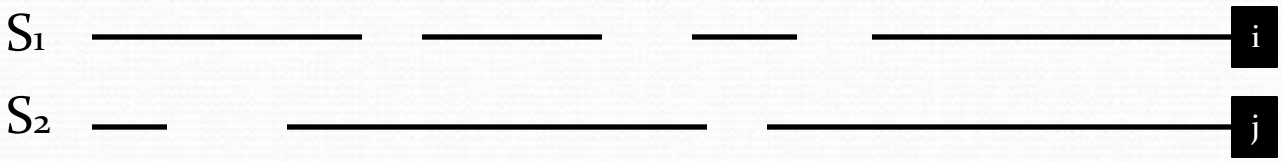
- Rozróżniamy 3 typy dopasowań prefiksów $S_1[1...i]$ oraz $S_2[1...j]$:
- 1. **Lewostronne**, gdzie $S_1(i)$ jest dopasowany do elementu po lewej stronie $S_2(j)$. Dopasowanie kończy przerwa w S_1.



- 2. **Prawostronne**, gdzie $S_1(i)$ jest dopasowany do elementu po prawej stronie $S_2(j)$. Dopasowanie kończy przerwa w S_2.



- 3. **Równomierne**, w którym $S_1(i)$ odpowiada $S_2(j)$. To znaczy, że $S_1(i) = S_2(j)$ lub $S_1(i) \neq S_2(j)$.



Algorytm na „najlepsze dopasowanie” dla dowolnej funkcji kary za długość przerwy – oznaczenia

- `E(i, j)` maksymalna wartość dopasowania typu 1.
- `F(i, j)` maksymalna wartość dopasowania typu 2.
- `G(i, j)` maksymalna wartość dopasowania typu 3.
- `V(i, j)` maksymalna wartość z

  `{E(i, j), F(i, j), G(i, j)}.`

Algorytm na „najlepsze dopasowanie” dla arbitralnej funkcji kary za długość przerwy – rekurencje

$$V(i, j) = \max[\ E(i, j), F(i, j), G(i, j)]$$

$$G(i, j) = V(i-1, j-1) + s(S_1(i), S_2(j))$$

$$E(i, j) = \max_{0 \le k \le j-1} [V(i, k) - \omega(j-k)]$$

$$F(i, j) = \max_{0 \le l \le i-1} [V(l, j) - \omega(i-l)]$$

- złożoność obliczeniowa(czasowa):

$$O(nm^2 + n^2 m)$$

Algorytm na „najlepsze dopasowanie” dla arbitralnej funkcji kary za długość przerwy - złożoność

`G(i,j)` – jedna komórka `(V(i-1;j-1))`

`E(i,j)` – j komórek z wiersza j `(V(i,0)-V(i,j-1))`

`F(i,j)` – i komórek z kolumny j `(V(0,j)-V(i-1,j))`

$m(m+1)/2=\Theta(m^2)$ – operacji do obliczenia całego wiersza
$n(n+1)/2 = \Theta(n^2)$ – operacji do obliczenia całej kolumny

należy obliczyć n wierszy oraz m kolumn

- złożoność obliczeniowa(czasowa):

$$O(nm^2 + n^2 m)$$

Algorytm na „najlepsze dopasowanie” dla arbitralnej funkcji kary za długość przerwy – wartości początkowe

- Jeżeli wszystkie spacje są zawarte w dopasowaniu:
  - Optimum w komórce (n, m) , gdzie n i m to długości łańcuchów
  - Warunki początkowe: $V(i,0) = -\omega(i) \qquad V(0,\ j) = -\omega(j)$

    $E(i,0) = -\omega(i) \quad F(0,\ j) = -\omega(j)$

- Jeżeli spacje(przerwy) brzegowe są wolne:
  - Optymalną wartością jest max wartość w wierszu n lub kolumnie m
  - Wartości bazowe:

$$V(i,0) = 0 \qquad V(0,\ j) = 0$$

Algorytm na „najlepsze dopasowanie” dla afinicznej(lub stałej) funkcji kary za długość przerwy - rekurencje

$$V(i,\ j) = \max[\ E(i,\ j), F(i,\ j), G(i,\ j)]$$

$$G(i,\ j) = \begin{cases} V(i-1,\ j-1) + W_m\,,\ S_1(i) = S_2(j) \\ V(i-1,\ j-1) - W_{ms}\,,\ S_1(i) \neq S_2(j) \end{cases}$$

$$E(i,\ j) = \max[\ E(i,\ j-1), (V(i,\ j-1) - W_g\ ] - W_s$$

$$F(i,\ j) = \max[\ F(i-1,\ j), (V(i-1,\ j) - W_g\ ] - W_s$$

- złożoność obliczeniowa(czasowa):

$$O(nm)$$

Algorytm na „najlepsze dopasowanie” dla afinicznej(lub stałej) funkcji kary za długość przerwy – wartości początkowe

- Jeżeli wszystkie spacje są zawarte w dopasowaniu:
  - Optimum w komórce (n, m) , gdzie n i m to długości łańcuchów
  - Warunki początkowe: $V(i,0) = E(i,0) = -W_g - iW_s$

    $V(0, j) = F(0, j) = -W_g - jW_s$

- Jeżeli spacje(przerwy) brzegowe są wolne:
  - Optymalną wartością jest max wartość w wierszu n lub kolumnie m
  - Wartości bazowe:

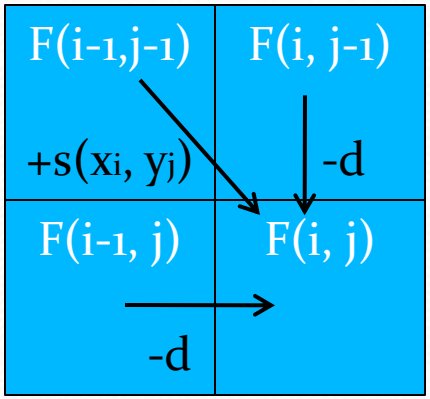
$$V(i,0) = V(0, j) = 0$$

Algorytm Needleman-Wunsch

- Konstruujemy macierz *F* indeksowaną przez `i` oraz `j`.
- `F(i, j)` jest wynikiem najlepszego dopasowania inicjalnych segmentów: `x[1..i]` oraz `y[1..j]`.
- Inicjujemy `F(0,0)=0`.
- Uzupełniamy tablicę od lewego górnego rogu.
- Jeżeli znamy `F(i-1,j-1)`, `F(i,j-1)`, `F(i-1,j)`, możemy obliczyć `F(i,j)`.
- Wartości początkowe:
  - `F(i,0)=-id`
  - `F(0,j)=-jd`
- `F(n, m)` – najlepszy wynik dopasowania dla `x[1..n]` i `y[1..m]`

Algorytm Needleman-Wunsch

- Współczynnik d=8
- Obliczanie rekurencyjne kolejnej komórki:
  - $F(i,j) = \max[\ F(i-1,j-1)+s(x_i,y_j),\ F(i-1,j)-d,\ F(i,j-1)-d\ ]$



Algorytm Needleman-Wunsch traceback

- Trasa pozwalająca na odtworzenie sekwencji
- Tworzy się ją od tyłu idąc po ścieżce do komórki, z której bieżąca komórka dziedziczy.
- Podczas szukania ścieżki odtwarzamy dopasowanie odkładając na początek parę symboli w sposób:
  - Jeżeli krok idzie do komórki (i-1, j-1) to odkładamy x_i, y_j.
  - Jeżeli do (i-1, j) to x_i oraz „_”.
  - Jeżeli do (i, j-1) to „_” oraz y_j.
- Kończymy procedurę jak osiągniemy początek, tj. `i=j=0`.

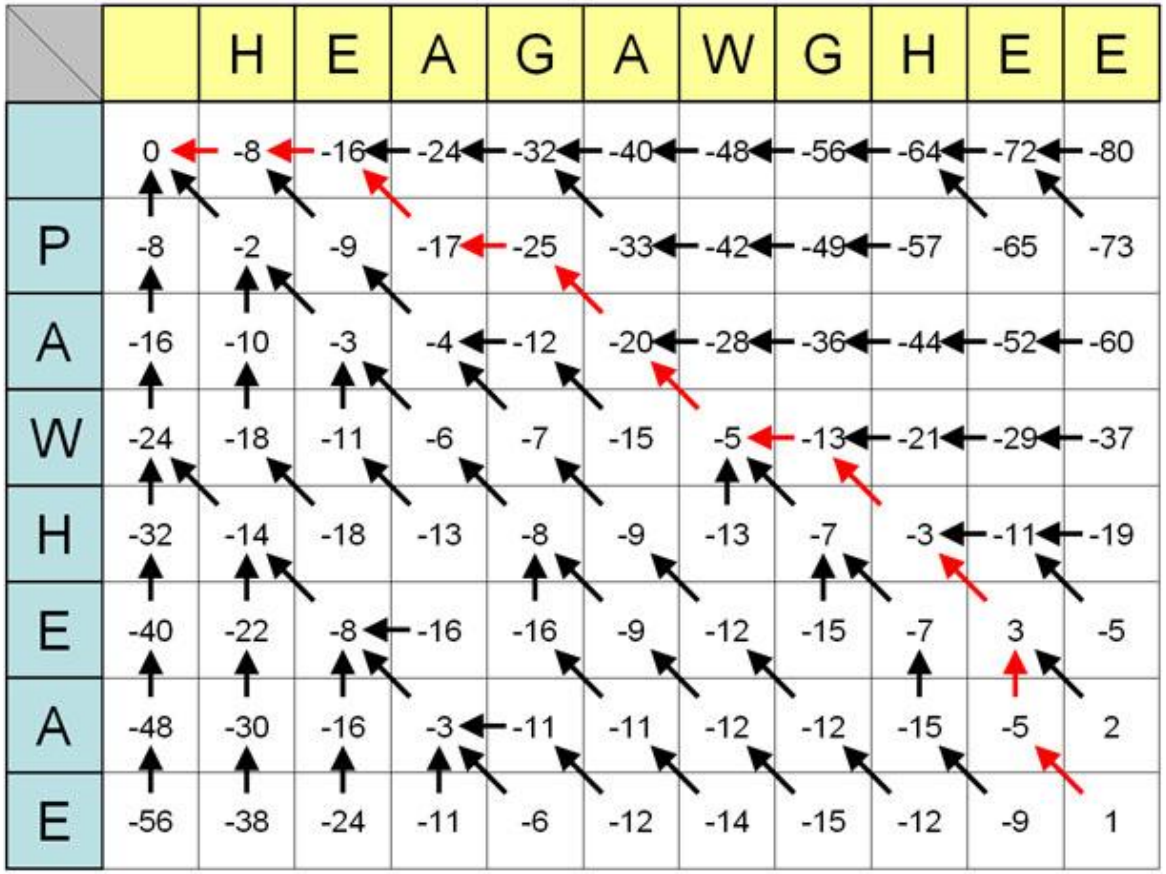
Algorytm Needleman-Wunsch tablica BLOSUM50

- Blocks Substitution Matrix
- Tablica prezentująca wagi mutacji aminokwasów.
- Powstała na podstawie obserwacji częstości mutacji.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	**5**	−2	−1	−2	−1	−1	−1	0	−2	−1	−2	−1	−1	−3	−1	1	0	−3	−2	0
R	−2	**7**	−1	−2	−4	1	0	−3	0	−4	−3	3	−2	−3	−3	−1	−1	−3	−1	−3
N	−1	−1	**7**	2	−2	0	0	0	1	−3	−4	0	−2	−4	−2	1	0	−4	−2	−3
D	−2	−2	2	**8**	−4	0	2	−1	−1	−4	−4	−1	−4	−5	−1	0	−1	−5	−3	−4
C	−1	−4	−2	−4	**13**	−3	−3	−3	−3	−2	−2	−3	−2	−2	−4	−1	−1	−5	−3	−1
Q	−1	1	0	0	−3	**7**	2	−2	1	−3	−2	2	0	−4	−1	0	−1	−1	−1	−3
E	−1	0	0	2	−3	2	**6**	−3	0	−4	−3	1	−2	−3	−1	−1	−1	−3	−2	−3
G	0	−3	0	−1	−3	−2	−3	**8**	−2	−4	−4	−2	−3	−4	−2	0	−2	−3	−3	−4
H	−2	0	1	−1	−3	1	0	−2	**10**	−4	−3	0	−1	−1	−2	−1	−2	−3	2	−4
I	−1	−4	−3	−4	−2	−3	−4	−4	−4	**5**	2	−3	2	0	−3	−3	−1	−3	−1	4
L	−2	−3	−4	−4	−2	−2	−3	−4	−3	2	**5**	−3	3	1	−4	−3	−1	−2	−1	1
K	−1	3	0	−1	−3	2	1	−2	0	−3	−3	**6**	−2	−4	−1	0	−1	−3	−2	−3
M	−1	−2	−2	−4	−2	0	−2	−3	−1	2	3	−2	**7**	0	−3	−2	−1	−1	0	1
F	−3	−3	−4	−5	−2	−4	−3	−4	−1	0	1	−4	0	**8**	−4	−3	−2	1	4	−1
P	−1	−3	−2	−1	−4	−1	−1	−2	−2	−3	−4	−1	−3	−4	**10**	−1	−1	−4	−3	−3
S	1	−1	1	0	−1	0	−1	0	−1	−3	−3	0	−2	−3	−1	**5**	2	−4	−2	−2
T	0	−1	0	−1	−1	−1	−1	−2	−2	−1	−1	−1	−1	−2	−1	2	**5**	−3	−2	0
W	−3	−3	−4	−5	−5	−1	−3	−3	−3	−3	−2	−3	−1	1	−4	−4	−3	**15**	2	−3
Y	−2	−1	−2	−3	−3	−1	−2	−3	2	−1	−1	−2	0	4	−3	−2	−2	2	**8**	−1
V	0	−3	−3	−4	−1	−3	−3	−4	−4	4	1	−3	1	−1	−3	−2	0	−3	−1	**5**

Algorytm Needleman-Wunsch przykład

- d=8, jako s(x, y) przyjmiemy wartości z tablicy BLOSUM50
- Sekwencje:
  - HEAGAWGHEE
  - PAWHEAE



- Jeden z wyników optymalnych:

```
HEAGAWGHE-E
--P-AW-HEAE
```

Optymalnych wyników może być więcej!

Bibliografia:

- Dan Gusfield, ***Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology***

- Michael S. Waterman, ***Introduction to Computational Biology: Maps, sequences and genomes***

- ***Wikipedia.org***