Notatki do wykładu "ALGORYTMY BIOLOGII MOLEKULARNEJ"

Jerzy Tiuryn
Instytut Informatyki
Uniwersytet Warszawski
rok akademicki 1999/2000

27 października 1999

Spis rzeczy

2 Podobieństwo dwóch sekwencji **14**
2.1 Globalne uliniowienie 15
2.1.1 Metoda dynamicznego programowania 17
2.1.2 Odtwarzanie optymalnych uliniowań 19
2.1.3 Odległość edycyjna 22

2 Podobieństwo dwóch sekwencji

Motywacje

Odkrywanie podobieństw pomiędzy dwoma sekwencjami ma fundamentalne znaczenie w biologii molekularnej. Jest to związane z obserwacją. że wiele podobnych sekwencji ma podobne cechy funkcjonalne lub strukturalne (odwrotna implikacja nie jest prawdziwa – istnieją białka mające podobne cechy funkcjonalne i strukturalne, ktorych sekwencje aminokwasów są zupełnie niepodobne). Na przykład, znane są geney występujące u różnych gatunków (np. muszka owocowa i człowiek) mające zadziwiające podobieństwo. Białka kodowane przez te geny pełnią podobne funkcje i mają podobny kształt 3D.

Geny u różnych gatunków mogą być podobne jeśli obydwa gatunki wywodzą się ze wspólnego korzenia. Wówczas geny praprzodka, w procesie

ewolucji, mogły się przekształcić w podobne, ale nie identyczne, geny osobników różnych odgałęzień. Przyjmuje się, że większe podobieństwo pomiędzy genami (genomami) różnych gatunków jest wskazówką na to, że te gatunki znajdują się bliżej w drzewie ewolucji. Ponieważ genomowy DNA danego gatunku jest całą informacją (genetyczną) o tym gatunku, zatem ewolucja jest związana ze zmianami DNA. Procesy związane z tymi zmianami są przedmiotem badań nowej dziedziny zwanej ewolucją molekularną.

Najprostsze zjawiska prowadzące do zmian DNA to *mutacje punktowe* polegające na zamianie jednego nukleotydu na inny, wypadnięciu bądź wstawieniu nowego nukleotydu. Podobne zjawiska zachodzą na poziomie RNA. Zjawiska te są wywołane błędami w kopiowaniu bądź zewnętrznymi warunkami (np. promieniowanie radioaktywne).

W tej części wykładu będziemy się zajmować dwoma typami problemów. Dane mamy dwa ciągi S_1 i S_2,

- **globalne uliniowienie** – porównujemy ze sobą S_1 i S_2 (możemy wstawiać spacje do obydwu ciągów) tak, aby zmaksymalizować ich podobieństwo.
- **lokalne uliniowienie** – poszukujemy fragmentów ciągów S_1 i S_2 o maksymalnym podobieństwie (do badanych fragmentów możemy wstawiać spacje).

Trochę notacji dotyczącej słów

Niech Σ oznacza skończony zbiór. Przez Σ^* będziemy oznaczać zbiór wszystkich skończonych ciągów (czyli *słów*) o elementach z Σ. Zbiór Σ będzie nazywany *alfabetem* a jego elementy *literami*. Szczególnym słowem jest *puste słowo*, oznaczane ε. Dla dowolnego słowa $S \in \Sigma^*$, przez $|S|$ będziemy oznaczać *długość* tego słowa, czyli liczbę liter w nim występujących. Dla słów $S_1, S_2 \in \Sigma^*$, przez S_1S_2 będziemy oznaczać wynik dopisania S_2 z prawej strony do S_1. Dla $1 \leq i \leq |S|$, przez $S(i)$ będziemy oznaczać i-tą literę słowa S. Powiemy, że słowo T jest em podsłowem słowa S, gdy istnieją słowa T_1, T_2, takie że $S = T_1TT_2$. Słowa T_1 i T_2 są wyznaczone jednoznacznie prze S i T. W sytuacji, gdy w powyższym rozkładzie $T_1 = \varepsilon$, to powiemy, że T jest *prefiksem* słowa S. Natomiast, gdy $T_2 = \varepsilon$, to powiemy, że T jest *sufiksem* słowa S.

2.1 Globalne uliniowienie

Pojęcie globalnego uliniowienia zostało wprowadzone w 1970 przez Needleman'a i Wunsch'a. Opiera się ono na pojęciu uliniowienia oraz funkcji podo-

bieństwa.

Dane dwa słowa $S_1, S_2 \in \Sigma^*$. Będziemy zakładać, że '–' jest specjalnym symbolem (*spacja*) nie należącym do Σ.[1] *Globalne uliniowienie* dla pary słów (S_1, S_2) to każda taka para słów $(S_1^\#, S_2^\#) \in (\Sigma \cup \{-\})^* \times (\Sigma \cup \{-\})^*$, która spełnia następujące trzy warunki:

- S_1 otrzymuje się z $S_1^\#$ przez usunięcie wszystkich symboli –. Podobnie S_2 otrzymuje sie z $S_2^\#$.
- $|S_1^\#| = |S_2^\#|$.
- Dla każdego $1 \leq i \leq |S_1^\#|$, symbole $S_1^\#(i)$ oraz $S_2^\#(i)$ nie są jednocześnie równe –.

Zadanie 2.1.1 Pokazać, że liczba możliwych uliniowań dla danych słów rośnie wykładniczo wraz z ich rozmiarem.

Alfabety występujące w zastosowaniach w biologii to: $\Sigma = \{A, C, G, T\}$ (dla porównywania DNA); $\Sigma = \{A, C, G, U\}$ (dla porównywania RNA) oraz $\Sigma =$ aminokwasy (dla porównywania białek).

Funkcja podobieństwa to pewna funkcja $s : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \longrightarrow \mathbb{R}$ mająca charakter kary/nagrody za odpowiadające sobie symbole. W zastosowaniach zwykle przyjmuje się, że dla $x, y \in \Sigma \cup \{-\}$,

$$s(x, x) > 0, \text{ o ile } x \neq -,$$

$$s(x, y) = s(y, x),$$

$$s(x, y) < s(x, x), \text{ o ile } x \neq y \text{ oraz } x \neq -,$$

$$s(x, -) < 0.$$

Nie istnieje jedna funkcja podobieństwa dobra dla wszystkich zastosowań. W literaturze jest dużo prac na temat jak dobierać s dla porównywania białek lub DNA, dla różnych zastosowań.

Przykład 2.1.1 Prostym przykładem funkcji podobieństwa jest s zdefiniowane następująco: $s(x, -) = -2$, dla $x \in \Sigma$ oraz dla $x, y \in \Sigma$,

$$s(x, y) = \begin{cases} 1, & \text{gdy } x = y \\ -1, & \text{gdy } x \neq y \end{cases}$$

Będziemy tej funkcji używać w przykładach. □

[1] Nazwy 'spacja' będziemy używać dla zaznaczenia, że chodzi o miejsce po wypadniętym nukleotydzie (lub aminokwasie).

Mając daną funkcję podobieństwa oraz uliniowienie $(S_1^\#, S_2^\#)$ definiujemy *podobieństwo* tego uliniowienia jako

$$\sum_{i=1}^{n} s(S_1^\#(i), S_2^\#(i)),$$

gdzie $n = |S_1^\#| = |S_2^\#|$.
Uwaga: gdy $n = 0$, to powyższa suma przyjmuje wartość 0.

Przykład 2.1.2 Weźmy, na przykład słowa $S_1 = CACTGT$ oraz $S_2 = CAGGTG$.

Jedno możliwe uliniowienie to $S_1^\# = S_1$ oraz $S_2^\# = S_2$. Podobieństwo tego uliniowienia wynosi -2.

Natomiast dla uliniowienia $S_1^\# = CAC–TGT$ oraz $S_2^\# = CAGGTG–$ dostajemy podobieństwo -1. □

Globalne podobieństwo dwóch ciągów $S_1, S_2 \in \Sigma^*$ to maksymalne podobieństwo uliniowienia, brane po wszystkich uliniowieniach:

$$sim(S_1, S_2) = max\{\sum_{i=1}^{n} s(S_1^\#(i), S_2^\#(i)) \mid (S_1^\#, S_2^\#) \text{ jest uliniowieniem dla } (S_1, S_2) \text{ oraz } n = |S_1^\#| = |S_2^\#|\}.$$

Globalne podobieństwo dwóch ciągów nazywa się też *wartością optymalnego uliniowienia*, a każde uliniowienie realizujące tę wartość nazywa się *optymalnym uliniowieniem*. W ogólności dana para słow może mieć więcej niż jedno optymalne uliniowienie.

2.1.1 Metoda dynamicznego programowania

Metoda ta posłuży nam do znalezienia szybkiego algorytmu obliczającego wartość globalnego podobieństwa dla dowolnych dwóch słów S_1, S_2. Zadanie to będziemy rozwiązywać dla dowolnej funkcji podobieństwa s. Główna idea tej metody polega na sukcesywnym obliczaniu poszukiwanej wartości, opierając się na wartościach obliczonych dla pewnych mniejszych podzadań. Te pośrednie wartości są przechowywane w tablicy.

Dla $0 \leq i \leq |S_1|$, niech $S_1[1..i]$ oznacza podsłowo słowa S_1 składające się z pierwszych i liter, czyli $S_1(1) \ldots S_1(i)$. Podobnie dla $S_2[1..i]$. Niech $V(i, j)$ będzie globalnym podobieństwem dla słów $S_1[1..i]$ oraz $S_2[1..j]$. Niech $m = |S_1|$, $n = |S_2|$.

Jak obliczyć $V(i,j)$? Oczywiście mamy $V(0,0) = 0$ oraz

$$V(0,j) = \sum_{k=1}^{j} s(-, S_2(k)),$$

$$V(i,0) = \sum_{k=1}^{i} s(S_1(k), -).$$

Kluczowa obserwacja polega na zauważeniu, że aby policzyć $V(i,j)$ dla $i > 0$ oraz $j > 0$, wystarczy znać wartości $V(i-1,j)$, $V(i-1,j-1)$ oraz $V(i,j-1)$.

Twierdzenie 2.1.1 *Dla $0 < i \leq m$ oraz $0 < j \leq n$, mamy:*

$$V(i,j) = max[V(i-1,j-1) + s(S_1(i), S_2(j)),\ V(i-1,j) + s(S_1(i), -),\\ V(i,j-1) + s(-, S_2(j))].$$

Dowód: Niech V oznacza liczbę po prawej stronie równości w tezie twierdzenia. Niech (T_1, T_2) będzie dowolnym uliniowieniem dla $(S_1[1..i], S_2[1..j])$. Jeśli ostatni symbol w T_1 i w T_2 jest literą z Σ, to podobieństwo tego uliniowienia jest nie wększe od $V(i-1,j-1) + s(S_1(i), S_2(j)) \leq V$. Jeśli, na przykład, ostatnim symbolem w T_1 jest $-$, to ostatnim symbolem w T_2 musi być $S_2(j)$. Zatem podobieństwo tego uliniowienia jest nie wększe od $V(i,j-1) + s(-, S_2(j)) \leq V$. Podobnie postępujemy w przypadku, gdy ostatnim symbolem w T_2 jest $-$. Tak więc udowodniliśmy, że $V(i,j) \leq V$.

Na odwrót, jeśli weźmiemy optymalne uliniowienie (T_1, T_2) dla par kolejnych słów

- $S_1[1..i-1]$, $S_2[1..j-1]$;
- $S_1[1..i-1]$, $S_2[1..j]$;
- $S_1[1..i]$, $S_2[1..j-1]$,

to dopisując z prawej strony:

- w pierszym przypadku: $S_1(i)$ do T_1 oraz $S_2(j)$ do T_2;
- w drugim przypadku: $S_1(i)$ do T_1 oraz – do T_2;
- w trzecim przypadku: – do T_1 oraz $S_2(j)$ do T_2,

otrzymamy w każdym przypadku uliniowanie dla słów $(S_1[1..i], S_2[1..j])$, a zatem wartość podobieństwa tego uliniowania jest nie wieksza od $V(i,j)$. Zatem V, będące maksimum z tych wartości, jest nie większe od $V(i,j)$. ■

Poprawność poniższego algorytmu jest oparta na Twierdzeniu 2.1.1.

Wejście: słowa $S_1, S_2 \in \Sigma^*$
Wynik: globalne podobieństwo, $sim(S_1, S_2)$

$m := |S_1|$;
$n := |S_2|$;
$V(0,0) := 0$;
for $i = 1$ **to** m **do**
    $V(i,0) := V(i-1,0) + s(S_1(i), -)$;
for $j = 1$ **to** n **do**
    $V(0,j) := V(0,j-1) + s(-, S_2(j))$;
for $i = 1$ **to** m **do**
    for $j = 1$ **to** n **do**
        $V(i,j) := max[V(i-1,j-1) + s(S_1(i), S_2(j)),\ V(i-1,j) + s(S_1(i), -),$
                $V(i,j-1) + s(-, S_2(j))]$;
return $V(m,n)$;

Algorytm 2.1.1: Obliczanie globalnego podobieństwa.

Obliczmy czas działania Algorytmu 2.1.1. Pierwsza pętla **for** wykonywuje $m-1$ kroków. Druga pętla wykanuje $n-1$ kroków. Natomiast trzecia pętla (ze względu na zagnieżdżenie pętli) wykonuje $(m-1)(n-1)$ kroków. Zatem łącznie program wykona $O(mn)$ kroków. Pamięć użyta przez ten program jest $O(mn)$, bo tyle miejsca zajmuje tablica V. Można łatwo poprawić użycie pamięci do $O(min(m,n))$, zapamiętując tylko cały porzedni wiersz (bądź kolumnę) w celu obliczenia następnego nowego wiersza (kolumny).

2.1.2 Odtwarzanie optymalnych uliniowań

Aby odtworzyć optymalne rozwiązanie, w miejscu (i,j) w tablicy V wystarczy umieścić informację o miejscach, z których pochodzi obliczane maksimum. Z Twierdzenia 2.1.1 wynika, że $V(i,j)$ jest równe maksimum z następujących trzech wartości:

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

- $V(i, j-1) + s(-, S_2(j))$.

Jeśli $V(i,j)$ jest równe

- pierwszej z powyższych trzech wartości, to wpisujemy w miejscu (i,j) symbol $\nwarrow$,
- drugiej z powyższych trzech wartości, to wpisujemy w miejscu (i,j) symbol $\uparrow$,
- trzeciej z powyższych trzech wartości, to wpisujemy w miejscu (i,j) symbol $\leftarrow$.

Oczywiście może się tak zdażyć, że w tablicy w jednym miejscu znajdą się dwa (a nawet trzy) symbole. Dodatkowo w pozycjach $(i,0)$, dla $i > 0$, wpisujemy symbol $\uparrow$. Natomiast w pozycjach $(0,j)$, dla $j > 0$, wpisujemy $\leftarrow$.

Po wpisaniu strzałek we wszystkie miejsca tablicy optymalne uliniowienie otrzymuje się przez wybranie dowolnej drogi z miejsca (m,n) do miejsca $(0,0)$. Przy czym przejście z miejsca (i,j) do (i',j') jest możliwe tylko wtedy, gdy w miejscu (i,j) znajduje się strzałka pokazująca na miejsce (i',j'). Na przykład, gdy pozycja (i,j) zawiera strzałki $\nwarrow$ oraz $\uparrow$, to możemy przejść z (i,j) tylko do $(i-1,j-1)$ lub do $(i-1,j)$. Na razie załóżmy, że taka droga istnieje.

Mając wybraną taką drogę D,[2] uliniowienie odpowiadające D konstruujemy od prawej do lewej w następujący sposób. Na początku mamy dwa puste ciągi (tworzące sufiksy konstruowanego uliniowienia) oraz aktualną pozycją (najbardziej prawą pozycją jeszcze nie rozpatrywaną) w D jest (m,n). W ogólności załóżmy, że T_1, T_2 są już skonstruowanymi sufiksami uliniowienia oraz (i,j) jest najbardziej prawą pozycją w D dotąd nie rozpatrywaną. Jeśli $i = 0 = j$, to T_1, T_2 jest poszukiwanym uliniowieniem. W przeciwnym przypadku, jeśli następną pozycją w D jest (i',j') to mamy następujące możliwości:

- $i' = i-1$ oraz $j' = j-1$. Wówczas do T_1 dopisujemy z lewej strony $S_1(i)$, a do T_2 dopisujemy z lewej strony $S_2(j)$.
- $i' = i-1$ oraz $j' = j$. Wówczas do T_1 dopisujemy z lewej strony $S_1(i)$, a do T_2 dopisujemy z lewej strony $-$.
- $i' = i$ oraz $j' = j-1$. Wówczas do T_1 dopisujemy z lewej strony $-$, a do T_2 dopisujemy z lewej strony $S_2(j)$.

[2] Czyli taki ciąg par (i,j) od (m,n) do $(0,0)$, że jeśli (i',j') stoi bezpośrednio za (i,j) to pozycja (i,j) w tablicy V musi zawierać strzałkę skierowaną w stronę (i',j').

Otrzymujemy w ten sposób nową parę słów, a następną pozycją, którą będziemy rozważać w następnym kroku jest (i', j').

Twierdzenie 2.1.2 *Jeśli D jest drogą od (m, n) do $(0, 0)$ zbudowaną przy użyciu strzałek, to uliniowienie wyznaczone przez tę drogę jest optymalne.*

Dowód: Niech (T_1, T_2) będzie uliniowieniem wyznaczonym przez D. Dowodzimy następującą, nieco ogólniejszą własność. Dla każdego $0 \leq k \leq |D|$, jeśli para (i, j) stoi na k-tym miejscu w D, to wartość podobieństwa dla uliniowienia $(T_1[1..k], T_2[1..k])$ jest równa $V(i, j)$. Oczywistą indukcję ze względu na k pozostawiamy czytelnikowi. Zatem wartość podobieństwa dla (T_1, T_2) wynosi $V(m, n)$, czyli uliniowienie to jest optymalne. ∎

Zauważmy, że z powyższego twierdzenia nie wynika czy taka droga od (m, n) do $(0, 0)$ musi istnieć. Również nie jest oczywiste czy każde optymalne uliniowienie dla S_1, S_2 możemy otrzymać tą metodą.

Twierdzenie 2.1.3 *Dla każdego optymalnego uliniowienia $(S_1^{\#}, S_2^{\#})$ dla słów (S_1, S_2) istnieje droga D od (m, n) do $(0, 0)$, wyznaczona przy pomocy w/w reguł i taka, że $(S_1^{\#}, S_2^{\#})$ jest wyznaczone przez D.*

Przykład 2.1.3 Znajdziemy wszystkie optymalne uliniowienia dla słów $S_1 = ATTGC$ oraz $S_2 = ATGC$. Używamy funkcji podobieństwa z Przykladu 2.1.1.

			A		T		G		C
		0	←	-2	←	-4	←	-6	←	-8
A	↑	-2	↖	1	←	-1	←	-3	←	-5
T	↑	-4	↑	-1	↖	2	←	0	←	-2
T	↑	-6	↑	-3	↖↑	0	↖	1	↖←	-1
G	↑	-8	↑	-5	↑	-2	↖	1	↖	0
C	↑	-10	↑	-7	↑	-4	↑	-1	↖	2

Mamy więc dwa optymalne uliniowienia o wartości 2:

$S_1^{\#} = ATTGC$
$S_2^{\#} = A\text{–}TGC$, oraz

$S_1^{\#} = ATTGC$
$S_2^{\#} = AT\text{–}GC$ □

Zadanie 2.1.2 Wyznaczyć wszystkie optymalne uliniowienia dla słów z Przykładu 2.1.2.

Zadanie 2.1.3 Obliczyć globalne podobieństwo słów $S_1 = CAGTATTCGCA$, $S_2 = AAGTTAGCAG$ dla funkcji podobieństwa $s(x, -) = -1$,

$$s(x, y) = \begin{cases} 1 & \text{gdy } x = y, \\ -1 & \text{gdy } x \neq y. \end{cases}$$

Zadanie 2.1.4 Udowodnić Twierdzenie 2.1.3.

Zadanie 2.1.5 (Hirschberg) Algorytm znajdujący optymalne uliniowienie, opisany w notatkach używa pamięci $O(mn)$. Znaleźć algorytm znajdujący optymalne uliniowienie, działający w czasie $O(mn)$ i w pamięci liniowej od rozmiaru słów.

Zadanie 2.1.6 * Znaleźć jakiekolwiek nietrywialne ograniczenie dolne na czas potrzebny do obliczania globalnego podobieństwa dwóch słów. Jak ten czas może zależeć od wyboru funkcji podobieństwa s?

2.1.3 Odległość edycyjna

Pojęcie odległości edycyjnej pomiędzy słowami zostało zaproponowane przez Levensteina w 1966r. Pojęcie to jest oparte na minmalnej liczbie mutacji, które przeprowadzają jedno słowo w drugie. Rozważmy cztery typy operacji na słowach:

- Wstawienie litery (I);
- Usunięcie litery (D);
- Zamiana liter (R);
- Pozostawienie litery nie zmienionej (M).

Niech $\mathcal{T} \in \{D, I, M, R\}^*$ będzie danym ciągiem operacji na słowach oraz niech $S_1, S_2 \in \Sigma^*$ będą dowolnymi słowami. Zdefinujemy relację $\mathcal{T} : S_1 \mapsto S_2$ (czytamy: "$\mathcal{T}$ przekształca S_1 w S_2") przez indukcję ze względu na $|\mathcal{T}|$.

- $\varepsilon : S_1 \mapsto S_2 \iff S_1 = S_2 = \varepsilon$;
- $I\mathcal{T} : S_1 \mapsto S_2 \iff$ istnieją $x \in \Sigma$ oraz $S_2' \in \Sigma^*$ takie, że $S_2 = xS_2'$ oraz $\mathcal{T} : S_1 \mapsto S_2'$;

- $D\mathcal{T} : S_1 \mapsto S_2 \iff$ istnieją $x \in \Sigma$ oraz $S_1' \in \Sigma^*$ takie, że $S_1 = xS_1'$ oraz $\mathcal{T} : S_1' \mapsto S_2$;
- $R\mathcal{T} : S_1 \mapsto S_2 \iff$ istnieją $x, y \in \Sigma$ oraz $S_1', S_2' \in \Sigma^*$ takie, że $S_1 = xS_1'$, $S_2 = yS_2'$ oraz $\mathcal{T} : S_1' \mapsto S_2'$;
- $M\mathcal{T} : S_1 \mapsto S_2 \iff$ istnieją $x \in \Sigma$ oraz $S_1', S_2' \in \Sigma^*$ takie, że $S_1 = xS_1'$, $S_2 = xS_2'$ oraz $\mathcal{T} : S_1' \mapsto S_2'$.

Niech $\|\mathcal{T}\|$ oznacza liczbę wystąpień symboli I, D, R w $\mathcal{T}$ (nie liczymy wystąpiń litery M). *Odległość edycyjną* pomiędzy dwoma słowami $S_1, S_2 \in \Sigma^*$ definiuje się następująco:

$$\delta(S_1, S_2) = min\{\|\mathcal{T}\| \mid \mathcal{T} : S_1 \mapsto S_2\}.$$

Twierdzenie 2.1.4 *Dla dowolnych $S_1, S_2, S_3 \in \Sigma^*$ zachodzi,*
(i) $\delta(S_1, S_2) \geq 0$, *oraz* $\delta(S_1, S_2) = 0 \iff S_1 = S_2$.
(ii) $\delta(S_1, S_2) = \delta(S_2, S_1)$.
(iii) $\delta(S_1, S_2) \leq \delta(S_1, S_3) + \delta(S_3, S_2)$.

Każda funkcja $\delta : \Sigma^* \times \Sigma^* \longrightarrow \mathbb{R}$ spełniająca warunki (i)-(iii) Twierdzenia 2.1.4 nazywa się *odległością*.

Zadanie 2.1.7 Znaleźć algorytm, który oblicza $\delta(S_1, S_2)$, dla dowolnych słów $S_1, S_2 \in \Sigma^*$ w czasie $O(|S_1||S_2|)$.

Zadanie 2.1.8 Określić funkcję podobieństwa $s : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \longrightarrow \mathbb{R}$, tak aby dla dowolnych $S_1, S_2 \in \Sigma^*$ zachdziło

$$\delta(S_1, S_2) = sim(S_1, S_2).$$

Zadanie 2.1.9 Niech s będzie funkcją podobieństwa z Przykładu 2.1.1. Niech $n \geq 1$ będzie dowolną liczbą naturalną i niech Σ^n oznacza zbiór wszystkich słów długości n. Definiujemy funkcję $d : \Sigma^n \times \Sigma^n \longrightarrow \mathbb{R}$ następującym wzorem

$$d(S_1, S_2) = sim(S_1, S_1) + sim(S_2, S_2) - 2sim(S_1, S_2).$$

Czy d jest odległością?

Zadanie 2.1.10 * Niech $s : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \longrightarrow \mathbb{R}$ będzie funkcją podobieństwa spełniającą warunki:
(i) $s(x, x) > 0$, dla $x \in \Sigma$;
(ii) $s(x, y) = s(y, x)$;

(iii) $s(x,x) > s(x,y)$, dla $x \neq y$, $x \in \Sigma$;
(iv) $s(x,-) < 0$, dla $x \in \Sigma$;
(v) $s(z,z) - s(z,x) \geq s(y,z) - s(y,x)$, dla $x,y,z \in \Sigma$.

Niech $sim(S_1, S_2)$ będzie globalnym podobieństwem wyznaczonym przez s. Niech $n \geq 1$. Definiujemy $d : \Sigma^n \times \Sigma^n \longrightarrow \mathbb{R}$ tak jak w Zadaniu 2.1.9. Czy d musi być odległością?