Algoritmi v bioinformatiki
Poravnava več zaporedij

Martin Milanič
martin.milanic@upr.si
UP FAMNIT

9. maj 2025

Literatura za to poglavje:

- **Compeau-Pevzner**, Bioinformatics Algorithms: An Active Learning Approach

  poglavje 5: How Do We Compare Biological Sequences? (str. 280–294)

- **Jones-Pevzner**, An Introduction to Bioinformatics Algorithms,
  poglavje 6.10: Multiple Alignment
  poglavje 10: Clustering and Trees

- **Gusfield**, Algorithms on Strings, Trees, and Sequences,
  poglavje 10: Multiple String Comparison – The Holy Grail

Obravnavali smo že poravnavo dveh zaporedij.


Kako pa je s poravnavo več zaporedij?


Biološka motivacija za poravnavo več zaporedij

Poravnava parov zaporedij je uporabna za primerjavo danega DNA zaporedja (npr. novega gena) z zaporedji, ki so že na voljo v bazah podatkov.

Pri poravnavi več zaporedij pa je cilj pogosto **ravno nasproten**:

Pogosto imajo proteini sorodno biološko funkcijo in/ali podobne tridimenzionalne strukture, vendar pa podobnosti na nivoju zaporedij aminokislin **niso signifikantne** ali pa **niso znane**.

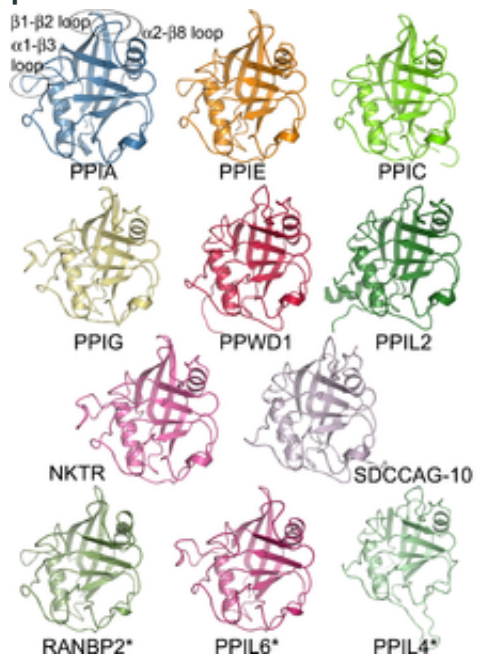
Lahko se zgodi, da primerjava **parov** takih zaporedij ne razkrije nikakršne podobnosti.

Šele s primerjavo zaporedij več aminokislinskih zaporedij hkrati pa dobimo **statistično signifikantne podatke o podobnostih med zaporedji**.

Ti podatki se potem lahko uporabijo tudi za ugotavaljanje evolucijske sorodnosti danih proteinov.

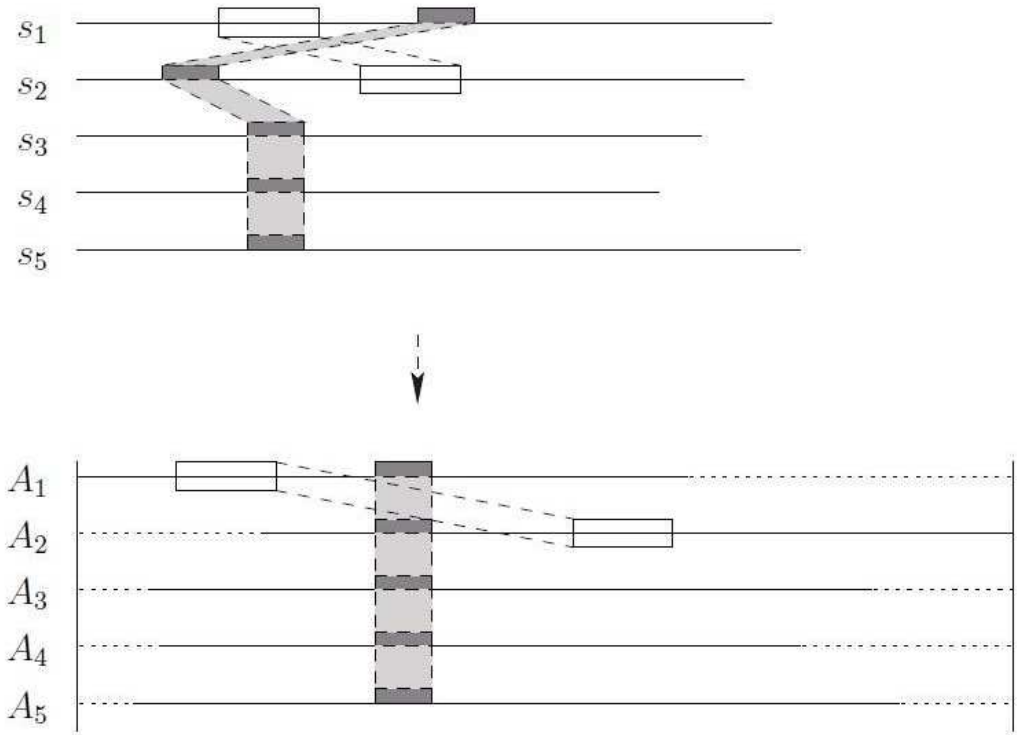
Poravnava več zaporedij:

- je uporabna pri dokazovanju podobnosti med člani družine proteinskih zaporedij,
- če se uporablja kot del iskanja po podatkovnih bazah, ima to prednost, da **poudari ohranjena področja**, kar omogoča odkrivanje sorodnih proteinov,
- je uporabna za napovedovanje sekundarne strukture proteinov.



Poravnava več zaporedij:

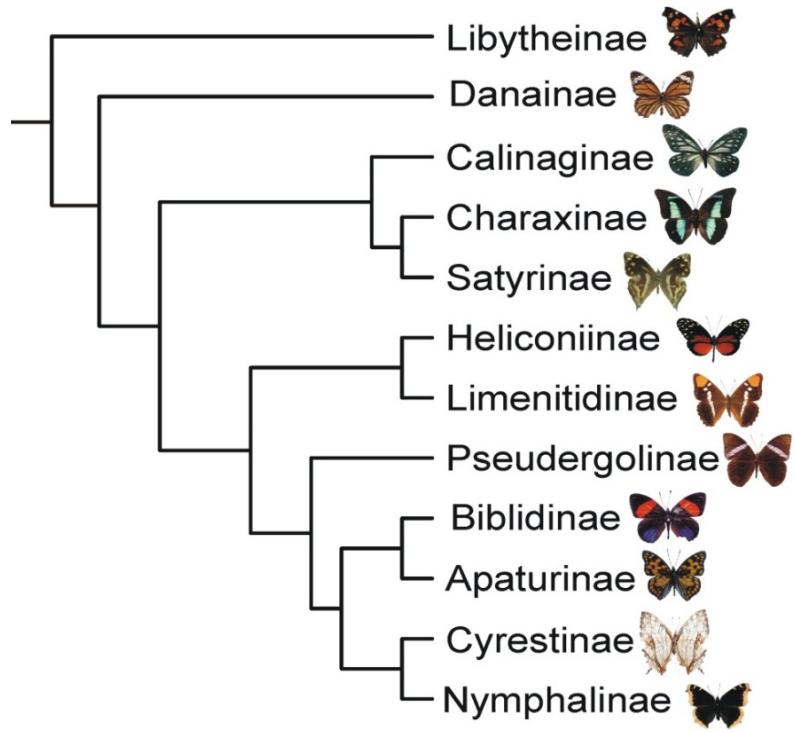
- omogoča **identifikacijo podobnosti**, ki jih s poravnavo parov zaporedij ne moremo zaznati



Majhen vzorec, vsebovan v precej zaporedjih, lahko postane viden šele s poravnavo več zaporedij.

Poravnava več zaporedij:

- lahko služi kot **uvod v sklepanje o vzorcih mutacijskih sprememb** oziroma o **stopnji evolucijskih razmerij** med osebki, kar pogosto vodi k določitvi filogenetskega drevesa.



Poravnava več zaporedij lahko omogoči tudi
razrešitev nejasnosti glede poravnav parov zaporedij.

Zgled:

Glede na Levenshteinovo razdaljo lahko zaporedji $s_1 = \texttt{POLICA}$ in $s_2 = \texttt{PTICA}$ optimalno poravnamo na dva načina:

$$A_1 = \begin{pmatrix} \texttt{P} & \texttt{O} & \texttt{L} & \texttt{I} & \texttt{C} & \texttt{A} \\ \texttt{P} & \texttt{-} & \texttt{T} & \texttt{I} & \texttt{C} & \texttt{A} \end{pmatrix} \quad \text{in} \quad A_2 = \begin{pmatrix} \texttt{P} & \texttt{O} & \texttt{L} & \texttt{I} & \texttt{C} & \texttt{A} \\ \texttt{P} & \texttt{T} & \texttt{-} & \texttt{I} & \texttt{C} & \texttt{A} \end{pmatrix}$$

Šele s tretjim zaporedjem $s_3 = \texttt{POTICA}$ postane očitno, da je prva poravnava najbrž tista prava.

Definicija problema

Definicija

(Globalna) poravnava k zaporedij $s_1, \ldots, s_k$ nad abecedo Σ je k-vrstična matrika A z elementi iz množice $\Sigma \cup \{-\}$, za katero velja:

- če iz i-te vrstice izbrišemo vse presledke, dobimo zaporedje s_i (za vse $1 \leq i \leq k$);
- noben stolpec ne vsebuje samih presledkov.

Vrednost poravnave A:

$$v(A) = \sum_{A_i \text{ stolpec } A} \delta(A_i)\,,$$

kjer je $\delta(x_1, \ldots, x_k)$ dana vrednostna funkcija.

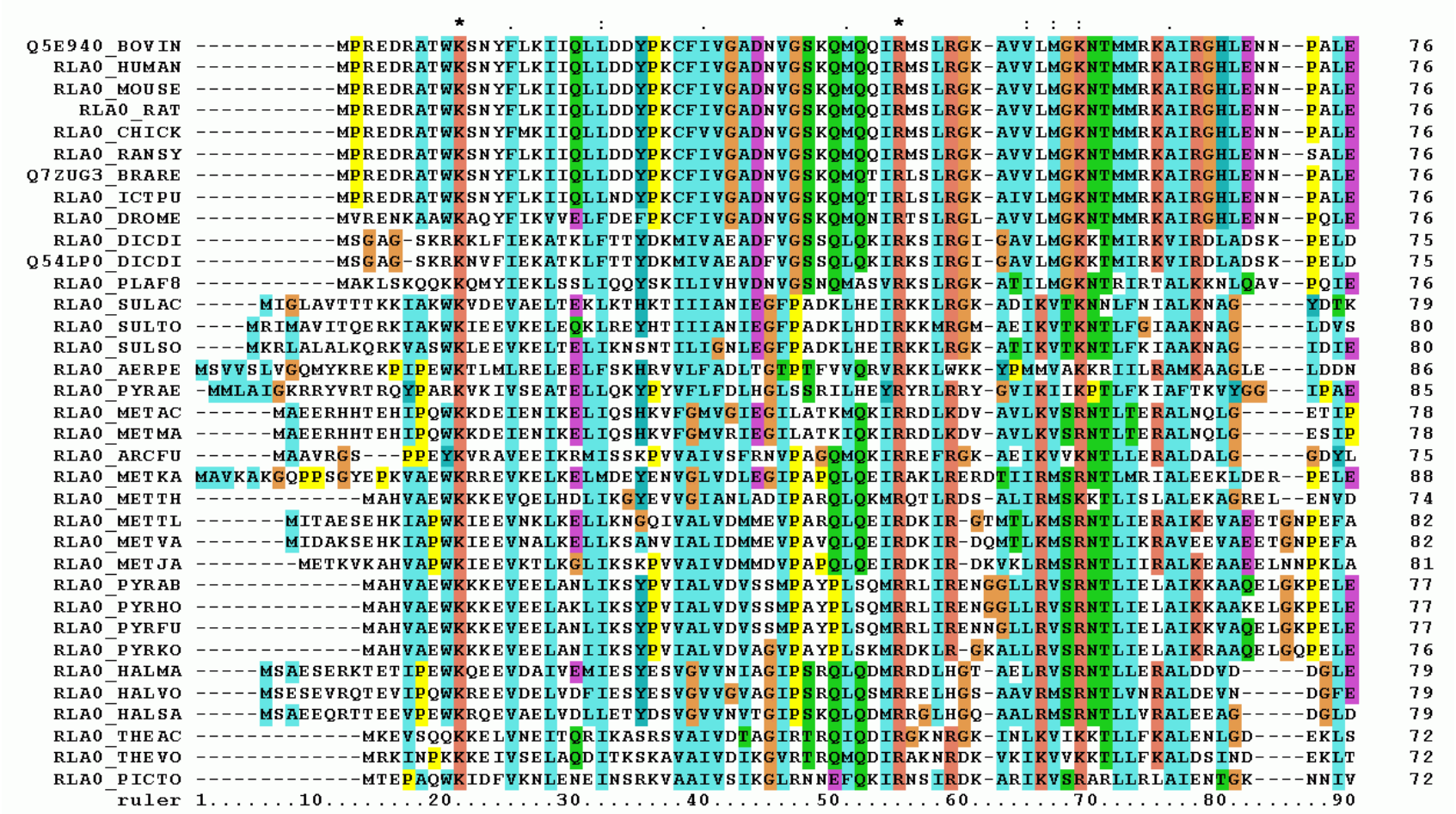
Problem poravnave več zaporedij:

Danih je k zaporedij $s_1, \ldots, s_k$ in vrednostna funkcija δ. Poišči tako poravnavo A^{opt} zaporedij $s_1, \ldots, s_k$, da bo vrednost $v(A^{opt})$ največja možna.

Zgled:

Prvih 90 stolpcev poravnave več proteinskih zaporedij za *acidični ribosomski protein* iz različnih organizmov

(generirano s programom ClustalX):



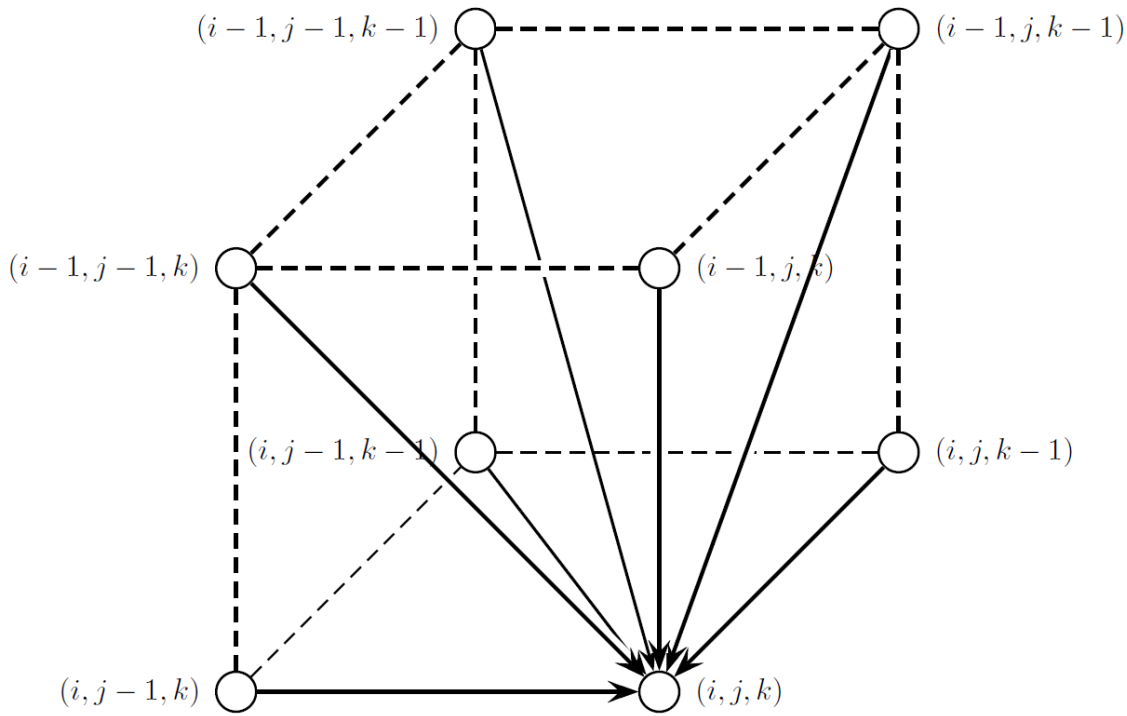
Vir slike: wikipedia

Dinamično programiranje?

Algoritem Needlemana in Wunscha je mogoče posplošiti za primer poravnave k zaporedij $s_1, \ldots, s_k$ dolžin $n_1, \ldots, n_k$:

- konstruiramo k-dimenzionalni graf poravnav;

  Zgled delčka 3-dimenzionalnega grafa poravnav:



- problem postane problem iskanja najtežje poti od točke $(0, 0, \ldots, 0)$ do točke $(n_1, \ldots, n_k)$;
- namesto maksimuma 3 števil moramo v vsaki notranji točki izračunati maksimum $2^k - 1$ števil

Zgled:

$k = 3$, dana imamo zaporedja s_1, s_2, s_3.

Označimo z $v_{i,j,\ell}$ največjo vrednost poravnave zaporedij $s_1[1 \ldots i], s_2[1 \ldots j]$ in $s_3[1 \ldots \ell]$.

Velja naslednja rekurzivna zveza (za $i, j, \ell \geq 1$):

$$v_{i,j,\ell} = \max \begin{cases} v_{i-1,j,\ell} & + & \delta(s_1[i], -, -), \\ v_{i,j-1,\ell} & + & \delta(-, s_2[j], -), \\ v_{i,j,\ell-1} & + & \delta(-, -, s_3[\ell]), \\ v_{i-1,j-1,\ell} & + & \delta(s_1[i], s_2[j], -), \\ v_{i-1,j,\ell-1} & + & \delta(s_1[i], -, s_3[\ell]), \\ v_{i,j-1,\ell-1} & + & \delta(-, s_2[j], s_3[\ell]), \\ v_{i-1,j-1,\ell-1} & + & \delta(s_1[i], s_2[j], s_3[\ell]) \end{cases}$$

Prostorska zahtevnost postane $\mathcal{O}(n^k)$ za k zaporedij dolžine n,

časovna zahtevnost pa $\mathcal{O}(2^k n^k)$.

Razen za zelo majhne k je ta pristop neučinkovit.

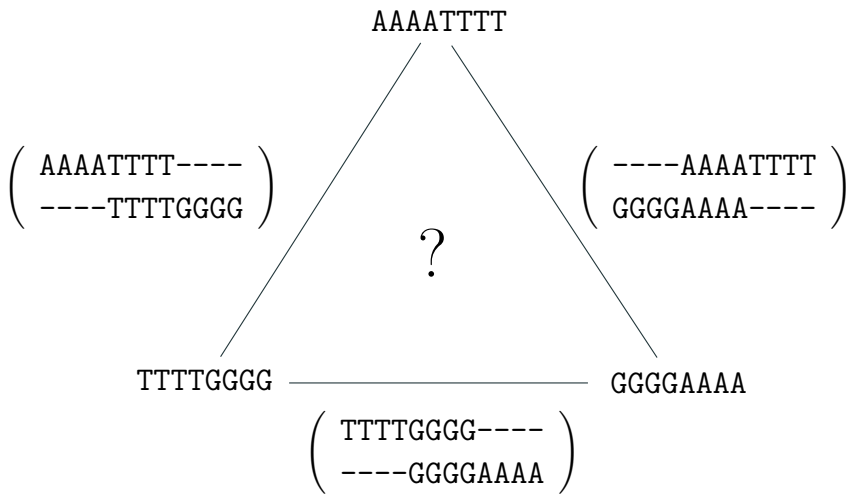
Nekateri drugi pristopi

Ker je problem več poravnav težek in algoritem dinamičnega programiranja prepočasen, se za ta problem uporabljajo tudi razne **hevristike**, ki poiščejo približno rešitev.

(A) Poravnamo vseh $\binom{k}{2}$ parov zaporedij in iz poravnave parov sestavimo poravnavo vseh k zaporedij.

To ni vselej mogoče, saj se lahko zgodi, da poravnave parov niso med seboj kompatibilne.

Zgled:



(B) Vzamemo dve zaporedji, ki sta si najbližji, in ju zamenjamo z zaporedjem, ki ga sestavimo iz njune poravnave.

Rekurzivno poravnamo dobljenih $k - 1$ zaporedij.

To je t.i. **progresivna poravnava**.

- uporablja se v popularnih programih **Clustal W, Clustal X** (Thompson idr. 1994, 1997)

Vrednost poravnave

Spomnimo se, da je vrednost poravnave A določena z

$$v(A) = \sum_{A_i \text{ stolpec } A} \delta(A_i)\,.$$

Vrednostne funkcije $\delta(x_1, \ldots, x_k)$ ni preprosto določiti.

V uporabi je veliko različnih možnosti:

(i) **“Vsota parov”**:

Vrednost poravnave je enaka vsoti vrednosti poravnav vseh $\binom{k}{2}$ parov zaporedij, induciranih s poravnavo k zaporedij.

(ii) **Metoda soglasja** (consensus method):

Vsaki poravnavi priredimo t.i. *zaporedje konsenza*, za katerega lahko izračunamo t.i. *napako poravnave*.

Iščemo tako poravnavo, da bo zaporedje konsenza imelo čim manjšo napako poravnave.

(iii) **Drevesna poravnava**:

Dano je drevo T z vhodnimi zaporedji na listih.

Iščemo taka zaporedja za notranje točke drevesa, da bo vsota vseh vrednosti poravnav parov zaporedij po vseh povezavah drevesa čim manjša.

Vsi ti problemi so NP-težki.

Progresivna poravnava

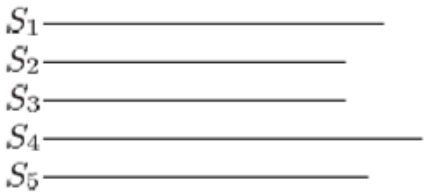
Algoritem Progresivna poravnava (Feng-Doolittle 1987)

Vhod: Zaporedja $s_1, \ldots, s_k$.

Izhod: Poravnava teh k zaporedij ($k \times \ell$ matrika).

1. Optimalno poravnaj vse možne pare zaporedij
2. S pomočjo teh optimalnih poravnav izračunaj **matriko razdalj** med zaporedji.
3. Iz matrike razdalj konstruiraj drevo
   (npr. z metodami **hierarhičnega gručenja**).
4. Progresivno poravnaj zaporedja drugo za drugim z uporabo drevesa iz točke 3.

1. in 2. korak

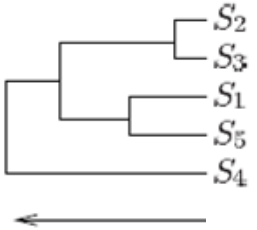


→

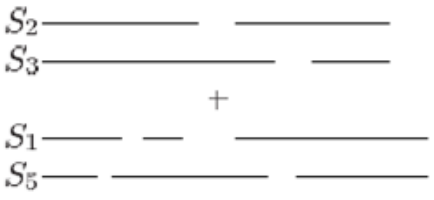
	S_1	S_2	S_3	S_4	S_5
S_1	0				
S_2	17	0			
S_3	17	4	0		
S_4	25	25	25	0	
S_5	10	17	17	25	0

→

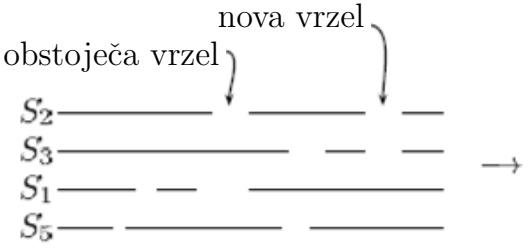
3. korak



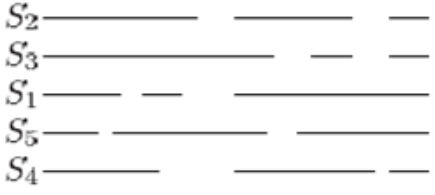
4. korak



→



→



1. korak: optimalna poravnava vseh parov zaporedij

Vsakega od $\binom{k}{2}$ parov zaporedij optimalno poravnamo z algoritmom dinamičnega programiranja Needlemana in Wunscha.

$$
\begin{aligned}
v(0,0) &= 0 \\
v(i,0) &= -i \cdot \sigma \quad \text{za vse } 1 \leq i \leq m \\
v(0,j) &= -j \cdot \sigma \quad \text{za vse } 1 \leq j \leq n \\
v(i,j) &= \max \begin{cases} v(i-1,j-1) + 1\,, & \text{če je } s_i = t_j; \\ v(i-1,j-1) - \mu\,, & \text{če je } s_i \neq t_j; \\ v(i-1,j) - \sigma\,, \\ v(i,j-1) - \sigma\,. \end{cases}
\end{aligned}
$$

2. korak: izračun matrike razdalj

Iz vrednosti optimalnih poravnav izračunamo matriko razdalj D.

Možnosti za to je več:

1. Razdalja med zaporedjema = Levenshteinova razdalja = število zamenjav, izbrisov in vstavljanj

2. Feng in Doolittle predlagata uporabo naslednjih formul:

$$D = -\ln(S_{eff})$$

$$S_{eff}(i,j) = \frac{S(i,j) - S_{\mathrm{rand}}(i,j)}{S_{\mathrm{ident}}(i,j) - S_{\mathrm{rand}}(i,j)}$$

$S(i,j) =$ vrednost optimalne poravnave zaporedij s_i in s_j

$S_{\mathrm{ident}}(i,j) =$ povprečje vrednosti poravnave parov (s_i, s_i) in (s_j, s_j)

$S_{\mathrm{rand}}(i,j) =$ povprečna vrednost poravnave (velikega števila) parov zaporedij (t_i, t_j), generiranih iz naključnih permutacij zaporedij s_i in s_j

3. korak: izračun drevesa iz matrike razdalj

Za ta korak obstaja več metod, na primer:

- **UPGMA** (Unweighted Pair Group Method with Arithmetic Averages)
- **NJ** (Neighbor Joining)
- **Rekonstrukcija dreves iz aditivnih matrik**

Algoritem UPGMA (Sokal-Michener 1958)

Vhod: Matrika razdalj D velikosti $n \times n$

Izhod: Drevo T, ki ponazarja hierarhično gručenje n elementov, predstavljenih z D.

Ustvari n gruč s po enim elementom.

Ustvari graf T, tako da vsaki gruči prirediš po eno (izolirano) točko C.

while obstaja več kot ena gruča

1. Poišči dve najbližji si gruči C_1 in C_2.

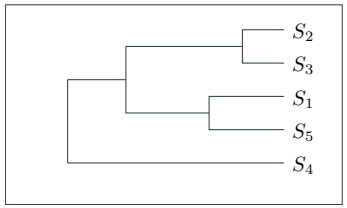
$$d_{avg}(C^*, C) = \frac{1}{|C^*||C|} \sum_{x \in C^*, y \in C} d(x, y)$$

2. Združi C_1 in C_2 v novo gručo s $|C_1| + |C_2|$ elementi.
3. Izračunaj razdaljo od C do vseh ostalih gruč.
4. Grafu T dodaj novo točko C in jo poveži s točkama C_1 in C_2.
5. Odstrani iz D vrstici in stolpca, ki ustrezata C_1 in C_2.
6. Dodaj v D vrstico in stolpec za novo gručo C.

end while
return T

4. korak: progresivna poravnava

Zaporedja poravnamo tako, da se sprehodimo po drevesu od listov do korena.



Poravnava dveh zaporedij:
Optimalno z dinamičnim programiranjem.

Poravnava dveh skupin že poravnanih zaporedij:
Izračunamo vse možne poravnave enega zaporedja iz ene skupine in enega zaporedja iz druge skupine.

Uporabimo najboljšo od poravnav in preostala zaporedja poravnamo vzdolž te najboljše poravnave, pri čemer vrstice ustrezno razširimo s presledki.

Zgled

Zgled: Dana so naslednja zaporedja:

$s_1 = \mathrm{ACGT},$

$s_2 = \mathrm{AGCGT},$

$s_3 = \mathrm{GCCAT},$

$s_4 = \mathrm{GCAT}.$

Poiščimo globalno poravnavo danih zaporedij z algoritmom progresivne poravnave.

Za razdaljo med zaporedji uporabimo Levenshteinovo razdaljo.

1. korak: Vsakega od $\binom{4}{2} = 6$ parov zaporedij optimalno poravnamo z algoritmom dinamičnega programiranja Needlemana in Wunscha

(popravljenega tako, da je "nagrada" za vsako vstavljanje, brisanje in zamenjavo -1, za ujemanje pa 0).

Dobimo naslednje poravnave:

$s_1 = \mathrm{ACGT}$, $s_2 = \mathrm{AGCGT}$:

s_1	A	–	C	G	T
s_2	A	G	C	G	T

Vrednost poravnave je -1.

$s_1 = \mathrm{ACGT}$, $s_3 = \mathrm{GCCAT}$:

s_1	A	–	C	G	T
s_3	G	C	C	A	T

Vrednost poravnave je -3.

$s_1 = \mathrm{ACGT}$, $s_4 = \mathrm{GCAT}$:

s_1	A	C	G	T
s_4	G	C	A	T

Vrednost poravnave je -2.

$s_2 = \mathrm{AGCGT}$, $s_3 = \mathrm{GCCAT}$:

s_2	A	G	C	G	T
s_3	G	C	C	A	T

Vrednost poravnave je -3.

$s_2 = \mathrm{AGCGT}$, $s_4 = \mathrm{GCAT}$:

s_2	A	G	C	G	T
s_4	$-$	G	C	A	T

Vrednost poravnave je -2.

$s_3 = \mathrm{GCCAT}$, $s_4 = \mathrm{GCAT}$:

s_3	G	C	C	A	T
s_4	G	C	$-$	A	T

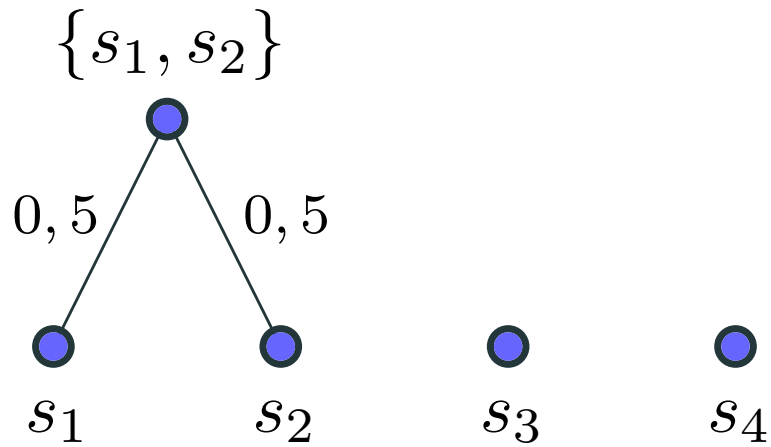
Vrednost poravnave je -1.

2. korak: Izračunamo matriko razdalj:

	s_1	s_2	s_3	s_4
s_1	0	1	3	2
s_2		0	3	2
s_3			0	1
s_4				0

3. korak: Izračunamo drevo z algoritmom UPGMA.

	s_1	s_2	s_3	s_4
s_1	0	1	3	2
s_2		0	3	2
s_3			0	1
s_4				0

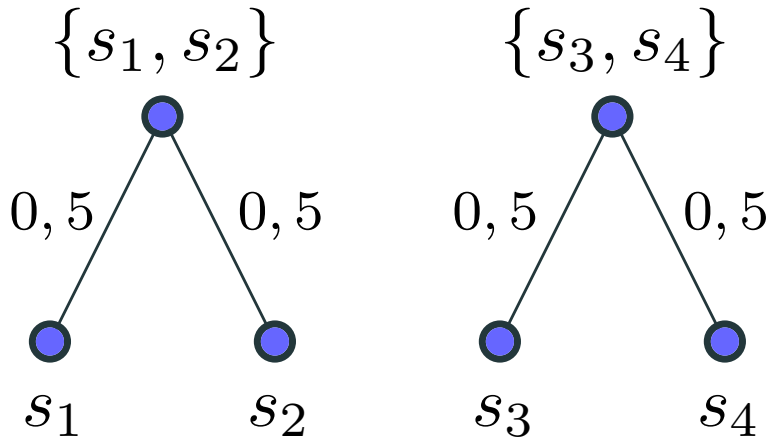


$d(\{s_1, s_2\}, s_3) = 3$

$d(\{s_1, s_2\}, s_4) = 2$

3. korak: Izračunamo drevo z algoritmom UPGMA.

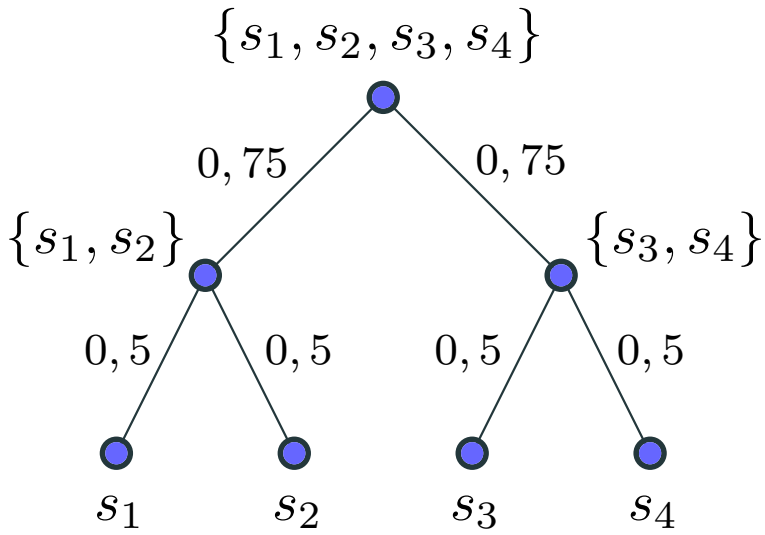
	$\{s_1, s_2\}$	s_3	s_4
$\{s_1, s_2\}$	0	3	2
s_3		0	1
s_4			0



$d(\{s_3, s_4\}, \{s_1, s_2\}) = 2,5$

3. korak: Izračunamo drevo z algoritmom UPGMA.

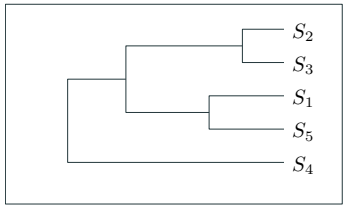
	$\{s_1, s_2\}$	$\{s_3, s_4\}$
$\{s_1, s_2\}$	0	$2,5$
$\{s_3, s_4\}$		0



$d(\{s_3, s_4\}, \{s_1, s_2\}) = 2,5$

4. korak: progresivna poravnava

Zaporedja poravnamo tako, da se sprehodimo po drevesu od listov do korena.



Poravnava dveh zaporedij:

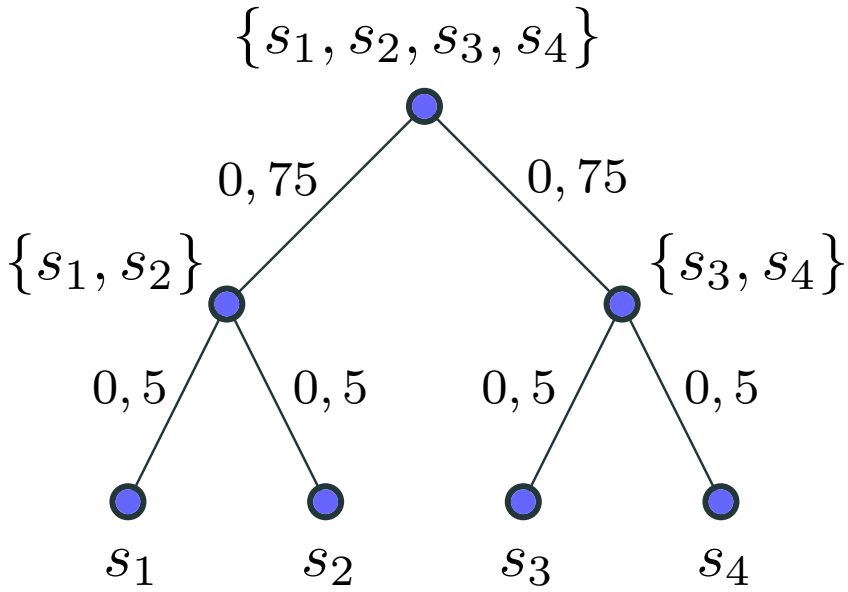
Optimalno z dinamičnim programiranjem.

Poravnava dveh skupin že poravnanih zaporedij:

Izračunamo vse možne poravnave enega zaporedja iz ene skupine in enega zaporedja iz druge skupine.

Uporabimo najboljšo od poravnav in preostala zaporedja poravnamo vzdolž te najboljše poravnave, pri čemer vrstice ustrezno razširimo s presledki.

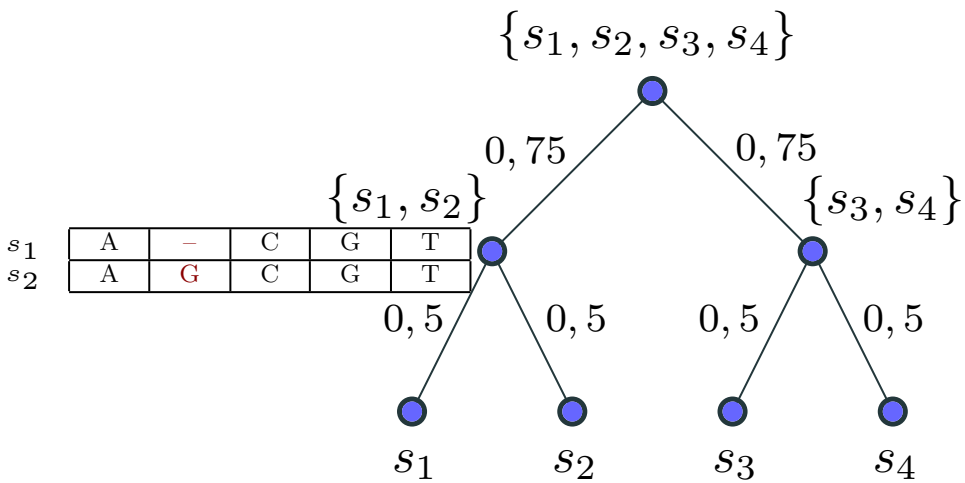
4. korak: Progresivno poravnamo zaporedja drugo za drugim z uporabo drevesa iz točke 3.



4. korak: Progresivno poravnamo zaporedja drugo za drugim z uporabo drevesa iz točke 3.

Zaporedji s_1 in s_2 smo že poravnali:

s_1	A	–	C	G	T
s_2	A	G	C	G	T



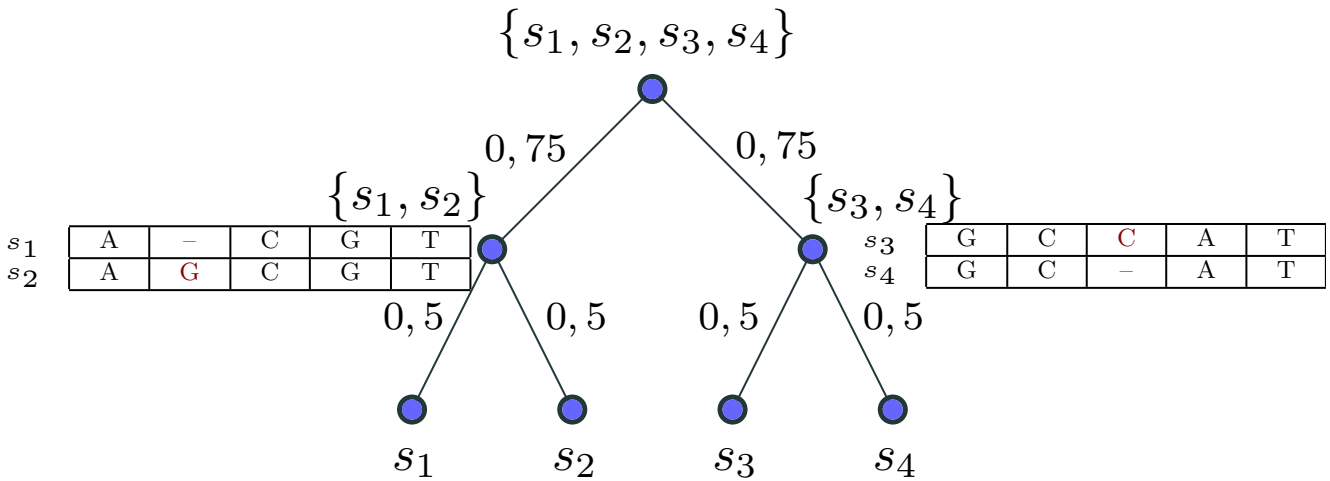
4. korak: Progresivno poravnamo zaporedja drugo za drugim z uporabo drevesa iz točke 3.

Zaporedji s_1 in s_2 smo že poravnali:

s_1	A	–	C	G	T
s_2	A	G	C	G	T

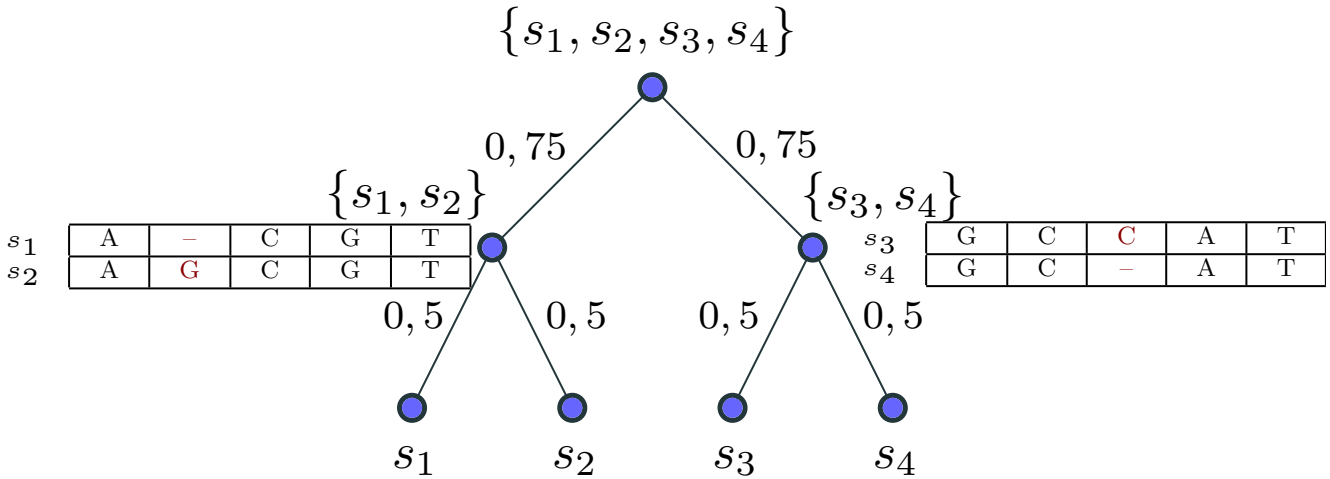
Prav tako tudi zaporedji s_3 in s_4:

s_3	G	C	C	A	T
s_4	G	C	–	A	T



4. korak: Progresivno poravnamo zaporedja drugo za drugim z uporabo drevesa iz točke 3.

Združiti moramo še poravnavi obeh skupin, $\{s_1, s_2\}$ in $\{s_3, s_4\}$.



Uporabimo najboljšo od poravnav enega zaporedja iz ene skupine in enega zaporedja iz druge skupine.

Preostala zaporedja ustrezno razširimo s presledki.

4. korak: Progresivno poravnamo zaporedja drugo za drugim z uporabo drevesa iz točke 3.

Najboljši poravnavi enega zaporedja iz ene skupine in enega zaporedja iz druge skupine sta v tem primeru dve:

poravnava zaporedij s_1**,** s_4

in

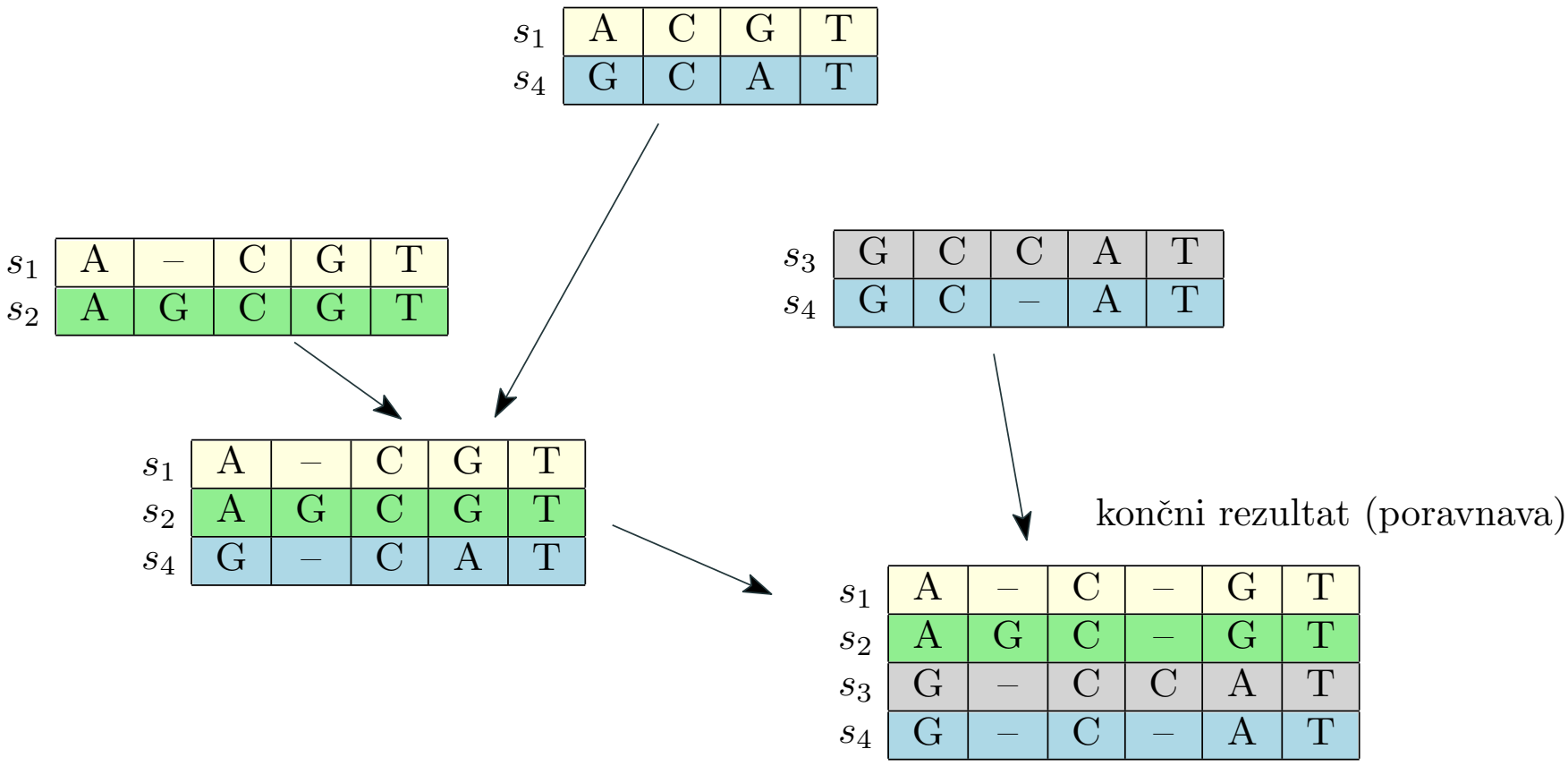
poravnava zaporedij s_2**,** s_4

(obe imata ceno 2).

Izberemo npr. poravnavo s_1 in s_4:

$s_1 = \mathrm{ACGT}$, $s_4 = \mathrm{GCAT}$:

s_1	A	C	G	T
s_4	G	C	A	T


s_1 A C G T
s_4 G C A T
s_1 A – C G T
s_2 A G C G T
s_3 G C C A T
s_4 G C – A T
s_1 A – C G T
s_2 A G C G T
s_4 G – C A T
končni rezultat (poravnava)
s_1 A – C – G T
s_2 A G C – G T
s_3 G – C C A T
s_4 G – C – A T

2-aproksimacijski algoritem

Za vrednostno funkcijo vzemimo **vsoto parov**:

vrednost poravnave je enaka vsoti vrednosti poravnav vseh $\binom{k}{2}$ parov zaporedij, induciranih s poravnavo k zaporedij.

Tokrat predpostavimo, da iščemo poravnavo **čim manjše vrednosti**.

Za ta model bomo predstavili polinomski algoritem za izračun poravnave k zaporedij,

katere vsota parov bo kvečjemu za faktor $2 - 2/k$ večja od optimalne vsote parov.

Predpostavimo še, da imamo dano vrednostno funkcijo δ za poravnavno dveh zaporedij, ki je **metrika**:

(i) $\delta(x, y) \geq 0$ in $\delta(x, y) = 0 \Leftrightarrow x = y$ za vse $x, y \in (\Sigma \cup \{-\})^*$;

(ii) $\delta(x, y) = \delta(y, x)$ za vse $x, y \in (\Sigma \cup \{-\})^*$;

(iii) velja *trikotniška neenakost*:
$\delta(x, z) \leq \delta(x, y) + \delta(y, z)$ za vse $x, y, z \in (\Sigma \cup \{-\})^*$.

Dana so zaporedja $s_1, \ldots, s_k$.

Naj bo $v^*(s_i, s_j)$ optimalna vrednost poravnave zaporedij s_i in s_j.

Za dano poravnavo A naj bo $v_A(s_i, s_j)$ vrednost poravnave zaporedij s_i in s_j, inducirane s poravnavo A.

Poglejmo si naslednji algoritem:

1. Poišči **središčno zaporedje** $s_c \in \{s_1, \ldots, s_k\}$, to je, zaporedje, ki minimizira vrednost $\sum_{j=1}^{k} v^*(s_c, s_j)$.

2. Izračunaj poravnavo A_c, za katero velja

$$v_{A_c}(s_c, s_j) = v^*(s_c, s_j) \text{ za vse } j \neq c\,.$$

Ideja: zaporedoma dodajaj zaporedja s_j, $j \neq c$, k poravnavi že obravnavanih zaporedij, tako da s_j optimalno poravnaš z zaporedjem s_c.

Presledke, ki si jih med tem dodal, po potrebi dodaj na ustrezna mesta že poravnanih zaporedij.

Zgled

Zgled:

Naj δ ustreza Levenshteinovi razdalji,

$$\delta(x, y) = \begin{cases} 1, & \text{če je } x \neq y; \\ 0, & \text{sicer.} \end{cases}$$

Dana so naslednja zaporedja:

$s_1 = \mathrm{ACGT}$,

$s_2 = \mathrm{AGCGT}$,

$s_3 = \mathrm{GCCAT}$,

$s_4 = \mathrm{GCAT}$.

Poiščimo globalno poravnavo danih zaporedij z zgornjim algoritmom.

Vrednosti $v^*(s_i, s_j)$ ustrezajo Levenshteinovi razdalji, ki smo jo že izračunali v zgledu za progresivno poravnavo:

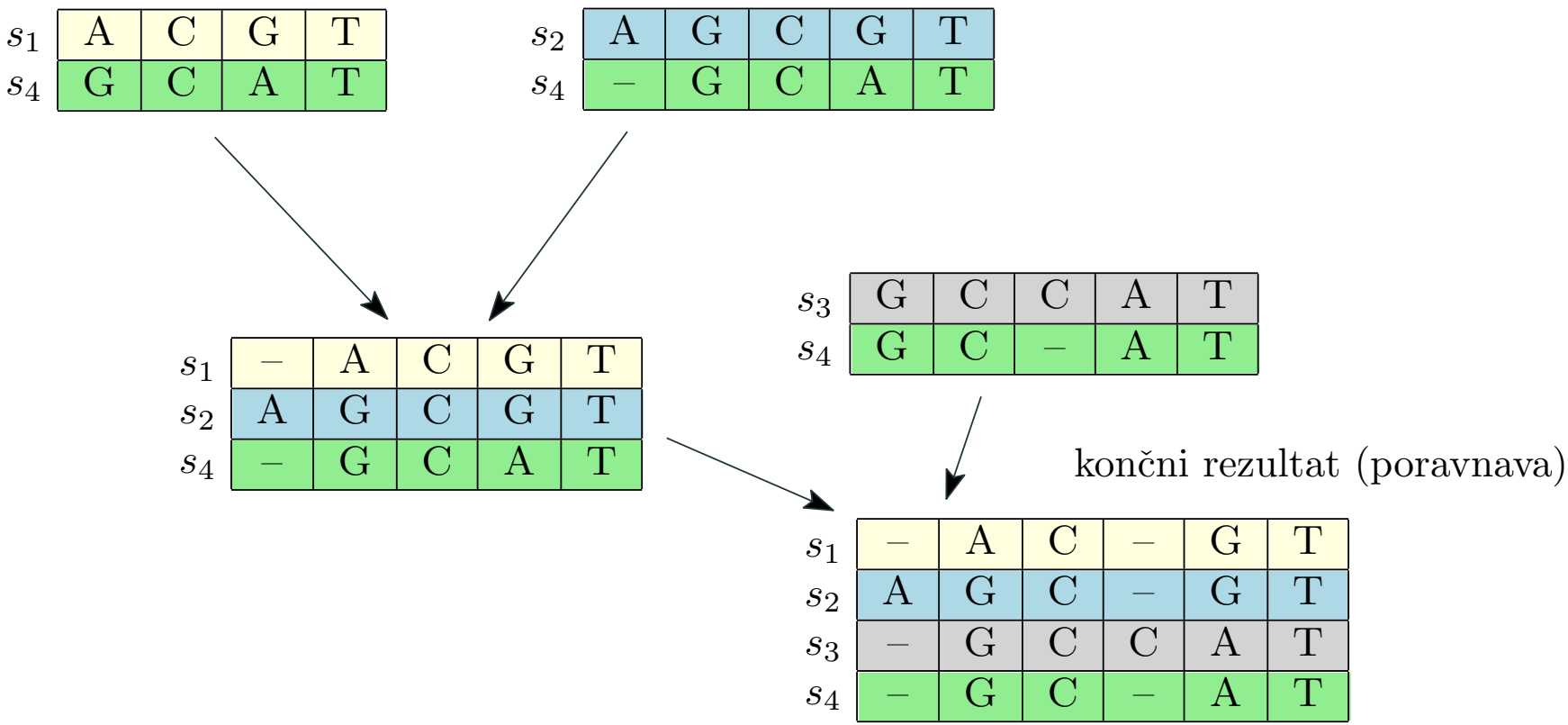
	s_1	s_2	s_3	s_4
s_1	0	1	3	2
s_2	1	0	3	2
s_3	3	3	0	1
s_4	2	2	1	0

Vrednosti $\sum_{j=1}^{k} v^*(s_c, s_j)$ so naslednje:

- $\sum_{j=1}^{k} v^*(s_1, s_j) = \sum_{j=1}^{k} v^*(s_2, s_j) = 6$,
- $\sum_{j=1}^{k} v^*(s_3, s_j) = 7$,
- $\sum_{j=1}^{k} v^*(s_4, s_j) = 5$.

Najmanjša vrednost je dosežena za $c = 4$.

Začnemo (na primer) s poravnavo zaporedij s_1 in s_4 in drugo za drugim dodamo še preostala zaporedja $s_j, j \neq c$.



Časovna zahtevnost: $\mathcal{O}(k^2 n^2)$.

Izrek

Če za vrednost poravnave A vzamemo

$$v(A) := \sum_{i<j} v_A(s_i, s_j)\,,$$

potem za optimalno poravnavo A^* velja

$$v(A_c) \leq (2 - 2/k)\, v(A^*)\,.$$

Utemeljitev aproksimacije:

Pišimo $M = \sum_j v^*(s_c, s_j)$.

Definirajmo $w(A) = \sum_{(i,j)} v_A(s_i, s_j)$

(vsota po vseh urejenih parih).

Seveda velja $w(A^*) = 2v(A^*)$ in $w(A_c) = 2v(A_c)$.

Dovolj je torej dokazati neenakost $w(A_c) \leq (2 - 2/k)w(A^*)$.

$$w(A_c) = \sum_{(i,j)} v_{A_c}(s_i, s_j) \leq \sum_{(i,j)} (v_{A_c}(s_i, s_c) + v_{A_c}(s_c, s_j))$$

(trikotniška neenakost)

$$\leq \sum_{(i,j)} (v^*(s_i, s_c) + v^*(s_c, s_j))$$

(po definiciji A_c).

Za vsak fiksen j se količina $v^*(s_c, s_j)$ pojavi natanko $2(k-1)$-krat.

Sledi:

$$w(A_c) \leq 2(k-1) \sum_j v^*(s_c, s_j) = 2(k-1)M\,.$$

Po drugi strani pa velja

$$w(A^*) = \sum_{(i,j)} v_{A^*}(s_i, s_j) \geq \sum_{(i,j)} v^*(s_i, s_j)$$

$$= \sum_i \sum_j v^*(s_i, s_j) \geq k \cdot \sum_j v^*(s_c, s_j) = kM$$

(neenakost velja po definiciji s_c).

Sledi

$$w(A_c) \leq 2(k-1)M \leq (2 - 2/k) \cdot kM \leq (2 - 2/k) \cdot w(A^*)\,.$$

□