

VI. ALKALMAZÁSOK

Ez a rész öt fejezetből áll.

A *tizenharmadik* fejezet az informatika egyik legdinamikusabban fejlődő területének, a bioinformatikának az alapvető algoritmusait foglalja össze.

A *tizennegyedik* fejezet témája az ember-gép kapcsolatok problémákban gazdag területének az a része, ahol ember és gép együttműködéséből lényegesen nagyobb teljesítmény adódik, mint amire külön akár az ember, akár a gép képes lenne.

A *tizenötödik* fejezet a számítógépi grafika algoritmusait foglalja össze.

A *tizenhatodik* fejezet az informatika és a térképészet határterületének, a térinformatikának alapfogalmaival és legfontosabb algoritmusaiival ismerteti meg az Olvasót.

A *tizenhetedik* fejezet a numerikus matematika alkalmazásával, a gépi számítások eredményeinek megjelenítésével és a számolásigényes tudományos problémák megoldásával együtt foglalkozó terület első magyar nyelvű összefoglalását nyújtja az Olvasóknak. A fejezet szerzői keresztapaként a *tudományos számítások* nevet javasolják a témakörnek.

13. Bioinformatika

Ebben a fejezetben karaktersorozatok, fák, sztochasztikus nyelvtanok biológiai célú elemzésével foglalkozunk.

Az itt bemutatásra kerülő algoritmusok a bioinformatika legfontosabb algoritmusai, számos szoftvercsomag alapját képezik.

13.1. Algoritmusok szekvenciákon

Ebben az alfejezetben olyan dinamikus programozási algoritmusokkal fogunk foglalkozni, amelyek véges karaktersorozatokon – melyeket a biológiában szokásos módon *szekvenciáknak* nevezünk – működnek. A dinamikus programozás minden esetben azon alapszik, hogy a szekvenciák prefixeinek a feldolgozásával jutunk el a teljes szekvenciákon szükséges számolások elvégzéséhez.

13.1.1. Két szekvencia távolsága lineáris résbüntetés mellett

Az információt hordozó DNS molekulák a sejt kettéosztódása előtt kettőződnek, az eredeti molekulával megegyező két molekula jön létre. Biokémiai szabályozás hatására mindkét utódsejtbe egy-egy DNS szál kerül, így az utódsejtek mindegyike tartalmazza a teljes genetikai információt. Azonban a DNS replikálódása nem tökéletes, véletlen mutációk hatására a genetikai információ kissé megváltozhat. Így egy egyed utódai között variánsok, mutánsok állnak elő, amelyekből az évmilliók alatt új fajok alakulnak ki.

Legyen adott két szekvencia. A kérdés az, hogy a két szekvencia mennyire rokon – azaz mennyi idő telt el a szétválásuk óta –, illetve milyen mutációk sorozatával lehet leírni a két szekvencia evolúciós történetét.

Tegyük fel, hogy az egyes mutációk egymástól függetlenek, így egy mutációsorozat valószínűsége az egyes mutációk valószínűségének a szorzata. Az egyes mutációkhoz súlyokat rendelünk, a nagyobb valószínűségű mutációk kisebb, a kisebb valószínűségű mutációk nagyobb súlyt kapnak. Egy jó választás a valószínűség logaritmusának a mínusz egyszerűsége. Ekkor egy mutációsorozat súlya az egyes mutációk súlyainak az összege. Feltételezzük, hogy egy mutációs változás és a megfordítottja ugyanakkora valószínűséggel fordul elő. Így a két szekvencia közös ősből való leszármazása helyett azt kell vizsgálni, hogyan alakulhatott ki az egyik szekvencia a másikkból. A *minimális törzsfejlődés* elvét feltételezve, azt a minimális súlyú mutációsorozatot keressük, amely az egyik szekvenciát a másikba

alakítja. Fontos kérdés, hogy hogyan lehet egy minimális súlyú mutációsorozatot végig egybe (lehet, hogy több minimális értékű sorozat van) gyorsan megkeresni. A naiv algoritmus megkeresi az összes lehetséges mutációsorozatot, és kiválasztja közülük a minimális súlyút. Ez nyilvánvalóan nagyon lassú, mert a lehetséges sorozatok száma exponenciálisan nő a szekvenciák hosszával.

Legyen Σ szimbólumok véges halmaza, Σ^* jelölje a Σ feletti véges hosszú szavak halmazát. Egy A szó első n betűjéből álló szót A_n jelöli, az n -edik karaktert pedig a_n . Egy szón a következő transzformációk hajthatók végre:

- Egy a szimbólum beszúrása egy szóba egy adott i pozíció előtt. Ezt a transzformációt $a \leftarrow^i$ jelöli.
- Egy a szimbólum törlése egy szóból egy adott i pozíciónál. Ezt a transzformációt $\leftarrow^i a$ jelöli.
- Egy a szimbólum cseréje egy b szimbólumra egy adott i pozícióban. Ezt a transzformációt $b \leftarrow^i a$ jelöli.

A transzformációk **kompozícióján** azok egymás utáni végrehajtását értjük, és a \circ szimbólummal fogjuk jelölni. A fenti három transzformációnak, és ezek tetszőleges véges kompozícióinak a halmazát τ -val jelöljük. Azt, hogy egy $T \in \tau$ transzformációsorozat az A szekvenciát B -vé transzformálja, $T(A) = B$ -vel jelöljük.

Legyen $w : \tau \rightarrow \mathbb{R}^+ \cup \{0\}$ egy olyan **súlyfüggvény**, hogy ha bármely T_1, T_2 és S transzformációsorozatra

$$T_1 \circ T_2 = S \quad (13.1)$$

teljesül, akkor

$$w(T_1) + w(T_2) = w(S) , \quad (13.2)$$

valamint $w(a \leftarrow^i b)$ független i -től. Két szekvencia, A és B transzformációs távolságán az A -t B -be vivő minimális súlyú transzformációk súlyát értjük, azaz

$$\delta(A, B) = \min\{w(T) | T(A) = B\} . \quad (13.3)$$

Ha w -ről feltesszük, hogy

$$w(a \leftarrow b) = w(b \leftarrow a) , \quad (13.4)$$

$$w(a \leftarrow a) = 0 , \quad (13.5)$$

$$w(b \leftarrow a) + w(c \leftarrow b) \geq w(c \leftarrow a) \quad (13.6)$$

bármely $a, b, c \in \Sigma \cup \{-\}$ -re, akkor a $\delta(\cdot, \cdot)$ transzformációs távolság metrika Σ^* -on, így jogos az elnevezés.

$w(\cdot)$ metrikus tulajdonsága miatt elég olyan transzformációsorozatokkal foglalkozni, amelyek során minden pozíció legfeljebb egyszer transzformálódik. A transzformáció sorozatokat a szekvenciák illesztésével ábrázoljuk. Konvenció alapján a felülre írt szekvencia az ősi szekvencia, az alulra írt a leszármazott. Például, az alábbi illesztés azt mutatja, hogy a harmadik és az ötödik pozícióban egy-egy csere történt, az első pozícióban egy beszúrás, a nyolcadikban pedig egy törlés:

```

- A U C G U A C A G
U A G C A U A - A G

```

Az egyes pozíciókban az egymás alá írt szimbólumokat **illesztett párnak** hívjuk. A transzformációsorozat súlya az egyes pozíciókban történt mutációk súlyainak az összege. Látható, hogy minden mutációsorozathoz egyértelműen megadható egy illesztés, amely azt ábrázolja, és egy illesztésből a mutációk sorrendjétől eltekintve egyértelműen megadhatók azok a mutációk, amelyeket az illesztés ábrázol. Mivel az összeadás kommutatív, ezért a teljes súly független a mutációk sorrendjétől.

Most megmutatjuk, hogy a lehetséges illesztések száma is exponenciálisan nő a szekvenciák hosszával. A lehetséges illesztések halmazának egy részhalmazát adják azok az illesztések, amelyekben beszúrás és törlés nem szerepel egymás melletti illesztett oszlopban, azaz nem található az illesztésben az alábbi mintázatok egyike sem:

$$\begin{array}{c} \# \quad - \quad - \quad \# \\ - \quad \# \quad \# \quad - \end{array} ,$$

ahol # tetszőleges karakter Σ -ból. Az ilyen illesztések száma $\binom{|A|+|B|}{|A|}$, mivel bijekció van az ilyen illesztések halmaza és az olyan C szavak között, amelyek pontosan a két szekvencia betűiből állnak, és mind A , mind B összes betűje növekvő sorrendben helyezkedik el C -ben. Ha például $|A| = |B| = n$, akkor $\binom{|A|+|B|}{|A|} = \Theta(2^{2n} / \sqrt{n})$.

Optimális illesztésnek nevezzük azt az illesztést, amelyhez a hozzárendelt mutációsorozat súlya minimális. Jelöljük $\alpha^*(A_i, B_j)$ -vel A_i és B_j optimális illesztéseinek a halmazát, $w(\alpha^*(A_i, B_j))$ -vel jelöljük az ezen illesztésekhez tartozó mutációsorozatok súlyát.

A gyors algoritmus ötlete az, hogy ha ismerjük $w(\alpha^*(A_{i-1}, B_j))$ -t, $w(\alpha^*(A_i, B_{j-1}))$ -et, valamint $w(\alpha^*(A_{i-1}, B_{j-1}))$ -et, akkor ebből konstans idő alatt kiszámítható $w(\alpha^*(A_i, B_j))$. Tekintsük ugyanis A_i és B_j egy optimális illesztésének az utolsó illesztett párját. Ezt elhagyva vagy A_{i-1} és B_j vagy A_i és B_{j-1} vagy A_{i-1} és B_{j-1} egy optimális illesztését kapjuk, rendre attól függően, hogy az utolsó pár egy törlést, beszúrást vagy cserét ábrázol. Így

$$\begin{aligned} w(\alpha^*(A_i, B_j)) = & \min\{w(\alpha^*(A_{i-1}, B_j)) + w(- \leftarrow a_i) ; \\ & w(\alpha^*(A_i, B_{j-1})) + w(b_i \leftarrow -) ; \\ & w(\alpha^*(A_{i-1}, B_{j-1})) + w(b_i \leftarrow a_i)\} . \end{aligned} \quad (13.7)$$

Az optimális illesztéshez tartozó súlyokat egy úgynevezett dinamikus programozási mátrix, D segítségével lehet megadni. A D mátrix $d_{i,j}$ eleme $w(\alpha^*(A_i, B_j))$ -t tartalmazza. Ez egy n és egy m hosszúságú szekvencia összehasonlítása esetén egy $(n+1)(m+1)$ -es táblázat, a sorok és oszlopok indexelése 0-tól n -ig, illetve m -ig megy. A kezdeti feltételek a nulladik sorra és oszlopra:

$$d_{0,0} = 0 , \quad (13.8)$$

$$d_{i,0} = \sum_{k=1}^i w(- \leftarrow a_k) , \quad (13.9)$$

$$d_{0,j} = \sum_{l=1}^j w(b_l \leftarrow -) . \quad (13.10)$$

A táblázat belsejének a kitöltése a (13.7) képlet alapján történik.

A táblázat kitöltésének az ideje $\Theta(nm)$. A táblázat segítségével megkereshetjük az

összes optimális illesztést. Egy optimális illesztés megkereséséhez a jobb alsó sarokból kiindulva mindig a minimális értéket adó előző pozíciót választva – több lehetőség is adódhat – visszafelé haladunk a bal felső sarokig, minden egyes lépés az optimális illesztésnek egy illesztett párját adja meg. A $d_{i,j}$ pozícióból fölfelé lépés az adott pozícióban való törlést, a balra lépés az adott pozícióban való beszúrást jelent, az átlón való haladás pedig vagy az adott pozícióban való szubsztitúciót jelzi, vagy azt mutatja, hogy az adott pozícióban nem történt mutáció, attól függően, hogy a_i nem egyezik meg b_j -vel, vagy megegyezik. Az összes optimális illesztések száma exponenciálisan nőhet a szekvenciák hosszával, viszont azok polinomiális időben ábrázolhatók. Ehhez egy olyan irányított gráfot kell készíteni, amelynek csúcsai a dinamikus programozási táblázat elemei, és egy él megy egy v_1 csúcsból v_2 -be, ha v_1 minimális értéket ad v_2 -nek. Ezen a gráfon minden irányított út $d_{n,m}$ -ből $d_{0,0}$ -ba egy optimális illesztést határoz meg.

13.1.2. Dinamikus programozás tetszőleges résbüntetés mellett

Mivel egy beszúrást ugyanakkora súllyal értékelünk, mint egy törlést, ezeket közös névvel **réseknek** szokás nevezni, a hozzá tartozó súlyt pedig **résbüntetésnek**. Általában egy súlyt szoktak használni minden karakter beszúrására és törlésére. Az alapalgoritmus úgy tekinti a szekvenciák változását, hogy egy hosszú rés kialakulását rövidebb rések egymásutánjának képzele. Ez biológiaiilag helytelen, mert tudjuk, hogy egy hosszabb részszekvencia beszúrása vagy törlése egyetlen mutációval is megtörténhet. Így az alapalgoritmus a hosszú réseket túlzott mértékben bünteti. Ez a felismerés motiválta a különböző résbüntető függvények bevezetését a biológiai szekvenciák elemzésében, melyekben egy k hosszúságú részt g_k értékkel büntetünk. Például az

```
- - A U C G A C G U A C A G
U A G U C - - - A U A G A G
```

illesztés súlya $g_2 + w(G \leftarrow A) + g_3 + w(A \leftarrow G) + w(G \leftarrow C)$.

Továbbra is azt a minimális súlyú mutációsorozatot keressük, amely az egyik szekvenciát a másikba alakítja. A dinamikus programozási algoritmus abban különbözik az előző algoritmustól, hogy az illesztés végén állhat egy hosszú rés, így $w(\alpha^*(A_i, B_j))$ kiszámításához nem csak $w(\alpha^*(A_{i-1}, B_j))$, $w(\alpha^*(A_i, B_{j-1}))$ és $w(\alpha^*(A_{i-1}, B_{j-1}))$ ismeretére van szükség, hanem $w(\alpha^*(A_{i-1}, B_{j-1}))$ -en kívül minden $w(\alpha^*(A_k, B_j))$, $0 \leq k < i$ -re és minden $w(\alpha^*(A_i, B_l))$, $0 \leq l < j$ -re. Az előbbiekhöz hasonlóan belátható, hogy

$$w(\alpha^*(A_i, B_j)) = \min\{w(\alpha^*(A_{i-1}, B_{j-1})) + w(b_j \leftarrow a_i), \\ \min_{0 \leq k < i}\{w(\alpha^*(A_k, B_j)) + g_{i-k}\}, \\ \min_{0 \leq l < j}\{w(\alpha^*(A_i, B_l)) + g_{j-l}\}\} . \quad (13.11)$$

Továbbra is egy $(n+1)(m+1)$ -es dinamikus programozási táblázat kitöltésével lehet kiszámítani egy n és egy m hosszúságú szekvencia optimális illesztését. A kezdeti feltételek a nulladik sorra és oszlopra:

$$d_{0,0} = 0 , \quad (13.12)$$

$$d_{i,0} = g_i , \quad (13.13)$$

$$d_{0,j} = g_j . \quad (13.14)$$

A táblázat $d_{i,j}$ elemének a kiszámításához $\Theta(i + j)$ időre van szükség, így a táblázat kitöltése – és így egy optimális illesztés súlya – $\Theta(nm(n + m))$ idő alatt van meg. Egy optimális illesztést, hasonlóan az előző algoritmushoz, a $d_{n,m}$ -ből a minimális értékeket adó pozíciókon át a $d_{0,0}$ -ig vezető út definiál.

Ennek az algoritmusnak a futási ideje tehát a szekvenciák hosszának a köbével arányos, ha kettő, nagyjából egyforma hosszú szekvenciát hasonlítunk össze. Ha a részbűtető függvényre bizonyos megkötéseket teszünk, akkor lehetőség van gyorsabb algoritmusok konstruálására. A következő két pontban ilyen algoritmusokra mutatunk példát.

13.1.3. Gotoh algoritmus affín részbűtetéssel

Egy részbűtető függvényt *affín függvénynek* hívunk, ha

$$g_k = ku + v, \quad u \geq 0, \quad v \geq 0. \quad (13.15)$$

Ilyen részbűtető függvényt alkalmaz a Gotoh-algoritmus, amelynek a futási ideje $\Theta(nm)$. Emlékeztetőül, az előbbi gyors algoritmusban

$$d_{i,j} = \min\{d_{i-1,j-1} + w(b_j \leftarrow a_i); p_{i,j}; q_{i,j}\}, \quad (13.16)$$

ahol

$$p_{i,j} = \min_{1 \leq k < i} \{d_{i-k,j} + g_k\}, \quad (13.17)$$

$$q_{i,j} = \min_{1 \leq l < j} \{d_{i,j-l} + g_l\}. \quad (13.18)$$

Az algoritmus a következő átindexelésen alapul:

$$\begin{aligned} p_{i,j} &= \min\{d_{i-1,j} + g_1, \min_{2 \leq k < i} \{d_{i-k,j} + g_k\}\} \\ &= \min\{d_{i-1,j} + g_1, \min_{1 \leq k < i-1} \{d_{i-1-k,j} + g_{k+1}\}\} \\ &= \min\{d_{i-1,j} + g_1, \min_{1 \leq k < i-1} \{d_{i-1-k,j} + g_k\} + u\} \\ &= \min\{d_{i-1,j} + g_1, p_{i-1,j} + u\}. \end{aligned} \quad (13.19)$$

És hasonlóan

$$q_{i,j} = \min\{d_{i,j-1} + g_1, q_{i,j-1} + u\}. \quad (13.20)$$

Így $p_{i,j}$ és $q_{i,j}$ konstans idő alatt kiszámítható, ezekből pedig $d_{i,j}$ is konstans idő alatt kiszámítható, a táblázat minden elemére. Így az algoritmus futási ideje $\Theta(nm)$ marad, és maga az algoritmus csak egy konstans faktorral lesz lassabb, mint az alapalgoritmus, amely nem enged meg hosszú réseket egyetlen mutációs lépésben.

13.1.4. Konkáv részbűtetés

Az affín részbűtető függvény helyességére nincs semmilyen biológiai magyarázat, elterjedt használatát (például Clustal-W) a hozzá tartozó gyors algoritmusnak köszönheti. Meg lehet szabni azonban egy sokkal realisabb feltételt a részbűtető függvényre, amelyre Gotoh algoritmusánál egy kicsit lassabb, de az általános részbűtető köbös idejű algoritmusnál mégis

lényegesen gyorsabb algoritmus létezik.

Egy résbüntető függvényt **konkávnak** nevezünk, ha bármely i -re $g_{i+1} - g_i \leq g_i - g_{i-1}$. Magyarán: a rések növekedését egyre kevésbé büntetjük. Nyilván, ha a függvény egy adott csúcstól csökkenni kezd, akkor elég nagy résekre negatív súlyokat kapunk. Ezt elkerülendő, a függvényről még fel szokták tenni, hogy szigorúan monoton növekszik, bár algoritmikai szempontból ennek semmi jelentősége nincs. Empirikus adatok alapján, ha két szekvencia d PAM egység óta evolválódott, akkor egy q hosszúságú beszúrás vagy törlés súlya

$$35.03 - 6.88 \log d + 17.02 \log q, \quad (13.21)$$

ami szintén konkáv függvény. (Egy PAM egység az az időtartam, amely alatt a szekvencia 1%-a változik meg.)

Konkáv résbüntető függvényekre létezik $O(nm(\lg n + \lg m))$ idejű algoritmus. Ez az algoritmus az úgynevezett előretékintő algoritmusok családjába tartozik. Az ELŐRETEKINTŐ algoritmus tetszőleges résbüntető függvény esetén a következő módon számítja ki a dinamikus programozási táblázat i -edik sorát.

ELŐRETEKINTŐ(i, m, q, d, g, w, a, b)

```

1  for  $j \leftarrow 1$  to  $m$ 
2      do  $q_1[i, j] \leftarrow d[i, 0] + g[j]$ 
3          $b[i, j] \leftarrow 0$ 
4  for  $j \leftarrow 1$  to  $m$ 
5      do  $q[i, j] \leftarrow q_1[i, j]$ 
6          $d[i, j] \leftarrow \min[q[i, j], p[i, j], d[i-1, j-1] + w(b_j \leftarrow a_i)]$ 
7         ▷ Ennél a lépésnél feltesszük, hogy  $p[i, j]$ -t és  $d[i-1, j-1]$ -et már kiszámoltuk.
8         for  $j \leftarrow j_1$  to  $m$                                      ▷Belső ciklus.
9             do if  $q_1[i, j_1] > d[i, j] + g[j_1 - j]$ 
10                then  $q_1[i, j_1] \leftarrow d[i, j] + g[j_1 - j]$ 
11                    $b[i, j_1] \leftarrow j$ 

```

A pszeudokódban $g[]$ a résbüntető függvény, b pedig egy pointer, amelynek a jelentése a későbbiekben derül ki. A hatodik sorban feltesszük, hogy $p[i, j]$ -t és $d[i-1, j-1]$ -et már kiszámoltuk. Könnyen belátható, hogy az ELŐRETEKINTŐ algoritmus pontosan ugyanazokat az összehasonlításokat végzi, mint az eredeti dinamikus programozási algoritmus, csak más sorrendben. Míg az eredeti algoritmus a sor j -edik pozíciójába érkezve megnézi, hogy mi lehet az optimális $q_{i,j}$, addig az ELŐRETEKINTŐ algoritmus a j -edik pozícióba érve $q_{i,j}$ -t már meghatározta, viszont megnézi, hogy ennek a pozíciónak az értékéhez a konkáv résbüntető értékeket hozzáadva, melyik q_{i,j_1} -re lehet optimális (belső ciklus). Az aktuális legjobb jelölt $q_1[i, j_1]$ -ben tárolódik, és mire a j_1 -edik cellához érünk, minden szükséges összehasonlítást elvégeztünk. Tehát az ELŐRETEKINTŐ algoritmus nem gyorsabb az eredeti algoritmusnál, de a koncepció segít a gyorsításban.

A gyorsítás alapja a következő észrevétel.

13.1. lemma. Legyen j az aktuális cella. Ha

$$d_{i,j} + g_{j_1-j} \geq q_1[i, j_1], \quad (13.22)$$

akkor minden $j_2 > j_1$ -re

$$d_{i,j} + g_{j_2-j} \geq q_1[i, j_2]. \quad (13.23)$$

Bizonyítás. A feltételből következik, hogy van olyan $k < j < j_1 < j_2$, melyre

$$d_{i,j} + g_{j_1-j} \geq d_{i,k} + g_{j_1-k} . \quad (13.24)$$

Adjunk az egyenlőtlenség mindkét oldalához $(g_{j_2-k} - g_{j_1-k})$ -t:

$$d_{i,j} + g_{j_1-j} + g_{j_2-k} - g_{j_1-k} \geq d_{i,k} + g_{j_2-k} . \quad (13.25)$$

A konkáv részbüntető függvény tulajdonságából

$$g_{j_2-j} - g_{j_1-j} \geq g_{j_2-k} - g_{j_1-k} , \quad (13.26)$$

és ebből átrendezve és a (13.25) egyenletet felhasználva :

$$d_{i,j} + g_{j_2-j} \geq d_{i,j} + g_{j_1-j} + g_{j_2-k} - g_{j_1-k} \geq d_{i,k} + g_{j_2-k} , \quad (13.27)$$

amiből már közvetlenül adódik az állítás. ■

Tehát az algoritmus ötlete az, hogy binárisan keressük meg azt a pozíciót, ahonnan felesleges összehasonlításokat végezni, mert az adott pozíció biztosan nem adhat optimális $q_{i,j}$ értéket azon túl. Ez azonban még mindig nem elég az algoritmus kívánt gyorsításához, legrosszabb esetben ugyanis még így is $\Theta(m)$ értéket kellene felülírni minden egyes pozíciónál. Az előző észrevétel egy következménye azonban már elvezet a kívánt gyorsításhoz.

13.2. következmény. Mielőtt a j -edik cella jelölteket küldene előre az algoritmus belső ciklusában, a j -edik pozíciótól a pozíciók blokkokra oszthatók, minden blokknak a b pointer ugyanakkora, és ezek a pointer értékek blokkonként csökkennek balról jobbra haladva.

Az algoritmus pszeudokódja a következő.

ELŐRETEKINTŐ-BINÁRISKERESŐ(i, m, q, d, g, w, a, b)

```

1  ( $pn[i], p[i, 0], b[i, 0]$ )  $\leftarrow$  ( $0, m, 0$ )
2  for  $j \leftarrow 1$  to  $m$ 
3      do  $q[i, j] \leftarrow q[i, b[i, pn[i]]] + g[j - b[i, pn[i]]]$ 
4           $d[i, j] \leftarrow \min[q[i, j], p[i, j], d[i - 1, j - 1] + w(b_j \leftarrow a_i)]$ 
5       $\triangleright$  Ennél a lépésnél feltesszük, hogy  $p[i, j]$ -t és  $d[i - 1, j - 1]$ -et már kiszámoltuk.
6      if  $p[i, pn[i]] = j$ 
7          then  $pn[i] \leftarrow pn[i] - 1$ 
8      if  $j + 1 < m$  és  $d[i, b[i, 0]] + g[m - b[i, 0]] > d[i, j] + g[m - j]$ 
9          then  $pn[i] \leftarrow 0$ 
10          $b[i, 0] \leftarrow j$ 
11     else if  $j + 1 < m$ 
12         then  $Y \leftarrow \max_{0 \leq X \leq pn[i]} \{X|d[i, b[i, X]] + g[p[i, X] - b[i, X]]$ 
13              $\leq p[i, j] + g[p[i, X] - j]\}$ 
14         if  $d[i, b[i, Y]] + g[p[i, Y] - b[i, Y]] = p[i, j] + g[p[i, X] - j]$ 
15             then ( $pn[i], b[i, Y]$ )  $\leftarrow$  ( $Y, j$ )
16             else  $E \leftarrow p[i, Y]$ 
17         if  $Y < pn[i]$ 
18             then  $B \leftarrow p[i, Y + 1] - 1$ 
19             else  $B \leftarrow j + 1$ 
20          $pn[i] \leftarrow pn[i] + 1$ 
21          $b[i, pn[i]] \leftarrow j$ 
22          $p[i, pn[i]] \leftarrow \max_{B \leq X \leq E} \{X|d[i, j] + g[X - j] \leq d[i, b[i, Y]] + g[X - b[i, Y]]\}$ 

```

Az algoritmus a következőképpen működik (lásd ELŐRETEKINTŐ-BINÁRISKERESŐ): minden sorra fenn kell tartani egy változót, amely az aktuális blokkok számát tárolja. Minden blokkra csak egy pointer-t tartunk fenn, kell készíteni a blokkok végeinek egy csökkenő listáját, valamint ezzel párhuzamosan a blokkok közös pointereinek egy listáját. A lista hossza értelemszerűen a blokkok számával egyezik meg. Ezután bináris kereséssel megkeressük azt az utolsó pozíciót, amelyre az aktuális pozíció még optimális jelöltet ad. Ezt úgy tesszük meg, hogy először kiválasztjuk a blokkot, majd a blokkon belül keressük meg a pozíciót. A blokkok száma eggyel több, mint ahány blokk maradt az új blokkvég után. A bináris keresés módjából adódóan ezt már ismerjük. Végül a listát felülírjuk. Elég egyetlen értéket felülírni mindkét listában, a pozíciók listájának új, utolsó értéke a bináris kereséssel megkeresett pozíció, a pointerek listájának az új utolsó értéke pedig az új blokk pointerértéke, azaz az aktuális pozíció. Így minden egyes pozícióban konstans mennyiségeket írunk felül. A leginkább időigényes rész a bináris keresés, ez $O(\lg m)$ idő minden egyes cellára.

Az oszlopok esetében teljesen hasonlóan járunk el. Egyetlen rész maradt vissza, ami a teljes algoritmus pontos leírásához kell. Ha soronként töltjük fel a dinamikus programozási táblázat értékeit, akkor minden egyes pozícióban más és más oszlop blokkrendszerét változtatjuk meg. De ez természetesen megtehető, ha minden egyes blokkrendszert tárolunk. Így az algoritmus teljes futási ideje valóban $O(nm(\lg n + \lg m))$.

13.1.5. Két szekvencia hasonlósága, Smith–Waterman algoritmus

Két biológiai szekvenciának nem csak a távolságát, de a hasonlóságát is mérni tudjuk. Két karakter hasonlóságára, $S(a, b)$ -re, a leggyakrabban használt hasonlósági függvény az úgynevezett *log-odds*:

$$S(a, b) = \log \left(\frac{\Pr\{a, b\}}{\Pr\{a\} \Pr\{b\}} \right), \quad (13.28)$$

ahol $\Pr\{a, b\}$ a két karakter együttes valószínűsége, $\Pr\{a\}$ és $\Pr\{b\}$ pedig a marginális valószínűségek. A hasonlóság pozitív, ha $\Pr\{a, b\} > \Pr\{a\} \Pr\{b\}$, egyébként meg negatív. A hasonlósági értékeket empirikus adatokból határozzák meg, aminosavak esetében a legismertebbek a PAM és a BLOSUM hasonlósági mátrixok.

Ha a beszúrásokat és törléseket negatív értékekkel büntetjük, akkor az előbbi alfejezetben megadott algoritmusok hasonlóságokkal ugyanúgy működnek, csak minimalizálás helyett maximalizálni kell.

Hasonlóság alapú értékeléssel azonban lehet egy speciális problémát definiálni, a maximális lokális hasonlóság problémáját, vagy más néven a lokális szekvenciaillesztés problémáját: adott két szekvencia, egy hasonlósági mátrix és egy részéértékelési séma, a feladat az, hogy adjuk meg a két szekvencia két olyan részstringjét, amelyek hasonlósága maximális, valamint adjunk meg egy ilyen illesztést is. A *részstring* egy szekvencia szomszédos karakterekből álló részszekvenciáját értjük. A probléma biológiai motivációja az, hogy a biológiai szekvenciák egyes részei lassan, míg más részek gyorsabban evolválódnak. A lokális szekvenciaillesztés a legkonzervatívabb részt találja meg, a legelterjedtebb felhasználása az adatbázisokban való keresés. Ugyanis a lokális illesztésből származó hasonlósági érték hatékonyabban tudja elkülöníteni a homológ és nem homológ szekvenciákat, ugyanis a statisztikát nem rontják le a változó szakaszokból adódó negatív értékek.

A Smith-Waterman algoritmus a következőképpen működik: A kezdeti feltételek a nulldik sorra és oszlopra:

$$d_{0,0} = d_{i,0} = d_{0,j} = 0. \quad (13.29)$$

A dinamikus programozási táblázat kitöltése lineáris részbüntetés mellett:

$$d_{i,j} = \max\{0; d_{i-1,j-1} + S(a_i, b_j), d_{i-1,j} + g; d_{i,j-1} + g\}. \quad (13.30)$$

Itt a g részbüntetés most negatív érték. A táblázat kitöltése után megkeressük a táblázat legnagyobb értékét, ez lesz a lokálisan legjobb illesztés hasonlósága, majd innen az optimális értékeket adó cellákon át haladunk vissza a 0 értékig, ez a visszakeresés adja meg az illesztést, hasonlóan a távolság alapú módszerekhez.

Könnyű bizonyítani, hogy az így megadott illesztés lokálisan a legjobb lesz: ha az illesztést a vége felől ki tudnánk úgy terjeszteni, hogy az illesztés értéke az aktuális illesztésnél nagyobb legyen, akkor lenne nagyobb érték a dinamikus programozási táblázatban. Ha az illesztés elejét lehetne megtoldani egy pozitív értékű illesztéssel, akkor a dinamikus programozási táblázatban nem 0 szerepelne a lokális illesztés kezdeténél.

13.1.6. Többszörös szekvenciaillesztés

Kettőnél több szekvencia egyszerre történő illesztését először Sankoff ismertette. Az első alkalmazása egy lokális optimalizálás háromszoros illesztéssel volt, mellyel egy bináris fa

belső csúcsaira adtak meg konszenzus szekvenciákat. Mára a bioinformatika egyik kulcskérdésévé vált a gyors és adekvát többszörös szekvencia illesztés, Dan Gusfield a bioinformatika Szent Gráljának nevezi. Ma a többszörös illesztés egyformán elterjedt az adatbázisokban való keresésre, valamint evolúciós leszármazások vizsgálatára. Segítségével meg lehet találni egy szekvencia család konzervatív régióit, azokat a pozíciókat, amelyek az adott fehérjecsalád funkcionális tulajdonságát kialakítják. Arthur Lesk szavaival: *Amit két homológ szekvencia suttog, azt egy többszörös illesztés hangosan kiáltja.*

A többszörös illesztés egymás alá írt k -asait illesztett k -asoknak hívjuk. A többszörös illesztés dinamikus programozási algoritmusára egyszerű általánosítása a páronkénti illesztés algoritmusának: k szekvencia illesztéséhez egy k dimenziós dinamikus programozási táblázatot kell kitölteni. A táblázat minden egyes elemének a kiszámításához ismerni kell azokat az elemeket, amelyeknek valahány indexe eggyel kisebb, ha nem engedünk meg többszörös réseket, és a koordinátatengelyekkel párhuzamos hipersíkok minden kisebb indexű elemét, ha többszörös réseket megengedünk. Így ezen algoritmusok memóriaigénye k darab, egyenként n hosszúságú szekvencia esetén $\Theta(n^k)$, számolásigénye pedig $\Theta(2^k n^k)$, ha lineáris résbüntetést alkalmazunk, és $\Theta(n^{2k-1})$, ha tetszőleges résbüntetést alkalmazunk.

A többszörös szekvencia illesztéssel két alapvető probléma van. Az egyik algoritmuselméleti probléma: a pontos megoldáshoz szükséges idő a szekvenciák számával exponenciálisan nő. Bebizonyították, hogy a többszörös illesztés NP-teljes probléma. A másik metodikai probléma: nem világos, hogyan kell értékelni egy többszörös illesztést, ha több faj leszármazási sorrendjére vagyunk kíváncsiak. Objektív értékelési lehetőség csak akkor adódna, ha ismernénk a leszármazási viszonyokat, ekkor lehetne egy evolúciós fa mentén értékelni egy többszörös illesztést.

Mindkét problémára heurisztikus megoldást ad a fa mentén való iteratív páronkénti illesztés. Ez a módszer először egy úgynevezett *vezérfát* állít elő páronkénti távolságokból kiindulva (ilyen fakészítő módszerekkel találkozhatunk például a 13.5 alfejezetben), majd ezt használja fel többszörös illesztésre. Először a fa alapján szomszédos szekvenciákat illeszt, majd a már illesztett szekvencia párokhoz, hármaskhoz stb. illeszt az újabb szekvenciákat, szekvencia párokat, hármaskokat, stb. úgy, hogy a már illesztett szekvenciák illesztett k -asait nem lehet megbontani, csak egy csupa rés jelekből álló oszlopot beilleszteni. Így $k - 1$ páronkénti illesztéssel kapjuk meg k szekvencia többszörös illesztését. Sokszor a már illesztett szekvenciákat csak egy úgynevezett profillal ábrázolják. Egy profil egy $(|\Sigma| + 1) \times l$ -es táblázat, ahol l az illesztés hossza. Az egyes oszlopokban az adott pozíciójú illesztett k -asról készült statisztika található. Az egyes értékek azt mutatják, hogy az ábécé adott betűje hány százalékban szerepel az illesztés adott illesztett k -asában. Az oszlop utolsó helyén a rés jel százalékos előfordulása található.

Természetesen a kapott többszörös illesztés felhasználható egy újabb fa készítésére, amiből egy újabb illesztés generálható, és ezt a ciklust addig lehet ismételni, ameddig az újabb iteráció már nem hoz változást az illesztésben. A módszer magyarázata az a feltételezés, hogy a közeli szekvenciák optimális páronkénti illesztése ugyanaz, mint amit az optimális többszörös illesztésből kapunk. A módszer hátránya az, hogy még ha az előbbi feltételezés igaz is, akkor is lehet több egyformán optimális illesztés, és ezek száma is exponenciálisan nőhet a szekvencia hosszával. Például tekintsük az AUCGGUACAG és az AUCAUACAG szekvenciák alábbi két optimális illesztését:

```

A U C G G U A C A G   A U C G G U A C A G
A U C - A U A C A G   A U C A - U A C A G

```

A páronkénti illesztésben a kettő közül nem tudunk választani, viszont a többszörös illesztésben az egyik már jobb lehet a másíknál. Például ha ezekhez a szekvenciákhoz illesztjük az AUCGAU szekvenciát, akkor a két páronkénti optimális illesztésből kiindulva a következő két, lokálisan optimális illesztést kapjuk:

```

A U C G G U A C A G   A U C G G U A C A G
A U C - A U A C A G   A U C A - U A C A G
A U C G A U - - - -   A U C - G - A U - -

```

melyből a baloldali valóban globálisan optimális, a jobboldali azonban csak lokálisan optimális.

Így az iteratív illesztés csak egy lokális optimumot határoz meg. További hátránya ennek a heurisztikus eljárásnak az, hogy nem képes egy felső becslést adni arra nézve, hogy a kapott illesztés súlya legfeljebb hányszorosa az optimális illesztés súlyának. Mindezek ellenére ez a módszer a leginkább elterjedt a gyakorlatban, mert viszonylag egyszerű és gyors, és általában biológiailag helyes eredményt ad.

Meg kell említeni egy új heurisztikus módszert a többszörös illesztésre, amelyik nem dinamikus programozáson, hanem mohó algoritmuson alapszik. A DiAlign résmentes homológ részeket keres szekvenciák páronkénti összehasonlításával. A kapott részstringek résmentes illesztéseit hívjuk kiátlóknak, hiszen a dinamikus programozási táblázatban az ezeknek az illesztéseknek megfelelő utak átlósan helyezkednek el. Innen ered a módszer neve is: Diagonal Alignment. Ezután a homológ párokat mohó módon fűzi össze többszörös illesztéssé. Az átlókat egy heurisztikus módszer szerint értékeljük az alapján, hogy mekkora hasonlósági értékeket kap az adott átló, illetve milyen hasonlósági értékű átlókkal nem kompatibilis az adott átló. Két átló akkor nem kompatibilis, ha nem szerepelhetnek egy közös (többszörös) illesztésben. Az átlókat az értékelésük alapján sorrendbe rakjuk, majd kiválasztjuk a legnagyobb értékűt. Ezután töröljük a listából az összes olyan átlót, amely nem kompatibilis a kiválasztott átlóval, majd kiválasztjuk a megmaradt átlók közül a legnagyobb értékűt, és így tovább, amíg van kiválasztható átló. A többszörös illesztést a kiválasztott átlók uniója adja, amely nem feltétlenül fedi le a megadott szekvenciák összes karakterét. Azon karakterek, amely egyetlen kiválasztott átlóban sem fordulnak elő, „nem illeszthető” minősítést kapnak. Bár a módszer hátránya az, hogy néha indokolatlanul nagy részeket tesz a többszörös illesztésbe, mivel egyáltalán nem bünteti a részeket, a DiAlign a gyakorlatban az egyik legjobb heurisztikus algoritmusnak bizonyult.

13.1.7. Memóriaredukció Hirschberg algoritmusával

Ha két szekvenciának csak a távolságát vagy hasonlóságát akarjuk megadni, de egy optimális illesztésüket nem, akkor lineáris vagy affin résbüntetés esetén ezt nagyon könnyű lineáris memóriagénnnyel megtenni. Vegyük észre ugyanis, hogy a dinamikus programozási táblázatban mindig csak a megelőző sorra van szükségünk, a korábbi sorokat elfelejthetjük. Ha azonban egy optimális illesztést is meg akarunk adni, akkor szükségünk van a teljes táblázatra. Ha a táblázatot újra és újra kitöltve adjuk meg az optimális illesztéshez szükséges információkat, akkor meghatározhatjuk ezt lineáris memóriagénnnyel, de ekkor a számolási idő növekszik meg a szekvenciák hosszával köbös méretűre.

Meg lehet adni azonban egy olyan algoritmust is, amely négyzetes időben és lineáris memóriagénnnyel határozza meg két szekvencia optimális illesztését, ez *Hirschberg algo-*

ritmusa, amelyet lineáris részbüntetés és távolság alapú illesztés esetére mutatunk be.

Az algoritmus bemutatásához bevezetjük egy szekvencia tetszőleges szuffixének a jelölését, A^k jelöli az A szekvencia a_{k+1} -gyel kezdődő szuffixét, azaz a $(k + 1)$ -edik karaktertől a szekvencia végéig terjedő stringet.

Hirschberg algoritmusa először $A_{\lfloor |A|/2 \rfloor}$ -re és B -re hajt végre egy dinamikus programozási algoritmust, a fentebb vázolt lineáris memóriagénnyel (azaz mindig csak az aktuális és az előző sor értékeit tárolja), valamint egy hasonló dinamikus programozási algoritmust az $A^{\lfloor |A|/2 \rfloor}$ megfordítottján és a B szekvencia megfordítottján.

A két dinamikus programozás alapján tudjuk, hogy mi $A_{\lfloor |A|/2 \rfloor}$ és B tetszőleges prefixe optimális illesztésének az értéke, valamint $A^{\lfloor |A|/2 \rfloor}$ és B tetszőleges szuffix optimális illesztésének az értéke. Ebből már meg tudjuk mondani, hogy mi lesz A és B optimális illesztésének az értéke,

$$\min_j \{w(\alpha^*(A_{\lfloor |A|/2 \rfloor}, B_j)) + w(\alpha^*(A^{\lfloor |A|/2 \rfloor}, B^j))\}, \quad (13.31)$$

és a számolásból adódik, hogy az optimális illesztésben $A_{\lfloor |A|/2 \rfloor}$ B_j azon prefixével van illesztve, melyre

$$w(\alpha^*(A_{\lfloor |A|/2 \rfloor}, B_j)) + w(\alpha^*(A^{\lfloor |A|/2 \rfloor}, B^j)) \quad (13.32)$$

minimális.

Mivel a lineáris memóriagényű dinamikus programozásban is ismerjük a dinamikus programozási táblázat utolsó előtti sorát, meg tudjuk mondani, hogy $a_{\lfloor |A|/2 \rfloor}$ és $a_{\lfloor |A|/2 \rfloor + 1}$ illesztve van-e B szekvencia valamely karakterével, vagy az optimális illesztésben ezen karakterek törölődtek. Az is megállapítható a dinamikus programozási táblázat utolsó két-két sorából, hogy történt-e a B szekvencia valamely karaktereinek a beszúrása $a_{\lfloor |A|/2 \rfloor}$ és $a_{\lfloor |A|/2 \rfloor + 1}$ közé.

Így az optimális illesztésnek legalább két oszlopát határoztuk meg. Ezután $A_{\lfloor |A|/2 \rfloor - 1}$ -re és B fennmaradó prefixére valamint $A^{\lfloor |A|/2 \rfloor + 1}$ -re és B fennmaradó szuffixére ugyanígy járunk el, azaz mindkét szekvenciapárra két lineáris memóriagényű dinamikus programozást végzünk el. Ennek eredményeképpen az A szekvencia negyedénél és háromnegyedénél kapunk meg az optimális illesztésből minimum két-két oszlopot. A következő iterációban a negyedtelt szekvenciákra végezzük el a fenti eljárást, és ezt folytatjuk addig, amíg az optimális illesztés összes oszlopát meg nem kapjuk.

Nyilvánvaló, hogy a memóriagény a szekvenciák hosszával csak lineárisan nő. Megmutatjuk, hogy a számolási igény továbbra is $\Theta(nm)$, ahol n és m a két szekvencia hossza. Ez abból adódik, hogy minden egyes iterációban legfeljebb feleannyit számolunk, mint az előző iterációban. Ugyanis egy A és B szekvenciapárra egy lépésben $|A| \times |B|$ mennyiséget számolunk, a következő lépésben viszont csak $(|A|/2) \times j^* + (|A|/2) \times (|B| - j^*)$ mennyiséget, ahol j^* az a pozíció, melyre a (13.31) képletben minimális értéket kapunk. Így a teljes számolási igény

$$nm \times \left(1 + \frac{1}{2} + \frac{1}{4} + \dots\right) = \Theta(nm). \quad (13.33)$$

13.1.8. Memóriaredukció saroklevágással

A dinamikus programozási algoritmus egyre hosszabb és hosszabb részszekvenciák illesztésével jut el a teljes szekvenciák illesztéséig. Ez az algoritmus gyorsabbá tehető, ha sikerül kiszűrni a részszekvenciák olyan rossz illesztéseit, amelyek biztosan nem vezetnek a teljes

szekvenciák egy optimális illesztéséhez. Az ilyen illesztéseket a dinamikus programozási táblázat jobb felső, valamint a bal alsó sarkában található pozíciókból a minimális értékeket adó pozíciókon át a $d_{0,0}$ -ba menő irányított utak adják meg, innen ered a technikának az elnevezése.

A legtöbb ilyen algoritmus egy úgynevezett teszt értéket használ. Ez a teszt érték egy előre megadott felső korlátja a két szekvencia evolúciós távolságának. A teszt értéket használó algoritmusok akkor tudják a két szekvencia távolságát kiszámítani, ha a megadott teszt érték valóban nagyobb a szekvenciák távolságánál. Ellenkező esetben az algoritmus nem jut el a jobb alsó sarkig, vagy ha eljut, az itt kapott érték hibás, nem egyezik meg az optimális illesztés súlyával. Így ezek az algoritmusok adatbázisokban való keresésre alkalmasak, amikor egy adott szekvenciához hasonló szekvenciákat kell kigyűjteni egy adatbázisból. A megadott teszt érték az a felső határ, amelyiknél kisebb távolságot adó illesztéseket kell megkeresni az adatbázisból.

Az alábbiakban két algoritmus leírását közöljük. Spouge algoritmusa általánosítása Fickett, valamint Ukkonen algoritmusainak. A másik algoritmus Gusfieldtől származik, amely példa olyan algoritmusra, amely a szekvenciák távolságánál kisebb teszt értéknél is eljut a bal alsó sarkig, de ekkor a kiszámított távolság nagyobb a megadott teszt értéknél, és ez jelzi, hogy a meghatározott távolság valószínűleg pontatlan.

Spouge algoritmusa csak azokat a $d_{i,j}$ mátrix elemeket számolja ki, amelyekre teljesül, hogy

$$d_{i,j} + |(n-i) - (m-j)|g \leq t, \quad (13.34)$$

ahol t a teszt érték, g az rések büntetése (hosszú rések nincsenek megengedve), n és m pedig a szekvenciák hossza. Az ötlete az algoritmusnak az, hogy bármely út $d_{i,j}$ -ből $d_{n,m}$ -be legalább $|(n-i) - (m-j)| \times g$ értékkel növeli meg az illesztés súlyát. Így, ha t legalább akkora, mint a szekvenciák távolsága, Spouge algoritmusa pontosan határozza meg a szekvenciák távolságát. Ez az algoritmus általánosítása Fickett, ill. Ukkonen algoritmusainak. Azok az algoritmusok szintén teszt értéket tartalmazó egyenlőtlenségeket használtak, de Fickett algoritmusában az egyenlőtlenség

$$d_{i,j} \leq t, \quad (13.35)$$

míg Ukkonen algoritmusában az egyenlőtlenség

$$|i-j|g + |(n-i) - (m-j)|g \leq t \quad (13.36)$$

alakú. Mivel mindkét esetben az egyenlőtlenség bal oldala nem nagyobb, mint Spouge egyenlőtlenségének a bal oldala, ezért ezek az algoritmusok legalább akkora részét számolják ki a dinamikus programozási táblázatnak, mint Spouge algoritmusa. Empirikus eredmények igazolják, hogy Spouge algoritmusa valóban gyorsabb. Az algoritmus kiterjeszthető konkáv résbüntető függvényekre is.

A k -eltérésű globális illesztés probléma Gusfieldtől származik, és a következő kérdést teszi fel: Van-e két szekvenciának olyan illesztése, amelynek a súlya nem nagyobb k -nál? A kérdést megválaszoló algoritmus futási ideje $O(kn)$, ahol n a hosszabb szekvencia hossza. Az algoritmus ötlete az az észrevétel, hogy a $d_{n,m}$ -ből $d_{0,0}$ -ba vezető, legfeljebb k súlyú utak nem tartalmaznak olyan $d_{i,j}$ pozíciót, amelyre $|i-j| > k/g$. Ekképpen az algoritmus csak azokat a $d_{i,j}$ mátrix elemeket számolja ki, amelyekre $(i-j) \leq k/g$, és figyelmen kívül hagyja a határelemek azon $d_{e,f}$ szomszédait, amelyekre $|e-f| > k/g$. Ha van a két szekvenciának

legfeljebb k súlyú illesztése, akkor $d_{n,m} \leq k$, és $d_{n,m}$ valóban a szekvenciák távolsága, ellenkező esetben $d_{n,m} > k$. Ez utóbbi esetben $d_{n,m}$ nem feltétlenül a szekvenciák távolsága: elképzelhető, hogy van olyan illesztés, amelyre az ezt meghatározó út kilép az $|i - j| \leq k/g$ egyenlőtlenség által definiált sávból, és az illesztés súlya mégis kisebb, mint a meghatározott sávban haladó legkisebb súlyú illesztés.

A sarokvágási technikát kiterjesztették olyan többszörös illesztésekre is, ahol egy illesztett k -ast a **páronkénti összeg** sémával értékelünk, azaz

$$SP_l = \sum_{i=1}^{k-1} \sum_{j=i+1}^k d(p_{i,l}, p_{j,l}), \quad (13.37)$$

ahol SP_l az l -edik illesztett k -as értékelése, $d(\cdot, \cdot)$ a $\Sigma \cup \{-\}$ halmazon definiált távolságfüggvény, k az illesztett szekvenciák száma, $p_{i,j}$ pedig a profil i -edik sorának j -edik eleme. Egy szekvencia l -suffixe az $(l + 1)$ -edik betűtől a szekvencia végéig terjedő részstring. Jelöljük $w_{i,j}(l, m)$ -lel az i -edik és a j -edik szekvencia l - illetve m -suffixének a távolságát. Carillo és Lipman algoritmus csak azokat a pozíciókat számolja ki, amelyekre

$$d_{i_1, i_2, \dots, i_n} + \sum_{j=1}^{k-1} \sum_{l=j}^k w_{j,l}(i_j, i_l) \leq t, \quad (13.38)$$

ahol t a teszt érték. Az algoritmus helyességét az bizonyítja, hogy a szekvenciák még nem illesztett suffixeinek a páronkénti összeg sémával adott optimális illesztésének a súlya nem lehet kisebb, mint a páronkénti illesztésből adódó súlyok összege. A $w_{i,j}(l, m)$ -ek a páronkénti illesztésekből számíthatók, így az algoritmus a gyakorlat számára elfogadható idő alatt ki tudja számolni hat darab 200 hosszúságú szekvencia optimális illesztését.

Gyakorlatok

13.1-1. Mutassuk meg, hogy egy n és egy m hosszúságú szekvencia esetén a lehetséges illesztések száma

$$\sum_{i=0}^{\min(n,m)} \frac{(n+m-i)!}{(n-i)!(m-i)!i!}.$$

13.1-2. Adjunk meg egy olyan értékelést és szekvenciák olyan sorozatát, amelyekre az optimális illesztések száma exponenciálisan nő a szekvenciák hosszával.

13.1-3. Adjuk meg Hirschberg algoritmusát többszörös szekvenciaillesztésre.

13.1-4. Adjuk meg Hirschberg algoritmusát affin részbüntető függvények esetén.

13.1-5. Adjuk meg a Smith–Waterman algoritmust affin részbüntetésekre.

13.1-6. Adjuk meg a Spouge-féle saroklevágási technikát affin részbüntetésekre.

13.1-7. Két szekvenciának egy többszörös illesztésükből levezetett páronkénti illesztése a következő: kivágjuk a többszörös illesztésből a két szekvenciára vonatkozó sort, ezeket egymás alá írjuk, majd elhagyjuk a csupa rést tartalmazó sorokat. Adjunk példát három szekvencia olyan optimális illesztésére, melyből valamelyik két szekvenciára levezetett páronkénti illesztés nem a két szekvencia optimális páronkénti illesztése.

13.2. Algoritmusok fákon

Az alább ismertetésre kerülő algoritmusok gyökereztetett fákon dolgoznak. A dinamikus programozás a gyökereztetett részfákon történő visszavezetéssel történik. Mint látni fogjuk, nem csak optimális eseteket határozhatunk meg, hanem ugyanakkora futási idő alatt algebrai kifejezéseket is meghatározhatunk.

13.2.1. A takarékosági elv kis problémája

A takarékosági (parszimónia) elv a biológiai szekvenciák változását minél kevesebb számú mutációval akarja leírni. Az alábbiakban csak cserékkel fogunk foglalkozni, azaz adottak egyenlő hosszú biológiai szekvenciák, és a feladat az, hogy adjuk meg a leszármazási kapcsolataikat a takarékosági elv alapján. Definiálhatjuk a takarékosági elv kis és nagy problémáját. A nagy problémában ismeretlen az evolúciós törzsfá topológiája, amely mentén a szekvenciák evolválódtak, a feladat ennek a topológiának a megkeresése, valamint ezen egy legtakarékosabb evolúciós történet megkeresése. Az így kapott megoldás tehát nem csak lokálisan – az adott topológiára nézve – lesz optimális, hanem globálisan is. Megmutatható, hogy a takarékosági elv nagy problémája NP-teljes probléma.

A kis problémában adott egy gyökeres fa topológia, és a feladat az, hogy keressünk egy legtakarékosabb evolúciós történetet ezen fa mentén. Az így kapott megoldás lokálisan optimális lesz, de nincs garancia a globális optimumra nézve. A szekvenciák minden egyes pozíciójához egymástól függetlenül kereshetjük meg a legtakarékosabb történetet, így elég azt az esetet megoldani, amikor a fa minden egyes levelén csupán egy karakter van megadva.

Ekkor egy evolúciós történetet a fa belső csúcsaihoz rendelt karakterekkel jellemezhetünk. Ha két szomszédos csúcson ugyanazt a karaktert látjuk, akkor a takarékosági elv alapján nem történt mutáció a két csúcst összekötő él mentén, egyébként meg egy mutáció történt. A naiv algoritmus végignézi az összes lehetséges hozzárendelést, ami nyilván lassú, hiszen a lehetséges címkézések száma exponenciálisan növekszik a fa leveleinek számával.

A dinamikus programozás a részfákon történő visszavezetéssel történik (Sankoff algoritmus). Most részfán csak azokat a részfákat értjük, amelyek egy belső csúcst, mint gyökeret tartalmaznak, és minden olyan csúcst, amely az adott gyökér alatt van. Így a részfák száma pontosan a belső csúcsok számával egyezik meg, és ezért egyértelműen beszélhetünk egy adott pont által definiált részfáról. Feltesszük, hogy egy adott r gyökerű részfa esetén ismerjük r minden t gyerekére és ω karakterre azt, hogy a t gyereke által definiált részfán minimum hány mutáció szükséges, ha a t csúcsban ω karakter található. Jelöljük ezt a számot $m_{t,\omega}$ -val. Ekkor

$$m_{r,\omega} = \sum_{t \in D(r)} \min_{\sigma \in \Sigma} \{m_{t,\sigma} + \delta_{\omega,\sigma}\}, \quad (13.39)$$

ahol a $D(r)$ r gyerekeinek a halmaza, Σ a lehetséges karakterek halmaza, $\delta_{\omega,\sigma}$ pedig 1 ha $\omega = \sigma$ és 0 egyébként. A (13.39) képlet helyessége abból adódik, hogy a megvizsgáltuk az összes lehetőséget arra vonatkozóan, hogy r gyerekeihez milyen karaktereket rendelhetünk, és azt már ismerjük, hogy ezen karakterek hozzárendelésekor legalább mennyi szubsztitúció szükséges az adott gyerek alatti részfán.

Az adott fa topológiájához szükséges mutációk száma $\min_{\omega \in \Sigma} m_{R,\omega}$, ahol R a fa gyökere. Egy legtakarékosabb evolúciós történetet a gyökérből visszafelé haladva a minimális értékeket adó karakterek beírásával kaphatunk meg. Ehhez természetesen minden r belső

csúcsra és ω karakterre tárolni kell $m_{r,\omega}$ -t.

A minimális mutációk számának meghatározásához $\Theta(n|\Sigma|^2)$ időre van szükség, ezután a legtakarékosabb történet megkeresése $\Theta(n|\Sigma|)$ időt vesz igénybe, ahol n a levelek száma. A teljes evolúciós történet meghatározása $\Theta(nl|\Sigma|^2)$ időt vesz igénybe, ahol l a szekvenciák hossza.

13.2.2. Felsenstein algoritmus

Felsenstein algoritmusában a bemenő adat DNS szekvenciák többszörös illesztése. Csak azokat a pozíciókat tekintjük, ahol az illesztésben nincs rés. Feltesszük, hogy az egyes pozíciók egymástól függetlenül evolválódtak, így egy evolúciós folyamat valószínűsége az egyes csúcson történt események valószínűségeinek a szorzata. Legyen adva egy fa topológiája, amely ábrázolja a szekvenciák leszármazási sorrendjét, valamint egy evolúciós modell, amely minden σ -ra, ω -ra és t -re megmondja, hogy mi annak a valószínűsége, hogy σ karakter ω -vá evolválódik t idő alatt. Ezt $f_{\sigma\omega}(t)$ jelöli. Valamint ismerjük a karakterek egyensúlyi eloszlását, amit π jelöl. A kérdés az, hogy mennyi a fa likelihoodja, azaz a fa valószínűsége egy adott paraméterhalmaz mellett. Egy adott paraméterhalmaz mellett a fa likelihoodjának a kiszámítását egy adott fa topológiáján mutatjuk meg (13.1. ábra). Elég azt megmutatni, hogy hogyan kell a likelihoodot egy pozícióra kiszámítani, a fa teljes likelihoodja a pozíciók likelihoodjainak a szorzata. Az adott pozícióra s_i jelöli az i -edik csúc karakterét, v_j pedig az j -edik él evolúciós ideje, pontosabban a mutációs ráta és az idő szorzata. A belső pontok állapotait persze általában nem ismerjük, ezért minden lehetséges állapotra összegezni kell:

$$L = \sum_{s_0} \sum_{s_6} \sum_{s_7} \sum_{s_8} \pi_{s_0} \times f_{s_0 s_6}(v_6) \times f_{s_6 s_1}(v_1) \times f_{s_6 s_2}(v_2) \times f_{s_0 s_8}(v_8) \times f_{s_8 s_3}(v_3) \times f_{s_8 s_7}(v_7) \times f_{s_7 s_4}(v_4) \times f_{s_7 s_5}(v_5). \quad (13.40)$$

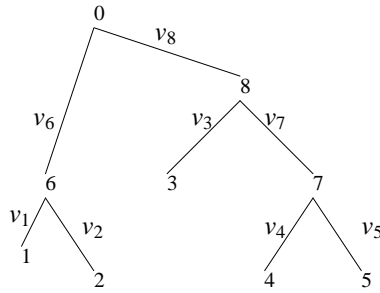
Ha négyelemű ábécét, azaz nukleinsav szekvenciákat tételezünk fel, akkor az összegzés 256 tagból áll, n faj esetén pedig 4^{n-1} tagból, ami könnyen lehet egy túl nagy szám. Szerencsére, ha az adott változótól nem függő szorzótényezőket kihozzuk a szumma jel elé, akkor az összegzés felbomlik egy

$$L = \sum_{s_0} \pi_{s_0} \left\{ \sum_{s_6} f_{s_0 s_6}(v_6) [f_{s_6 s_1}(v_1)] [f_{s_6 s_2}(v_2)] \right\} \times \left\{ \sum_{s_8} f_{s_0 s_8}(v_8) [f_{s_8 s_3}(v_3)] \left(\sum_{s_7} f_{s_8 s_7}(v_7) [f_{s_7 s_4}(v_4)] [f_{s_7 s_5}(v_5)] \right) \right\} \quad (13.41)$$

szorzatra, aminek a számítási igénye jóval kisebb. Vegyük észre, hogy a (13.41) egyenletben a zárójelezések pontosan leírják a fa topológiáját. Minden összegzés külön elvégezhető, és ezeket az összegeket szorozzuk össze, így a számítási idő lecsökken $\Theta(|\Sigma|^2 n)$ -re egy pozícióra, a teljes fa likelihoodjának a kiszámítása $\Theta(|\Sigma|^2 nl)$ -re, ahol l a pozíciók száma.

Gyakorlatok

13.2-1. Adjunk algoritmust a súlyozott takarékosági elv kis problémájára, azaz amikor az egyes változásokat súlyozzuk, és a változások súlyainak az összegét akarjuk minimalizálni.



13.1. ábra. A fa, amin Felsenstein algoritmusát bemutatjuk. Az élekre írt v -k az éleken eltelt evolúciós időt jelölik.

13.2-2. Az egyes genomok géntartalma különbözik, egy adott faj valamely génje egy másik fajból hiányozhat. A géntartalom változására a legegyszerűbb modell az, amelyben két mutációt különböztetünk meg: egy gén törlődik egy genomból vagy egy gén megjelenik a genomban. Adott néhány faj géntartalma, valamint az ezen fajok leszármazását bemutató törzsfá. Adjuk meg azt az evolúciós történetet, amely minimális számú mutációval írja le az adott fajok evolúcióját.

13.2-3. Adott szekvenciákra és törzsfára adjuk meg a Maximum Likelihood evolúciós történetet, azaz a fa belső csúcsain azokat a szekvenciákat, amelyre a likelihood maximális.

13.2-4. Írjuk fel a takarékosági elv kis problémáját a (13.40) képletbe hasonló alakban (csak az összegzések helyett minimumok szerepeljenek), és mutassuk meg, hogy Sankoff algoritmusát tulajdonképpen Felsenstein algoritmusához hasonló átrendezésen alapszik.

13.2-5. Fitch algoritmusát a következőképpen működik. Minden r csúcshoz hozzárendelünk egy karakterhalmazt, C_r -t, a levelekhez egyelemű halmazokat, amelyek az adott levélen található karaktert tartalmazzák, minden r belső csúcshoz pedig

$$\begin{aligned} \bigcap_{t \in D(r)} C_t, & \text{ ha } \bigcap_{t \in D(r)} C_t \neq \emptyset, \\ \bigcup_{t \in D(r)} C_t & \text{ egyébként.} \end{aligned}$$

Miután elértünk a gyökérhez, a gyökérhez rendelt halmazból tetszőlegesen kiválasztunk egy karaktert, majd lefelé haladva minden egyes belső csúcshoz kiválasztjuk ugyanazt a karaktert, mint amit a felette levő csúcshoz rendeltünk, amennyiben szerepel ez a karakter az adott halmazban, egyébként egy tetszőleges karaktert választunk. Mutassuk meg, hogy így egy legtakarékosabb történethez jutunk. Mennyi lesz ezen algoritmus futási ideje?

13.2-6. Mutassuk meg, hogy a 13.2.1. pontban megadott algoritmussal minden lehetséges legtakarékosabb evolúciós történetet megkaphatunk. Adjunk példát olyan legtakarékosabb történetre, amelyet Fitch algoritmusával nem kaphatunk meg.

13.3. Algoritmusok sztochasztikus nyelvtanokon

Az alábbiakban generatív nyelvtanok sztochasztikus változataival fogunk foglalkozni. A sztochasztikus generatív nyelvtanok központi szerepet játszanak a modern bioinformatikában. Két nyelvtantípus terjedt el széles körben, a rejtett Markov-modellek leggyakoribb

alkalmazási területei fehérjék térszerkezetének jóslása, illetve génkeresés, a sztochasztikus környezetfüggetlen nyelvtanok pedig az RNS molekulák másodlagos szerkezetének jóslásában játszanak fontos szerepet.

13.3.1. Rejtett Markov-modellek: előre, hátra és Viterbi algoritmus

Az alábbiakban definiáljuk formálisan a *rejtett Markov-folyamatokat*. Legyen adva állapotok egy véges X halmaza. A halmazban van két kitüntetett elem, a kezdő és a végállapot. Az állapotokat két részhalmazra bontjuk, emittáló és nem-emittáló állapotokra. Egyelőre feltesszük, hogy csak a kezdő és a végállapot nem emittáló. Később látni fogjuk, hogy ez a feltételezés nem túl szigorú (lásd 13.3-3. gyakorlat).

Megadunk egy M transzformációs mátrixot, melynek egy m_{ij} eleme megadja az i -ből a j állapotba ugrás valószínűségét, így értelemszerűen a mátrix nem-negatív, és minden sor összege 1 (teljes valószínűség tétele). A végállapotra a mátrix nem tartalmaz sort, a kezdőre oszlopot.

Megadunk egy Σ ábécét, és minden emittáló állapotra egy eloszlásfüggvényt az ábécé elemein, amely megmondja, hogy az adott állapot mely valószínűséggel melyik karaktert fogja emittálni amikor a folyamat az adott állapotban van. π_{ω}^i -vel fogjuk jelölni annak a valószínűségét, hogy az i állapot az ω karaktert emittálja, feltéve persze, hogy az i állapotban van a folyamat. A folyamat a kezdő (*START*) állapotból indul, és a végállapotba (*END*) érkezik. A *START* állapotba ugrani nem lehet. Minden egyes diszkrét időpontban a folyamat továbbhalad, a megadott M mátrix szerint. Minden emittáló állapot emittál egy karaktert, mikor a folyamat az adott állapotban van. A folyamat attól válik rejtetté, hogy mi csak az emittált karakterek láncát látjuk, magát a folyamatot nem. Néha előfordul, hogy nem vezetünk be kezdő és végállapotot, ekkor meg kell adni egy kezdeti eloszlást, hogy a $T = 0$ időpontban melyik állapotban vagyunk (azaz melyik állapot emittál először). Három fontos kérdést tehetünk fel, melyeket dinamikus programozási algoritmussal fogunk megválaszolni.

Az első kérdésünk a következő: adott egy Markov-folyamat, és egy emittált szekvencia. Adjuk meg a Markov-folyamatot azt az utat, amelyik az adott szekvenciát emittálta, és a valószínűsége a legnagyobb.

A kérdéses út megtalálását Viterbi algoritmusával oldjuk meg, ami szintén dinamikus programozási algoritmus. Szokás szerint A_k jelöli az A szekvencia első k karakteréből álló szekvenciát, és a_k a k -edik karaktert. A dinamikus programozás azon alapszik, hogy ha ismerjük minden k -ra és i -re a $\Pr_{max} \{A_k, i\}$ valószínűséget, azaz az A_k szekvenciát emittáló, és aktuálisan az i állapotban végződő utak valószínűségei közül a maximálisat, akkor

$$\Pr_{max} \{A_{k+1}, j\} = \max_i (\Pr_{max} \{A_k, i\} m_{i,j} \pi_{a_{k+1}}^j) . \quad (13.42)$$

A képlet abból adódik, hogy annak a valószínűsége, hogy egy út az adott szekvenciát emittálta, egyszerűen a szorzata az egyes ugrások valószínűségének és az egyes emissziók valószínűségének. Ha adva van ilyen szorzatok halmaza, amelyekben az utolsó két szorzótényező ugyanaz (jelen esetben $m_{i,j} \pi_{a_{k+1}}^j$), akkor ezek közül az a maximális, amelyikben a többi szorzótényező szorzata maximális.

Ha megjegyezzük, hogy az adott $\Pr_{max} \{A_{k+1}, j\}$ kiszámításához melyik i -t használtuk fel, akkor nyomon tudjuk követni a maximális emittáló útvonalat. Az *END* állapot nem

emittál, így az algoritmus befejeződése

$$\Pr_{max} \{A\} = \Pr_{max} \{A, END\} = \max_i (\Pr_{max} \{A, i\} m_{i,END}) , \quad (13.43)$$

ahol $\Pr_{max} \{A\}$ a legvalószínűbb út valószínűsége. Ha nincsen végállapot, akkor

$$\Pr_{max} \{A\} = \max_i (\Pr_{max} \{A, i\}) . \quad (13.44)$$

Ha van *START* állapot, akkor a folyamat szükségképpen a *START* állapotból indul, úgyhogy $\Pr_{max} \{A_0, START\} = 1$. Ha nincs start állapot, akkor

$$\Pr_{max} \{A_1, j\} = p_j \pi_{a_1}^j , \quad (13.45)$$

ahol p_j annak a valószínűsége, hogy a folyamat a j állapotból indul.

A második kérdésünk a következő: ha adott egy Markov-folyamat, és egy emittált szekvencia, akkor mi annak a valószínűsége, hogy az adott Markov-folyamat az adott szekvenciát emittálta? Ez a valószínűség egyszerűen az emittáló utak valószínűségeinek az összege. Mivel azonban a lehetséges emittáló utak száma exponenciálisan nőhet a szekvencia hosszával, a naiv módszer, miszerint számoljuk ki minden egyes út valószínűségét, és adjuk ezeket össze, nem járható.

Dinamikus programozással azonban kiszámolható a kérdéses valószínűség. Ezt a dinamikus programozási algoritmust hívják FORWARD algoritmusnak, és nagyon hasonlít Viterbi algoritmusára, csak maximum helyett összegzések vannak benne. A dinamikus programozás azon alapszik, hogy ha ismerjük minden k -ra és i -re a $\Pr \{A_k, i\}$ valószínűséget, azaz az A_k szekvenciát emittáló, és aktuálisan az i állapotban végződő utak valószínűségeinek összegét, akkor

$$\Pr \{A_{k+1}, j\} = \sum_i \Pr \{A_k, i\} m_{i,j} \pi_{a_{k+1}}^j . \quad (13.46)$$

Az *END* állapot nem emittál, így az algoritmus $\Pr \{A\}$ -t a következőképpen számolja ki:

$$\Pr \{A\} = \Pr \{A, END\} = \sum_i \Pr \{A, i\} m_{i,END} . \quad (13.47)$$

A legvalószínűbb emisszió útját ki tudjuk számolni, és ebből megkaphatjuk azt, hogy a legvalószínűbb úton mely karaktert mely állapot emittálta. Azonban ennek az útvonalnak lesznek jobban és kevésbé megbízható részei. Ezért érdeklődhetünk a $\Pr \{a_k\text{-t az } i \text{ állapot emittálta} \mid \text{a folyamat az } A \text{ szekvenciát emittálta}\}$ valószínűség iránt is, ami a harmadik olyan kérdésünk, melyet dinamikus programozással válaszolunk meg. Ez a valószínűség nem más, mint azon utak valószínűségeinek az összege, melyekre az i állapot bocsátotta ki az a_k karaktert, osztva a teljes kibocsátási valószínűséggel. A kérdéses utak száma exponenciálisan nőhet a szekvencia hosszával, így a naiv algoritmus, amely megkeresi ezen utakat, és egyesével összeadja a valószínűségeiket, megint csak nem járható a gyakorlatban.

Először is ki kell számolni annak a valószínűségét, hogy egy folyamat az A^k szekvenciát emittálta, feltéve, hogy a_k -t a i állapot emittálta, ahol A^k az A szekvencia vége, a $k + 1$ -edik karaktertől kezdve. Ezt a Forward algoritmushoz hasonló BACKWARD algoritmussal lehet megadni. Jelöljük $\Pr \{A^k, i\}$ -vel annak a valószínűségét, hogy egy folyamat az A^k szekvenciát

emittálta, feltéve, hogy a_k -t a i állapot emittálta. Ekkor

$$\Pr\{A^k, i\} = \sum_j (\Pr\{A^{k+1}, j\} m_{i,j} \pi_{d_{k+1}}^j) . \quad (13.48)$$

Jelöljük a $\Pr\{a_k\text{-t az } i \text{ állapot emittálta} \mid \text{a folyamat az } A \text{ szekvenciát emittálta}\}$ valószínűséget $\Pr\{a_k = i \mid A\}$ -val.

$$\Pr\{a_k = i \mid A\} \Pr\{A\} = \Pr\{A \wedge a_k = i\} = \Pr\{A_k, i\} \Pr\{A^k, i\} , \quad (13.49)$$

amiből:

$$\Pr\{a_k = i \mid A\} = \frac{\Pr\{A_k, i\} \Pr\{A^k, i\}}{\Pr\{A\}} , \quad (13.50)$$

ami éppen a keresett valószínűség.

13.3.2. Sztochasztikus környezetfüggetlen nyelvtanok: belülről, kívülről és a CYK algoritmus

Megmutatható, hogy minden környezetfüggetlen nyelvtan átírható úgynevezett **Chomsky-féle normálformába**. A Chomsky-féle normálformában minden levezetési szabály $W_v \rightarrow W_y W_z$ vagy $W_w \rightarrow a$ alakú, ahol minden W nemterminális szimbólum, a pedig terminális szimbólum. A sztochasztikus környezetfüggetlen nyelvtanokban a levezetési szabályokhoz valószínűségeket rendelünk és minden nemterminális szimbólum lehetséges levezetéseihez rendelt valószínűségek összege 1.

Legyen adott egy sztochasztikus környezetfüggetlen nyelvtan és egy szekvencia (szó). Három kérdést tehetünk fel, az első: Mi a szekvencia levezetésének a valószínűsége, azaz a lehetséges levezetések valószínűségeinek az összege. A második kérdés az, hogy mi a legvalószínűbb levezetés, a harmadik pedig az, hogy mi annak a valószínűsége, hogy egy részsót egy adott W_x nemterminálisból kiindulva vezettük le, feltéve, hogy az adott szót vezettük le. Hasonlóan a rejtett Markov-folyamatokhoz, az első kérdésre két algoritmust adunk meg, a KÍVÜLRŐL, illetve a BELÜLRŐL algoritmusokat, melyek analógok az ELŐLRŐL, illetve HÁTULRŐL algoritmusokkal. A második kérdésre a CYK (Cocke–Younger–Kasami) algoritmus adja meg a választ, amely a VITERBI algoritmussal analóg. A harmadik kérdésre pedig a KÍVÜLRŐL és a BELÜLRŐL algoritmusok közös alkalmazásával kaphatunk választ. A bemutatásra kerülő algoritmusok hasonlóak a rejtett Markov-folyamatok algoritmusaihoz, a futási idők azonban lényegesen nagyobbak.

Jelöljük a $W_v \rightarrow W_y W_z$ levezetés valószínűségét $t_v(y, z)$ -vel, a $W_v \rightarrow a$ levezetés valószínűségét $e_v(a)$ -val. A BELÜLRŐL algoritmus minden $i \leq j$ -re és v -re kiszámolja az $\alpha(i, j, v)$ valószínűséget, ami annak a valószínűsége, hogy a W_v nemterminálisból levezetjük az a_i -től a_j -ig terjedő részsót. A dinamikus programozás kezdeti feltételei:

$$\alpha(i, i, v) = e_v(a_i) , \quad (13.51)$$

minden i -re és v -re. A fő rekurzió:

$$\alpha(i, j, v) = \sum_{y=1}^M \sum_{z=1}^M \sum_{k=i}^{j-1} \alpha(i, k, y) t_v(y, z) \alpha(k+1, j, z) , \quad (13.52)$$

ahol M a nemterminális szimbólumok száma. A dinamikus programozási táblázat egy felső háromszög mátrix minden nemterminális szimbólumra. A táblázat kitöltése a főátlóval kezdődik, és innen haladunk a jobb felső sarok felé, a főátlóval párhuzamosan töltve ki a mátrixot. A levezetés valószínűségét $\alpha(1, L, 1)$ adja meg, ahol L a szekvencia hossza, W_1 pedig a kezdő nemterminális. Az algoritmus futási ideje $\Theta(L^3 M^3)$, a memóriaigény $\Theta(L^2 M)$.

A KÍVÜLRŐL algoritmus minden $i \leq j$ -re és v -re a $\beta(i, j, v)$ számokat számolja ki, ami azon levezetések valószínűsége, melyben az a_i -től a_j -ig terjedő részsztót W_v vezet le osztva $\alpha(i, j, v)$ -vel, illetve 0, ha $\alpha(i, j, v) = 0$. Az algoritmus kezdeti feltételei:

$$\beta(1, L, 1) = 1, \quad (13.53)$$

$$\beta(1, L, v) = 0 \quad \text{ha } v \neq 1. \quad (13.54)$$

A fő rekurzió:

$$\begin{aligned} \beta(i, j, v) = & \sum_{y=1}^M \sum_{z=1}^M \sum_{k=1}^{i-1} \alpha(k, i-1, z) t_y(z, v) \beta(k, j, y) + \\ & \sum_{y=1}^M \sum_{z=1}^M \sum_{k=j+1}^L \alpha(j+1, k, z) t_y(v, z) \beta(i, k, y). \end{aligned} \quad (13.55)$$

A (13.55) képlet helyessége abból adódik, hogy végignézzük az összes lehetőséget, hogy az a nemterminális szimbólum, amely W_v -t levezette, mely részét vezette le a szekvenciának. Mint látható, a számoláshoz szükségünk van az α -kra, ilyen téren a KÍVÜLRŐL algoritmus eltér a HÁRULRŐL algoritmustól, amelyet futtathatunk anélkül, hogy az ELÖLRŐL algoritmust alkalmaztuk volna, míg a KÍVÜLRŐL algoritmus előtt mindenképpen le kell futtatni a BELÜLRŐL algoritmust.

A CYK algoritmus kezdeti értékeinek beállítása megegyezik a BELÜLRŐL kezdeti értékeinek beállítással, a fő rekurzió is nagyon hasonlít a BELÜLRŐL algoritmus rekurziójához, csak összeadás helyett maximalizálni kell:

$$\alpha_{\max}(i, j, v) = \max_y \max_z \max_{i \leq k \leq j-1} \alpha_{\max}(i, k, y) t_y(y, z) \alpha_{\max}(k+1, j, z). \quad (13.56)$$

A legvalószínűbb levezetés valószínűségét pedig $\alpha_{\max}(1, L, 1)$ adja meg. A legvalószínűbb levezetést az optimális értékeket adó levezetési szabályokon keresztül kaphatjuk meg.

Végezetül annak a valószínűségét, hogy W_v vezette le az a_i -től a_j -ig terjedő részsztót, feltéve, hogy az A szekvenciát vezettük le,

$$\frac{\alpha(i, j, v) \beta(i, j, v)}{\alpha(1, L, 1)} \quad (13.57)$$

adja meg.

Gyakorlatok

13.3-1. A reguláris nyelvtanokban a levezetési szabályok $W_v \rightarrow aW_y$ vagy $W_v \rightarrow a$ alakúak. Mutassuk meg, hogy minden rejtett Markov-folyamat sztochasztikus reguláris nyelvtan, de nem minden sztochasztikus reguláris nyelvtan rejtett Markov-folyamat.

13.3-2. Adjunk meg dinamikus programozási algoritmust, mely adott sztochasztikus reguláris nyelvtanra és A szekvenciára megadja a

- levezetés valószínűségét,
- a legvalószínűbb levezetést,
- valamint annak a valószínűségét, hogy a szekvencia egy adott a_i karakterét egy adott levezetési szabály vezette le.

13.3-3. Egy rejtett Markov-modellben lehetnek úgynevezett *csendes* vagy nem kibocsátó állapotok, melyek a rejtett Markov-modell ábrázolását elősegíthetik. Mutassuk meg, hogy minden rejtett Markov-modell, mely csendes állapotokat tartalmaz, átírható olyan rejtett Markov-modellé, amely nem tartalmaz csendes állapotokat, és ekvivalens az eredeti modellel, azaz ugyanazon szekvenciákat ugyanakkora valószínűséggel bocsátja ki.

13.3-4. A *páros rejtett Markov-modellek* olyan rejtett Markov-modellek, melyekben az egyes állapotok nem csak egy szekvenciába bocsátanak ki karaktereket, hanem kettőbe. Egyes állapotok csak az egyik szekvenciába, mások csak a másikba, ismét mások pedig mindkét szekvenciába bocsátanak ki karaktereket. Egy állapot egy lépésben mindegyik szekvenciába legfeljebb egy karaktert bocsáthat ki. Adjuk meg a páros rejtett Markov-folyamatok VITERBI, ELŐRE és HÁTRA algoritmusait.

13.3-5. Viterbi algoritmusában nem használtuk ki, hogy a tranzíciók és a kibocsátási valószínűségek valószínűségek, azaz nem negatívak és egygyé összegződnek. Valamint az algoritmus akkor is működik, ha szorzás helyett összeadások vannak, és maximalizálás helyett akár minimalizálhatunk is. Adjunk meg egy olyan módosított páros rejtett Markov-modellt (amelyben a „valószínűségek” nem feltétlenül nem negatívak, és nem feltétlenül összegződnek egygyé), melyre Viterbi algoritmusával összeadásokkal és minimalizálással ekvivalens Gotoh algoritmusával.

13.3-6. Másodlagos térszerkezetnek nevezzük az RNS-ek olyan bázispárosodását, amelyben a bázispárosodott nukleinsavakat összekötő, a szekvencia fölött haladó ívek nem metszik egymást. A lehetséges bázispárosodások: $A - U$, $U - A$, $C - G$, $G - C$, $G - U$ és $U - G$. Adjunk meg egy dinamikus programozási algoritmust, amely egy adott RNS szekvenciára megkeresi azt a másodlagos térszerkezetet, melyben a párosodott nukleinsavak száma maximális.

13.3-7. A Knudsen–Hein-nyelvtan levezetési szabályai:

$$\begin{aligned} S &\rightarrow LS|L, \\ F &\rightarrow dFd|LS, \\ L &\rightarrow s|dFd, \end{aligned}$$

ahol minden s terminális levezetést még helyettesíteni kell az RNS szekvenciák lehetséges karaktereivel, a dFd kifejezésben pedig a két d terminális szimbólumot helyettesíteni kell a lehetséges bázispárosodásokkal. Mutassuk meg, hogy adott szekvencia és a levezetések adott valószínűségi eloszlása esetén meg lehet adni egy olyan dinamikus programozási algoritmust, amely megadja a szekvencia levezetésének a valószínűségét, anélkül, hogy Chomsky-féle normálformába íránk át a nyelvtant.

13.4. Szerkezetek összehasonlítása

Az alábbi részben különböző szerkezeteket hasonlítunk össze dinamikus programozási algoritmusok segítségével. Mint meg fogjuk mutatni, a címkézett, gyökeres fák illesztésére használt algoritmus általánosítása a szekvenciák összehasonlítására használt algoritmusnak.

A rejtett Markov-modellek összehasonlítását végző algoritmus egy lineáris egyenletrendszer megoldásával adja meg két rejtett Markov-modell együttes kibocsátásának a valószínűségét, azaz annak a valószínűségét, hogy két rejtett Markov-modell ugyanazt a szekvenciát bocsátja ki.

13.4.1. Címkézett, gyökeres fák illesztése

Legyen Σ egy véges ABC, $\Sigma^- = \Sigma \cup \{-\}$, $\Sigma^2 = \Sigma^- \times \Sigma^- \setminus \{-, -\}$. Egy F fa címkézésén egy olyan függvényt értünk, mely az F fa minden egyes $n \in V_F$ csúcsához hozzárendeli Σ egy karakterét. Ha egy gyökeres fából kitörünk egy csúcsot, akkor a kitörölt csúcs gyerekei a kitörölt csúcs szülőjének a gyerekei lesznek. Amennyiben a fa gyökerét töröljük ki, a fa erdőre esik szét. Legyen A egy gyökeres fa, melynek csúcsai Σ^2 elemeivel vannak címkézve, az ezt meghatározó függvény legyen $c : V_A \rightarrow \Sigma^2$. Azt mondjuk, hogy A illesztése az F és G Σ karaktereivel címkézett, gyökeres fáknek, ha A címkézéseinek első és második koordinátáján vett megszorításával, és az így '–' szimbólummal címkézett csúcsok törlésével rendre az F és G fákat kapjuk vissza.

Legyen adva egy hasonlósági függvény, $s : \Sigma^2 \rightarrow R$. Erre a hasonlósági függvényre semmilyen megkötést nem teszünk, egy karakter lehet akár kevésbé hasonló önmagához, mint egy másik karakterhez. F és G fák optimális illesztésén egy olyan Σ^2 elemeivel címkézett A fát értünk, melyre

$$\sum_{n \in V_A} s(c(n)) \quad (13.58)$$

maximális. Ezt a fát $A_{F,G}$ -vel fogjuk jelölni. Vegyük észre, hogy egy szekvencia ábrázolható olyan faként, melynek egyetlen levele van.

A továbbiakban csak olyan fakkal foglalkozunk, melyekben bármely csúcsnak legfeljebb két gyereke van. Az optimális illesztést megadó dinamikus programozás a gyökeres részfákon történik, ezek azon részfák, melyek a fa egy adott csúcsát, mint gyökeret és ezek leszármazottjait tartalmazzák. Az r gyökér által meghatározott részfát t_r -rel jelöljük. Egy fát egy üres fához csak egyféleképpen illeszthetünk. Két, a , ill. b karakterekkel címkézett levél illesztése csak három módon lehetséges: az illesztés vagy egyetlen csúcsot tartalmaz, és (a, b) -vel van címkézve, vagy két csúcsot tartalmaz, ezek közül az egyik $(a, -)$, a másik $(-, b)$ címkézésű, és a két csúcs közül az egyik a gyökér, a másik ennek a gyereke. Ez utóbbi két lehetőség ekvivalens a hasonlósági függvény szempontjából.

Hasonlóan, egy levelet egy gyökeres részfával úgy lehet összeilleszteni, hogy az A illesztésben vagy együtt van címkézve a levél karaktere a részfa valamely karakterével, vagy egy '–' szimbólummal van együtt címkézve. Ez utóbbi címkézést tartalmazó csúcsot a részfa sokféleképpen lehet beszúrni, de ezek mindegyike ekvivalens.

Ezután az inicializáció után a dinamikus programozásban egyre nagyobb részfákat illesztünk össze. Feltehetjük, hogy t_r , illetve t_s részfák esetében ismerjük már az A_{t_r, t_r} , A_{t_r, t_s} , A_{t_u, t_s} , A_{t_v, t_s} , A_{t_u, t_u} , A_{t_u, t_v} , A_{t_v, t_u} és A_{t_v, t_v} illesztéseket, és ezen illesztések értékeit, ahol u és v csúcsok r gyerekei, x és y csúcsok pedig s gyerekei (amennyiben valamelyik csúcsnak csak

egy gyereke van, akkor természetesen kevesebb részproblémára vezetjük vissza a problémát). Valamint ismerjük a t_u, t_v, t_x és t_y fák az üres részfához való illesztésének az értékét is. Legyen r címkézése a , s címkézése b . Ezek után A_{r,t_s} meghatározásához konstans sok lehetőséget kell végignézni: vagy az egyik részfa a másik részében valamely gyerekhez van illesztve, és ekkor a másik gyerek és a gyökér a '–' szimbólummal címkéződik az illesztésben, vagy r és s összeillesztődik, vagy bár nem illesztődnek össze, de A_{r,t_s} -ben az egyik gyökérnek megfelelő csúcs a gyökér, a másik pedig ennek a gyereke. Ez utóbbi két esetben a gyerekeket vagy összeillesztjük, vagy nem. Zz öt lehetséges eset.

Mivel a lehetséges gyökeres részfák száma egyenlő a fa csúcsainak számával, az optimális illesztés megkereshető $\Theta(|F||G|)$ időben, ahol $|F|$ és $|G|$ F és G csúcsainak száma.

13.4.2. Két rejtett Markov-modell együttes kibocsátási valószínűsége

Legyen adva két Markov-modell, M_1 és M_2 . A két modell együttes kibocsátási valószínűsége definíció szerint:

$$C(M_1, M_2) = \sum_s \Pr_{M_1} \{s\} \Pr_{M_2} \{s\}, \quad (13.59)$$

ahol az összegzés az összes lehetséges szekvencián megy, $\Pr_M \{s\}$ pedig annak a valószínűsége, hogy az M modell az s szekvenciát bocsátotta ki. Azt, hogy a p út az s szekvenciát bocsátotta ki, $e(p) = s$ -sel jelöljük, egy *START* állapottól x állapotig tartó utat pedig $[x]$ -szel. Mivel a kibocsátási valószínűség a lehetséges kibocsátó utak valószínűségeinek az összege,

$$\begin{aligned} C(M_1, M_2) &= \sum_s \left(\sum_{p_1 \in M_1, e(p_1)=s} \Pr_{M_1} \{p_1\} \right) \left(\sum_{p_2 \in M_2, e(p_2)=s} \Pr_{M_2} \{p_2\} \right) \\ &= \sum_{p_1 \in M_1, p_2 \in M_2, e(p_1)=e(p_2)} \Pr_{M_1} \{p_1\} \Pr_{M_2} \{p_2\}. \end{aligned} \quad (13.60)$$

Ez utóbbi képletben figyelembe kell venni, hogy egy útvonalra több lehetséges kibocsátás van, az összegzések a lehetséges útvonalak és kibocsátások együtteseinek, az útvonal valószínűségébe pedig beleértjük a kibocsátási valószínűségeket is. Jelöljük \bar{p}_1 -gyel azt az útvonalat, amelyet p_1 -ből kapunk a végállapot elhagyásával, valamint p_1 -nek az END_1 állapot előtti állapota legyen x_1 . (\bar{p}_2 -t és x_2 -t hasonlóan definiáljuk.) Ekkor

$$\begin{aligned} C(M_1, M_2) &= \sum_{p_1 \in M_1, p_2 \in M_2, e(p_1)=e(p_2)} m_{x_1, END_1} m_{x_2, END_2} \Pr_{M_1} \{\bar{p}_1\} \Pr_{M_2} \{\bar{p}_2\} \\ &= \sum_{x_1, x_2} m_{x_1, END_1} m_{x_2, END_2} C(x_1, x_2), \end{aligned} \quad (13.61)$$

ahol $m_{x, END}$ az x -ből az END állapotba ugrás valószínűsége, valamint

$$C(x_1, x_2) = \sum_{[x_1] \in M_1, [x_2] \in M_2, e([x_1])=e([x_2])} \Pr_{M_1} \{[x_1]\} \Pr_{M_2} \{[x_2]\}. \quad (13.62)$$

$C(x_1, x_2)$ -t meg lehet adni a következő képlettel is:

$$C(x_1, x_2) = \sum_{y_1, y_2} m_{y_1, x_1} m_{y_2, x_2} C(y_1, y_2) \sum_{\sigma \in \Sigma} \Pr \{\sigma | x_1\} \Pr \{\sigma | x_2\}, \quad (13.63)$$

ahol $\Pr\{\sigma|x_i\}$ annak a valószínűsége, hogy az x_i állapot σ -t bocsátotta ki. A (13.63) képlet egy lineáris egyenletrendszer definiál az összes x_1 és x_2 kibocsátó állapotokra. A kezdeti feltételek:

$$C(\text{START}_1, \text{START}_2) = 1, \quad (13.64)$$

$$C(\text{START}_1, x_2) = 0, \quad x_2 \neq \text{START}_2, \quad (13.65)$$

$$C(x_1, \text{START}_2) = 0, \quad x_1 \neq \text{START}_1. \quad (13.66)$$

Azonban a dinamikus programozás a szokásostól eltérően nem egy táblázat kitöltésével, hanem a (13.63) képlet által meghatározott egyenletrendszer megoldásával történik. Így az együttes kibocsátási valószínűség meghatározható $O((n_1 n_2)^3)$ időben, ahol n_i az M_i modellben a kibocsátó állapotok száma.

Gyakorlatok

13.4-1. Adjuk meg két fa lokális hasonlóságát, ami a két fa leginkább hasonló részfái illesztésének az értéke. Részfán most a fa tetszőleges összefüggő részét értjük.

13.4-2. Rendezett fák olyan gyökeres fákat értünk, melyben minden csúcs gyerekei rendezve vannak. Rendezett fák rendezett illesztése megőrzi a két fa gyerekeinek a rendezését. Adjunk olyan algoritmust, amely két rendezett fának egy optimális rendezett illesztését adja meg, és a számolási igénye mind a fák csúcscsámainak, mind a gyerekszám maximális értékének polinomiális függvénye.

13.4-3. Vegyük azt a végtelen dimenziós euklideszi teret, melynek a koordinátái a lehetséges szekvenciák. Minden rejtett Markov-modell egy vektorral adható meg ebben a térben, a vektor j -edik koordinátája megadja a j -edik szekvencia levezetésének a valószínűségét. Határozzuk meg két rejtett Markov-modell által bezárt szöveget ebben a térben.

13.4-4. Adjuk meg egy rejtett Markov-modell által kibocsátott szekvenciák hosszainak generátorfüggvényét, azaz a

$$\sum_{i=0}^{\infty} p_i \xi^i$$

függvényt, ahol p_i annak a valószínűsége, hogy a rejtett Markov-modell i hosszúságú szekvenciát bocsájt ki.

13.4-5. Adjuk meg egy páros rejtett Markov-modell által kibocsátott szekvenciák hosszainak generátorfüggvényét, azaz a

$$\sum_{i=0}^{\infty} \sum_{j=0}^{\infty} p_{i,j} \xi^i \eta^j$$

függvényt, ahol $p_{i,j}$ annak a valószínűsége, hogy a rejtett Markov-modell által kibocsátott első szekvencia i , a második pedig j hosszúságú.

13.5. Törzsfakészítés távolságon alapuló algoritmusokkal

Ebben a fejezetben törzsfákban olyan összefüggő, irányítatlan, súlyozott élű, körmentes gráfokat értünk, melyben semelyik csúcshoz nincs kettes fokszáma. A súlyok nem negatívak, és minden olyan élre, mely két belső csúcst köt össze, a súlyok pozitívak. Olyan törzsfakészítő módszerekkel ismerkedünk meg, amelyek bemenő adatai objektumok halmaza,

valamint mindegyik objektumpárra megadott távolság. Ez a távolság származhat például szekvenciák optimális illesztéséből vett távolságokból, de az itt bemutatásra kerülő algoritmusok tetszőleges távolságokra működnek. A fák levelei a megadott objektumok, a fa topológiáját és a fa éleinek a hosszát pedig a távolságadatokból származtatjuk. Minden fa ábrázol egy, a leveleken, mint objektumokon definiált metrikát: két objektum közötti távolságot az ezen objektumokat összekötő út hosszával definiáljuk. Az algoritmusok jóságát lehet mérni a bemeneti távolságok és a megkonstruált fa által meghatározott távolságok közötti különbséggel.

Két speciális metrikát fogunk definiálni, az ultrametrikát és az additív metrikát. Az osztályozó algoritmusok mindig olyan törzsfát készítenek, amelyek ultrametrikát reprezentálnak. Be fogjuk bizonyítani, hogy amennyiben a bemeneti adatokban szereplő távolságok ultrametrikus tulajdonságúak, akkor az osztályozó algoritmusok által meghatározott fa pontosan ezt fogja reprezentálni.

Hasonlóan a szomszédok egyesítése módszer additív metrikát reprezentáló fát készít, és ha a bemenő távolságokra teljesül az additív metrika, akkor a szomszédok egyesítése visszaadja ezt a metrikát.

Mindkét bizonyítás esetében szükségünk lesz az alábbi lemmára:

13.3. lemma. *Bármely metrikára legfeljebb egy olyan fa van, amely ezt ábrázolja.*

Bizonyítás. Két objektumra az állítás triviális. A bizonyítás indirekten, teljes indukcióval történik. Az indukciót három objektummal kezdjük. Három objektum esetében egyetlen fa topológia létezik, a csillag alakú. Legyen az i , j és k leveleket a fa belső csúcsával összekötő élek hossza rendre x , y és z . Az élek hosszait az

$$x + y = d_{i,j}, \quad (13.67)$$

$$x + z = d_{i,k}, \quad (13.68)$$

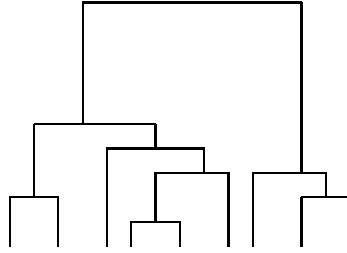
$$y + z = d_{j,k}, \quad (13.69)$$

egyenletrendszer adja meg, aminek egyetlen megoldása van, mivel az

$$\begin{vmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{vmatrix} \quad (13.70)$$

determináns nem 0.

$n > 3$ objektum esetében tegyük fel, hogy van két fa, amely ugyanazt a metrikát reprezentálja. Ekkor az első fán keressünk két olyan levelet, i -t és j -t, melyeket összekötő úton egyetlen egy csúcs van, legyen az a csúcs u . Ilyen i és j csúcsokat minden fában találunk, vegyünk egy olyan utat a fában, melyben a belső csúcsok száma a legnagyobb, az út mindkét végén ilyen levélpár található. Ha a második fában i -t és j -t összekötő úton egyetlen csúcs van, akkor a két fában i -t a belső csúccsal összekötő élek hossza azonos, és úgyszintén a j -t a belső csúccsal összekötő élek hossza azonos, mivel tetszőleges k ($k \neq i, j$) objektumra mindkét fa esetében ugyanazt a részfát kell kapnunk (melyben az u és k közötti utat egyetlen $d_{u,k}$ hosszúságú éllel ábrázoljuk). Legyártunk egy új metrikát, melyben elhagyjuk az i és j objektumokat, bevezetünk egy u' objektumot, melynek bármely k objektumtól vett távolsága $d_{i,k} - d_{i,u}$, ahol $d_{i,u}$, az i -t u -val összekötő él hossza. A két fában elhagyjuk az i és j csúcsokat,



13.2. ábra. Egy dendrogram.

ha u fokszáma 3 volt i és j elhagyása előtt, akkor u levél lesz az új fában, és most ez fogja reprezentálni u' -t, ha u nem levél i és j elhagyása után, akkor u -ba behúzzunk egy u' levelet, az új él hossza pedig 0. Így olyan fákat kapunk, melyek ezt a metrikát reprezentálják, és az indukció szerint azonosak.

Ha viszont a második fában i -t j -vel összekötő úton nem egy csúcs van, akkor ellentmondásra jutunk. Ugyanis ebben a fában van az i -t j -vel összekötő úton egy olyan u_1 csúcs, melyre $d_{i,u} \neq d_{i,u_1}$. Vegyünk a második fában egy olyan k csúcsot, melyre az i -t k -val összekötő út áthalad u_1 -en. Az első fából számolva

$$d_{i,k} - d_{j,k} = d_{i,u} - d_{j,u} = 2d_{i,u} - d_{i,j}, \quad (13.71)$$

míg a második fán

$$d_{i,k} - d_{j,k} = d_{i,u_1} - d_{j,u_1} = 2d_{i,u_1} - d_{i,j}, \quad (13.72)$$

ami ellentmond annak, hogy $d_{i,u} \neq d_{i,u_1}$. ■

13.5.1. Osztályozó algoritmusok

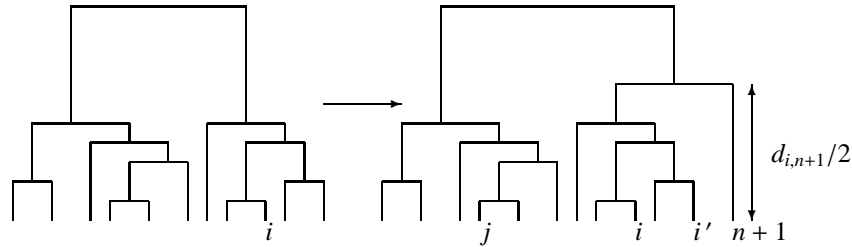
13.4. definíció. Egy metrikát **ultrametriának** nevezzük, ha bármely i , j és k csúcsokra

$$d_{i,j} \leq \max\{d_{i,k}, d_{j,k}\} \quad (13.73)$$

Könnyen belátható (lásd 13.5-1. gyakorlat), hogy egy ultrametriában bármely három csúcs között levő három távolság vagy mind azonos, vagy közülük kettő azonos, a harmadik pedig ennél kisebb.

13.5. tétel. Ha objektumok egy véges halmazán definiált metrika ultrametrika, akkor pontosan egy olyan fa létezik, amely ezt ábrázolja. Továbbá ezt a fát le lehet gyökereztetni úgy, hogy minden levélnek a gyökértől vett távolsága azonos legyen.

Bizonyítás. A 13.3. lemma alapján legfeljebb egy ilyen fa van, így elég megkonstruálni egy ilyen fát bármely ultrametriára. Az ultrametrius fákat dendrogramokként fogjuk ábrázolni, ezekben a reprezentációkban a vízszintesen húzott élek hosszát 0-nak tekintjük (lásd 13.2. ábra). A tétel bizonyítása a levelek, mint objektumok száma szerinti indukcióval történik. Kettő objektum esetében nyilván meg tudjuk konstruálni a dendrogramot. Ha n levélre

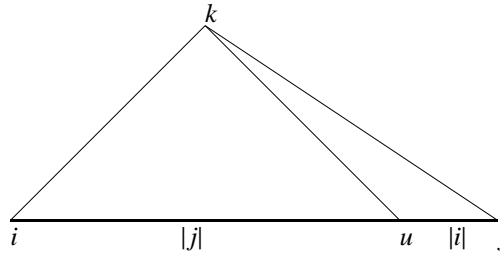
13.3. ábra. Az $(n + 1)$ -edik levél bekötése a dendrogramba.

megkonstruáltuk már a dendrogramot, az $(n + 1)$ -edik levéllel a következőképpen járunk el: Keresünk egy olyan i -t, melyre $d_{i,n+1}$ minimális. Ezután i -ből elindulunk a gyökér felé, és $d_{i,n+1}/2$ magasságban kötjük be az $(n + 1)$ -edik levelet (lásd 13.3. ábra). Ez a dendrogram helyesen ábrázolja az összes csúcshoz az $(n + 1)$ -edik levéltől vett távolságát. Ugyanis az összes olyan i' csúcsra, amely az $n + 1$ -ik levél bekötésének helye alatt helyezkedik el, $d_{i,i'} \leq d_{i,n+1}$, és az ultrametrikus tulajdonságból, valamint $d_{i,n+1}$ minimalitásából következik, hogy ekkor $d_{i,n+1} = d_{i',n+1}$. Másrészt az összes többi j csúcsra $d_{i,j} > d_{i,n+1}$, és az ultrametrikus tulajdonságból adódóan ekkor $d_{j,n+1} = d_{i,j}$. ■

Könnyen belátható, hogy a bizonyításban használt megkonstruálás számolási igénye $O(n^2)$, ahol n az objektumok száma. Megadhatunk egy másik algoritmust is, amely megkeresi azt az i és j objektumpárt, amire $d_{i,j}$ minimális. Az ultrametrikus tulajdonságból adódóan minden k -ra $d_{i,k} = d_{j,k} (\geq d_{i,j})$, így az i és j objektumpárt lecserélhetjük egyetlen új objektumra, és ezen új objektumnak az összes többitől vett távolsága jól definiált. Az i és j objektumot összekötjük $d_{i,j}/2$ magasságban, és az így kapott rész-dendrogramot egy objektumnak tekintve folytatjuk az iterációt. Ez az algoritmus lassabb, mint az előbbi bizonyításban megadott algoritmus, viszont ez az alapja az úgynevezett osztályozó algoritmusoknak. Az osztályozó algoritmusok mindig dendrogramot adnak, akkor is, ha a bemenő távolságok nem alkotnak ultrametrikát. Viszont ha a bemenő adatok ultrametrikus tulajdonságúak, akkor az osztályozó algoritmusok többsége pontosan visszaadja az ezt reprezentáló dendrogramot.

Mindegyik osztályozó algoritmus azt az i és j objektumokat (illetve az iteráció további lépéseiben objektumok helyett objektumhalmazok is szerepelhetnek) keresi meg, amelyre $d_{i,j}$ minimális. A módszerek közötti különbség abban rejlik, hogy ezután hogyan határozzák meg az új objektumhalmaz és a többi objektum(halmaz) közötti távolságot. Ha az új objektumot u -val jelöljük, akkor az alább ismertetésre kerülő módszerek következőképpen definiálják $d_{u,k}$ -t:

- EGYSZERŰ-LÁNC: $d_{u,k} = \min\{d_{i,k}, d_{j,k}\}$.
- TELJES-LÁNC: $d_{u,k} = \max\{d_{i,k}, d_{j,k}\}$.
- CSOPORTÁTLAG: (UPGMA) u és k elemei páronkénti távolságainak számtani közepe, azaz $d_{u,k} = (d_{i,k} \times |i| + d_{j,k} \times |j|) / (|i| + |j|)$, ahol $|i|$ és $|j|$ az i és j objektumhalmazok elemszámai.
- EGYSZERŰ-ÁTLAG: Az átlagok átlagát vesszük, azaz $d_{u,k} = (d_{i,k} + d_{j,k}) / 2$.
- CENTROID: Ezt a módszert leggyakrabban akkor alkalmazzák, amikor az objektumok



13.4. ábra. $d_{u,k}$ számolása a CENTROID módszer szerint.

beágyazhatóak Euklideszi térbe. Ekkor a két objektumhalmaz közötti távolságot az objektumhalmazok centrumai közötti távolságként lehet definiálni. A számoláshoz azonban nem feltétlenül kell az Euklideszi tér koordinátáit használni, hiszen a kérdéses $d_{u,k}$ távolság nem más, mint az i , j és k csúcsok által meghatározott háromszögben a k csúcsból kiinduló, az ij szakaszt $|j| : |i|$ arányban osztó szakasz hossza (lásd 13.4. ábra), ez pedig a $d_{i,j}$, $d_{i,k}$ és $d_{j,k}$ adatokból már meghatározható. Ez a számolási mód akkor is alkalmazható, ha az objektumok nem ágyazhatók be Euklideszi térbe, így bár a módszer ötlete geometriai indíttatású, bármely távolságmátrixra alkalmazható.

- **MEDIÁN:** Az u objektumhalmaz centrumát i és j centrumainak centrumaként definiáljuk. Így ez a módszer úgy viszonyul a centroid módszerhez, mint az egyszerű átlag a csoportátlaghoz. Erre a módszerre is igaz, hogy nem kell $d_{u,k}$ számolásához az Euklideszi koordinátákat ismerni, hiszen a keresett távolság az ijk háromszögben a k -ból induló súlyvonal hossza.

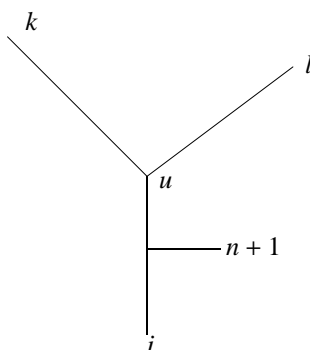
Könnyen belátható, hogy az első négy módszer visszaadja a távolságokat reprezentáló dendrogramot, amennyiben a bemenő adatok ultrametrikus tulajdonságúak, hiszen ekkor $d_{i,k} = d_{j,k}$. A CENTROID és a MEDIÁN módszerek azonban nem adják vissza az ultrametrikát reprezentáló dendrogramot, hiszen $d_{u,k}$ kisebb lesz, mint $d_{i,k}$ (ami egyenlő $d_{j,k}$ -val).

Az osztályozó algoritmusok általános problémája, hogy mindig dendrogramot adnak vissza, és ez biológiailag nem feltétlenül helyes. Ugyanis biológiai szekvenciák leszármazási kapcsolatait csak az úgynevezett *molekuláris óra* működésének esetében ábrázolja helyesen egy dendrogram. A molekuláris óra elmélet szerint az egyes szekvenciák a törzsfejlődés során adott időtartam alatt ugyanakkora mennyiségű mutáción mentek át, azonban számos biológiai példa mutatja azt, hogy ez nem mindig teljesül. Ezért szeretnénk egy olyan algoritmust, amely csak akkor ad ultrametrikus fát a bemenő adatsorokra, ha a bemenő távolságok valóban ultrametrikus tulajdonságúak. Ezért mára a SZOMSZÉDOK EGYESÍTÉSE algoritmus sokkal népszerűbbé vált a bioinformatikai alkalmazásokban, mint az osztályozó algoritmusok.

13.5.2. Szomszédok egyesítése

13.6. definíció. Egy metrikát *additív* vagy *négycsúcs metrikának* nevezünk, ha bármely i , j , k és l csúcsára

$$d_{i,j} + d_{k,l} \leq \max\{d_{i,k} + d_{j,l}, d_{i,l} + d_{j,k}\}. \quad (13.74)$$



13.5. ábra. Az $(n + 1)$ -edik levél gyökereztetése additív fa megkonstruálásához.

13.7. tétel. *Ha objektumok egy véges halmazán definiált metrika additív, akkor pontosan egy olyan fa létezik, amely ezt reprezentálja.*

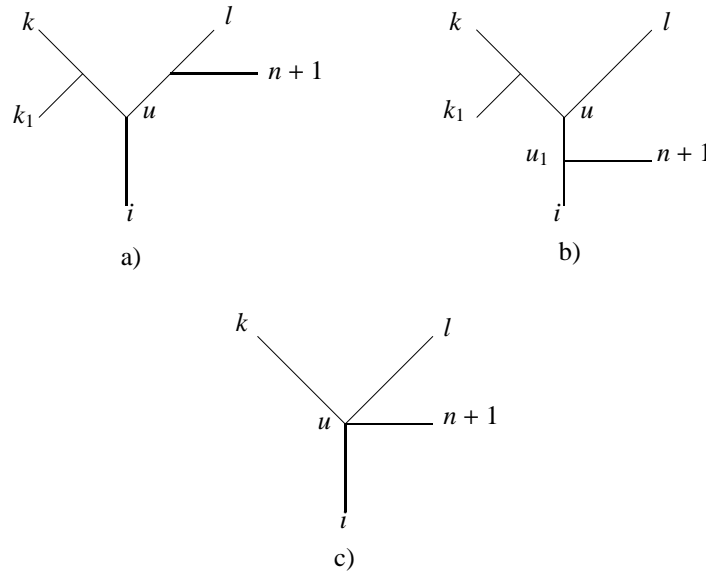
Bizonyítás. A 13.3. lemma alapján legfeljebb egy ilyen fa van, így elég megkonstruálni egy ilyen fát bármely additív metrikára. Először a konstrukciót adjuk meg, ezután bizonyítjuk a konstrukció helyességét:

Három objektumra a (13.67–13.69) egyenletek alapján megkonstruálunk egy fát, ezután a konstrukció indukcióval történik, feltesszük, hogy $n \geq 3$ objektumra már elkészítettük a fát, az $(n + 1)$ -edik objektumot reprezentáló levelet pedig egy éllel valahova bekötjük a már meglévő fába. Ehhez először meghatározzuk az új fa topológiáját, majd az új él hosszát. A topológia meghatározásához egy tetszőleges i levéltől indulunk el. Jelöljük i szomszédját u -val, u -ból még legalább két él indul ki, ezen élekből kiinduló utakon keressünk egy-egy levelet, jelöljük ezeket k -val és l -lel (lásd 13.5. ábra). Az $(n + 1)$ -edik levél bekötése i -ből nézve az u csúcson innen van, ha

$$d_{i,n+1} + d_{k,l} < d_{i,k} + d_{n+1,l} . \quad (13.75)$$

Hasonlóképpen megállapíthatjuk, hogy az $(n + 1)$ -edik levél bekötése k , illetve az l csúcsból nézve u -n innen van-e. Ha u fokszáma nagyobb, mint három, akkor a további élekből kiinduló utakon is keressünk l' csúcsokat, és az i , $n + 1$, k és l' csúcsnégyesekre hasonlóan járunk el. Az additív metrika tulajdonságából következik, hogy legfeljebb egy esetben állhat fenn az adott irányú egyenlőtlenség. Ha egyetlen esetben áll fenn ilyen irányú egyenlőtlenség, és ez az i levél esete, akkor az $(n + 1)$ -edik levelet az i -t u -val összekötő élhez kötjük be. Ha egyenlőtlenség más esetben áll fenn, akkor vesszük azt a maximális részfat, amelynek egy levele u , és tartalmazza az $(n + 1)$ -edik levél bekötését. Definiáljuk $d_{u,n+1}$ -et mint $d_{i,n+1} - d_{i,u}$, és ezután u -t i -nek átnevezve folytatjuk a bekötés helyének keresését. Ha minden esetben egyenlőséget kapunk, akkor az $(n + 1)$ -edik levelet az u csúcshoz kötjük be.

Miután megtörtént a bekötés helyének megkeresése, meghatározzuk a bekötő él hosszát. Ha az $(n + 1)$ -ik élt az i -t u -val összekötő élen kötjük be, akkor jelöljük a bekötő csúcsot u_1 -gyel (13.6/b ábra). Definiáljuk $d_{u,n+1}$ -et $(d_{l,n+1} - d_{l,u})$ -ként. Ekkor a d_{u,u_1} , d_{i,u_1} , valamint a $d_{u_1,n+1}$ távolságokat az i , u és $n + 1$ objektumok távolságait reprezentáló fa adja meg, melyet a (13.67–13.69) egyenletek alapján számolunk ki. Ha az $(n + 1)$ -edik levelet az



13.6. ábra. Néhány fa topológia a 13.7. tétel bizonyításához.

u csúcshoz kötjük be, akkor $d_{u,n+1} = d_{i,n+1} - d_{i,u}$.

Ezután rátérünk a konstrukció helyességének a bizonyítására. Először azt látjuk be, hogy amikor az $(n+1)$ -ik levél bekötésének a helyét keressük, és egy új részfán definiáljuk a $d_{u,n+1}$ távolságot, akkor a megadott definíció jól definiált, azaz bármely olyan j csúcsra, mely nem szerepel az újonnan definiált részfában, $d_{j,n+1} - d_{j,u} = d_{i,n+1} - d_{i,u}$. Ha az új részfa tartalmazza l -t, akkor ez azon $j = k$ csúcsra nyilván teljesül, mely k csúcs alapján határoztuk meg az $(n+1)$ -edik levél helyzetét (lásd 13.6/a ábra). Az $(n+1)$ -edik levél helyzetéből és az additív metrika tulajdonságából adódóan

$$d_{k,n+1} + d_{i,l} = d_{i,n+1} + d_{k,l}, \quad (13.76)$$

amiből ha felhasználjuk a $d_{i,l} = d_{i,u} + d_{u,l}$ és a $d_{k,l} = d_{k,u} + d_{u,l}$ egyenlőségeket, adódik, hogy

$$d_{k,n+1} - d_{k,u} = d_{i,n+1} - d_{i,u}. \quad (13.77)$$

Ugyanígy minden olyan k_1 levélre, melyet nem választ el k -tól az u csúcs, fennáll a

$$d_{k_1,n+1} + d_{i,l} = d_{i,n+1} + d_{k_1,l} \quad (13.78)$$

egyenlőség. Ez az additív metrikából és a

$$d_{k,k_1} + d_{l,n+1} < d_{k,n+1} + d_{k_1,l} \quad (13.79)$$

egyenlőtlenségből adódik, ez utóbbi pedig levezethető a

$$d_{k,k_1} + d_{i,l} < d_{k_1,l} + d_{k,i} \quad (13.80)$$

és

$$d_{l,n+1} + d_{k,i} < d_{i,l} + d_{k,n+1} \quad (13.81)$$

egyenlőtlenségekből. Hasonlóan, ha u fokszáma háromnál nagyobb, akkor minden olyan csúcra, melyet u elválaszt az $(n + 1)$ -edik levéltől, hasonló egyenlőségek és egyenlőtlenségek állnak fenn.

Az új élhosszak kiszámolásából adódik, hogy a $d_{i,n+1}$ távolságot helyesen reprezentálja az új fa, és így a $d_{j,n+1}$ távolságot az összes olyan j csúcra, melyet i elválaszt $(n + 1)$ -től. (Ne feledjük el, hogy a beágazás helyének megkereséséből adódóan az ábrán i lehet egy korábbi u .)

Ha az $(n + 1)$ -edik levél bekötése az u -t az i -vel összekötő szakaszon van (13.6/b ábra), akkor $d_{u,n+1}$ definíciójából adódóan $d_{l,n+1}$ -et is helyesen ábrázolja a fa. A

$$d_{k,n+1} + d_{i,l} = d_{k,i} + d_{l,n+1} \quad (13.82)$$

egyenlőségéből egyszerűen levezethető, hogy

$$d_{k,n+1} = d_{k,u} + d_{u,n+1} , \quad (13.83)$$

így a $d_{k,n+1}$ távolságot is jól reprezentálja a fa. Az előbbi levezetésekkel analóg módon belátható, hogy minden olyan k_1 -re, melyet u nem választ el k -től, a $d_{k_1,n+1}$ távolságot helyesen reprezentálja a fa, és valójában az összes olyan j csúcra, melyet u elválaszt $n + 1$ -től, a $d_{j,n+1}$ távolságot helyesen reprezentálja az elkészített fa.

Ha az $(n + 1)$ -edik levelet az u csúcshoz kötjük be (13.6/c ábra), akkor a

$$d_{i,n+1} + d_{k,l} = d_{k,i} + d_{l,n+1} = d_{k,n+1} + d_{j,i} \quad (13.84)$$

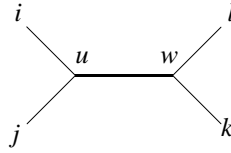
egyenlőségekből már levezethető, hogy mind $d_{k,n+1}$ -et, mind $d_{l,n+1}$ -et helyesen reprezentálja a fa, és a fentiekhez hasonló okoskodásból következik, hogy ez valójában igaz minden olyan csúcra, melyet u elválaszt $(n + 1)$ -től.

Ezzel n csúcs távolságait helyesen reprezentáló fából kiindulva megkonstruáltunk egy olyan fát, mely $n + 1$ csúcs távolságát reprezentálja helyesen (feltéve, hogy a csúcsokra teljesül az additív metrika), így bizonyítottuk a 13.7. tételt. ■

Könnyen belátható, hogy a fenti algoritmus, mely megkonstruálja azt a fát, amely egy additív metrikát reprezentál, $O(n^2)$ időt igényel. Az algoritmus azonban csak akkor működik helyesen, ha a bemenő távolságok additív metrikát alkotnak. Ellenkező esetben több esetben is fennállhat a (13.75) egyenlőtlenség, így nem tudnánk eldönteni, hogy hol kössük be az $(n + 1)$ -edik levelet. Az alábbiakban megadunk egy $\Theta(n^3)$ idejű algoritmust, amely szintén az additív metrikát reprezentáló fát adja vissza, ha a bemenő távolságok additív metrikát alkotnak, de egy additív fát ad vissza egyéb esetekben is.

A SZOMSZÉDOK-EGYESÍTÉSE algoritmus a következőképpen működik: Adott csúcsok egy halmaza n elemmel, és egy ezen értelmezett metrika, d . Először kiszámítjuk az összes i csúcra a többi csúcstól vett távolságok összegét:

$$v_i = \sum_{j=1}^n d_{i,j} . \quad (13.85)$$



13.7. ábra. Az i , j , k és l csúcsok elhelyezkedése, amennyiben i -t és j -t egyetlen u belső csúcs választja el.

Ezután megkeressük azt a csúcspárt, melyre

$$s_{i,j} = (n - 2)d_{i,j} - v_i - v_j \quad (13.86)$$

minimális. Az i és j csúcsokból az új, u belső csúcsig húzott élek hossza

$$e_{i,u} = \frac{d_{i,j}}{2} - \frac{v_i - v_j}{2n - 4}, \quad (13.87)$$

illetve

$$e_{j,u} = \frac{d_{i,j}}{2} - e_{i,u} \quad (13.88)$$

Ezután következik a távolságmátrix átszámolása. Az i és j csúcsok kiesnek, helyükre kerül be az u csúcs. Az u csúcs és a többi csúcs közötti távolságot az alábbi képlettel határozzuk meg:

$$d_{k,u} = \frac{d_{k,i} + d_{k,j} - d_{i,j}}{2}. \quad (13.89)$$

13.8. tétel. Ha a bemenő csúcsokon megadott d metrika additív metrika, akkor a SZOMSZÉDOK-EGYESÍTÉSE algoritmus visszaadja azt a fát, mely ezt a metrikát reprezentálja.

Bizonyítás. A 13.7. tételből adódóan pontosan egy fa létezik, amely ezt a metrikát ábrázolja. Ha az algoritmusban az újonnan kiválasztott i és j csúcsokat ezen a fán csak egyetlen belső csúcs választja el, akkor egyszerű számolásból adódik, hogy a SZOMSZÉDOK-EGYESÍTÉSE algoritmus helyesen jár el. Így elég azt bizonyítani, hogy a kiválasztott i és j csúcsok mindig a megadott módon helyezkednek el.

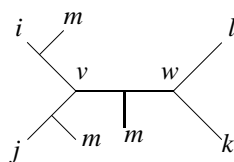
Először azt látjuk be, hogy ha i -t és j -t csak egyetlen egy belső csúcs választja el, akkor bármely k -ra $s_{i,j} < s_{i,k}$ és $s_{i,j} < s_{k,j}$. Valóban, alkalmazva a (13.86) egyenletben szereplő definíciót, az $s_{i,j} < s_{i,k}$ egyenlőtlenséget átrendezve kapjuk, hogy

$$\sum_{l \neq i,j} (d_{i,j} - d_{i,l} - d_{j,l}) - 2d_{i,j} - \sum_{m \neq j,k} (d_{j,k} - d_{j,m} - d_{k,m}) + 2d_{j,k} < 0 \quad (13.90)$$

Amennyiben $l = m \neq i, j, k$, a szummákból kapjuk, hogy

$$(d_{i,j} - d_{i,l} - d_{j,l}) - d_{j,k} + d_{j,l} + d_{k,l} = 2d_{w,l} - 2d_{u,l} < 0 \quad (13.91)$$

(lásd még 13.7. ábra). A szummán kívüli tagok, valamint a szummán belül az $l = k$ és $m = i$ esetek pedig pontosan 0-ra összegződnek, így bizonyítottuk, hogy a (13.90) egyenlőtlenség fennáll.

13.8. ábra. Az m csúcsok lehetséges elhelyezkedése a fán.

Ezután a 13.8. tételt indirekt módon bizonyítjuk. Tegyük fel, hogy az i -t és j -t nem egyetlen belső csúcs választja el, de $s_{i,j}$ minimális. A fentiekből következik, hogy sem i -hez, sem j -hez nem található olyan csúcs, melyet csak egy belső csúcs választ el i -től, illetve j -től. Keresünk olyan k és l párt, melyet csak egyetlen w belső csúcs választ el, i -t w -vel összekötő út és i -t j -vel összekötő út utolsó közös csúcsa pedig legyen v . $s_{i,j}$ minimalitásából

$$s_{k,l} - s_{i,j} > 0. \quad (13.92)$$

Ezt átrendezve kapjuk, hogy

$$\sum_{m_1 \neq k,l} (d_{k,l} - d_{m_1,k} - d_{m_1,l}) - 2d_{k,l} - \sum_{m_2 \neq i,j} (d_{i,j} - d_{m_2,i} - d_{m_2,k}) + 2d_{i,j} > 0. \quad (13.93)$$

A szummákön kívüli tagok, valamint az $m_1 = k$, $m_1 = l$, $m_2 = i$ és $m_2 = j$ esetek pontosan 0-ra összegződnek. A többi $m = m_1 = m_2 \neq i, j, k, l$ esetekben a kérdéses kifejezés

$$d_{k,l} - d_{m,k} - d_{m,l} - d_{i,j} + d_{m,i} + d_{m,k}. \quad (13.94)$$

Ha m az i -t j -vel összekötő úton kapcsolódik az i , j , k és l levelek által kifeszített részfához, akkor a (13.94) kifejezés mindig negatív lesz (lásd 13.8. ábra). Nevezzük ezen m csúcsokat I. esetnek. Ha m a v és w közötti úton kötődik be, akkor a (13.94) kifejezés lehet pozitív. Nevezzük ezen eseteket II. esetnek. Mivel a teljes kifejezésnek pozitívnek kell lennie, ezért adódik, hogy a II. esetek száma több kell, hogy legyen, mint az I. esetek száma.

Tudjuk, hogy az i -t v -vel összekötő úton van egy v' csúcs, és ebből kiindulva találunk olyan k' és l' csúcsokat, melyeket csak egyetlen w' belső csúcs választ el. Ezekre megint a II. esetek száma több kell, hogy legyen, mint az I. esetek száma, de ezzel ellentmondásra jutunk a k és l csúcsok esetével. Így i és j szomszédok kell, hogy legyenek, és ezzel bizonyítottuk a 13.8 tételt. ■

Gyakorlatok

13.5-1. Mutassuk meg, hogy ultrametrikában bármely három csúcsból származó három távolság vagy mind azonos, vagy kettő azonos, a harmadik pedig ezeknél kisebb. Bizonyítsuk be az additív metrikák esetében a tetszőleges négy csúcsból származó távolságösszegekre fennálló analóg állítást is.

13.5-2. Mutassuk meg, hogy minden ultrametrika egyben additív metrika is.

13.5-3. Adjunk példát olyan metrikára, amely nem additív.

13.5-4. Mutassuk meg, hogy minden additív metrika euklideszi.

13.5-5. Adjuk meg a pontos képletet, amely a centroid módszer esetében meghatározza $d_{u,k}$ -t $d_{i,j}$, $d_{i,k}$ és $d_{j,k}$ segítségével.

13.5-6. Mutassuk meg, hogy a csúcsok számának négyzetével arányos időben eldönthető, hogy egy metrika additív-e, illetve ultrametrikus-e.

13.6. Válogatott témák

Ebben a részben olyan témákkal foglalkozunk, amelyek általában nem szerepelnek bioinformatikai tankönyvekben, vagy csak vázlatosan vannak tárgyalva. Mi is csak a legfontosabb eredményeket említjük meg, és a tételeket nem bizonyítjuk.

13.6.1. Genomok átrendeződése

Egy organizmus genomja különböző génekből áll. A kétszálú DNS-nek csak az egyik szála a kódoló gén, a másik szál ennek a reverz komplementere. Mivel a DNS irányított, így beszélhetünk a gének irányítottságáról is. Ha minden génből egyetlen másolat található a genomban, akkor a gének sorrendje leírható egy előjeles permutációként, ahol az előjel megadja a kódoló szál irányát.

Ha adva van két genom azonos géntartalommal, előjeles permutációként ábrázolva, akkor a feladat az, hogy keressük meg azt a minimális mutációsorozatot, amely az egyik genomot a másikba transzformálja. Három mutációtípust különböztetünk meg:

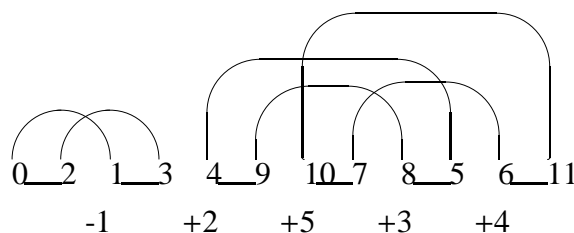
- *Inverzió.* Egy inverzió a genom egy darabját megfordítja. Az adott darabon a gének sorrendje, és a leolvasási irány, azaz az előjel is megváltozik.
- *Transzpozíció.* Egy transzpozíció a genom egy darabját egy másik helyre teszi át, úgy, hogy a gének leolvasási iránya nem változik meg.
- *Invertált transzpozíció.* Ennek hatására nem csak a genom egy darabjának a helyzete változik meg, de az elmozdított darab leolvasási iránya is megváltozik.

Ha feltesszük, hogy csak inverziók történtek, akkor meg tudunk adni egy $\Theta(n^2)$ idejű algoritmust, amely meghatároz egy olyan minimális mutációsorozatot, amely az egyik genomot a másikba transzformálja, sőt, a szükséges mutációk száma $\Theta(n)$ időben eldönthető, ahol n a gének száma.

Ha más, vagy többfajta mutációtípust veszünk figyelembe, akkor a probléma bonyolultsága nem ismert. Transzpozíciókra a legjobb közelítés egy 1.5-közelítés. Ha mind a három fajta mutációt figyelembe vesszük, akkor a legjobb eredmény egy 2-közelítő algoritmus. Ezenkívül súlyozott mutációkra létezik egy $(1 + \epsilon)$ -közelítés is, de a súlyok specialitása miatt tudjuk, hogy egy legkisebb súlyú mutációsorozatban nem lesz invertált transzpozíció.

Ha az előjeleket nem ismerjük, és csak inverziókat veszünk figyelembe, akkor a probléma bizonyítottan NP-teljes. Ugyanígy NP-teljes probléma az optimális inverziós medián megtalálása három előjeles permutáció esetében. Az optimális inverziós medián az az előjeles permutáció, melynek a három előjeles permutációtól vett távolságainak az összege minimális.

Az alábbiakban vázoljuk a két genom inverziós távolságának meghatározására szolgáló elméletet, az úgynevezett Hannenhalli–Pevzner-elméletet. Ahelyett, hogy egy π_1 permutációt transzformálnánk a π_2 permutációba, a $\pi_2^{-1}\pi_1$ -et transzformáljuk az identikus permutációba.



13.9. ábra. A $-1, +2, +5, +3, +4$ előjeles permutáció ábrázolása nem előjeles permutációként, és a permutáció töréspontgráfja.

tációba. Egyszerű csoportelméleti okoskodásból következik, hogy a két feladat egymással ekvivalens. Ezért feltesszük, hogy a két genomból a keresett $\pi_2^{-1}\pi_1$ permutációt már meghatároztuk, a továbbiakban ezt π -vel jelöljük.

Egy n elemű előjeles permutációt egy $2n$ hosszú előjel nélküli permutációval ábrázolunk a következőképpen. Minden $+i$ -t lecserélünk egy $2i - 1, 2i$ párra, minden $-i$ -t pedig egy $2i, 2i - 1$ párra. Ezenfelül az így kapott permutációt 0 és $2n + 1$ közé keretezzük. Ezután elkészítjük az úgynevezett töréspont-gráfot. Ennek csúcsai a nem előjeles permutáció elemei, beleértve a 0 -t és a $(2n + 1)$ -et is. A permutáció két elemét egy egyenes vonallal kötjük össze, ha a különbségük abszolútértéke nagyobb, mint egy. Valamint két csúcsot egy ívvel kötünk össze, ha egymás utáni számok, de a permutációban nem egymás után állnak. Egy példát adunk meg a 13.9. ábrán. Könnyen belátható, hogy a töréspont-gráfot egyértelműen fel lehet bontani körökre, a körökben az egyenes élek és ívek felváltva jönnek. Egy kört irányított-nak hívunk, ha egy, a körön megtett séta során legalább egy egyenes élen balról jobbra, és legalább egy egyenes élen jobbról balra is haladtunk. Minden más kör irányítatlan.

Két kör átfed, ha valamely íveik szükségképpen metszik egymást. A permutáció átfedési gráfjainak a csúcsai a töréspont-gráf körei, és két csúcs akkor van összekötve, ha a töréspont-gráfban a két kör metszi egymást. Az átfedési gráf komponensekre bomlik, egy komponens irányított, ha van benne irányított kör, egyébként irányítatlan. Az irányítatlan komponensek közül **nem-gátaknak** hívjuk azokat, amelyekre a töréspont-gráfon van két olyan irányítatlan komponens, amelyet az adott komponens elválaszt egymástól. Itt az elválasztáson azt értjük, hogy az egyik irányítatlan komponens valamely ívéből nem tudunk a csúcsokat összekötő vonal felett úgy átmenni a másik irányítatlan komponens valamely ívéhez, hogy ne metszenénk a nem-gát valamely ívét. A többi irányított komponenst **gátaknak** hívjuk.

A gátak közül szupergátaknak hívjuk azokat, amelyeket ha kitörlünk, akkor valamely nem-gát gáttá válik. Ez olyan esetekben fordul elő, amikor a nem-gát pontosan az adott gátat választja el más nem-irányított komponensektől. Egy permutációt erődnék hívunk, ha páratlan számú gátja van, és ezek mindegyike szupergát.

13.9. tétel. Legyen adva egy π előjeles permutáció. Inverziók egy optimális sorozata, amely ezt a permutációt rendezi,

$$b_\pi - c_\pi + h_\pi + f_\pi \quad (13.95)$$

mutációból áll, ahol b_π a töréspont-gráfban az egyenes élek száma, c_π a töréspont-gráfban a körök száma, h_π a gátak száma, és $f_\pi = 1$, ha π erőd, egyébként pedig 0 .

A tételt itt nem bizonyítjuk.

A (13.95) képletben szereplő mennyiséget meg lehet $\Theta(n)$ időben határozni, ahol n az előjeles permutáció mérete.

Nyilván b_π és c_π kiszámolható $O(n)$ időben. A nehéz rész h_π és f_π kiszámolása. A problémát az okozza, hogy az átfedési gráfban az élek száma lehet $\Omega(n^2)$. Ezért a gyors algoritmus nem határozza meg a teljes átfedési gráfot, hanem csak minden komponensén egy feszítő részét.

13.6.2. Sörétes-puska nukleinsavleolvasás

Egy genom DNS-e általában milliós nagyságrendű, vagy még több nukleinsavból áll. Egy biokémiai technikával meghatározható a DNS egyik végén található nukleinsavak sorrendje, de a leolvasási bizonytalanság növekszik, ahogy haladunk a szekvenciában előre, és kb. 500 nukleinsav után a leolvasás teljesen bizonytalanná válik.

Ezt a biokémiai problémát a következőképpen oldják meg. A DNS-ből számos kópiát vesznek, és ezek mindegyikét véletlen módon széttördelik olyan méretű részekre, melyet az előbb leírt technikával aztán már meg lehet határozni. Ezek után az átfedő részletekből kell összerakni az eredeti hosszú szekvenciát. Ezt a technikát hívjuk **sörétes-puska** nukleinsavleolvasásnak, angolul *shotgun sequencing*-nek.

Matematikailag úgy lehet definiálni a feladatot, hogy adott szekvenciáknak keressük a legrövidebb közös szuperszekvenciáját. Egy B szekvencia szuperszekvenciája A -nak, ha A részszekvenciája B -nek (részszekvencián egy szekvencia nem feltétlenül összefüggő részét értjük). Maier bebizonyította, hogy a legrövidebb szuperszekvencia NP-teljes probléma, ha az ábécé mérete legalább 5, és sejtése szerint ugyanez a helyzet a legalább háromelemű ábécék esetén is. Később megmutatták, hogy a feladat minden nem triviális ábécére NP-teljes.

Hasonló a legrövidebb közös szuperstring probléma, ami szintén NP-teljes (részstringen egy szekvencia összefüggő részét értjük). Ez utóbbi probléma az, ami igazán biológiailag érdekes, hiszen átfedő stringeket keresünk. A megoldásra számos közelítő algoritmus született. Egy mohó algoritmus minden stringpárra megkeresi a maximális átfedéseket, majd ezt próbálja mohó módon összefűzni egy legrövidebb szuperstringgá. Az algoritmus futási ideje $O(Nm)$, ahol N a szekvenciák száma, m pedig a szekvenciák összhossza. Az így megtalált szuperstring mérete bizonyítottan kisebb, mint $4n$, ahol n a legrövidebb szuperstring hossza. Egy továbbfejlesztett algoritmus bizonyítottan 3-közelítő, és a sejtés az, hogy valójában sose kapunk $2n$ -nél hosszabb szuperstringet.

A sörétes-puska nukleinsavleolvasás során a nukleinsavak meghatározása nem tökéletes, előfordulhatnak beszúrások, törlések és cserék is a meghatározás közben. Ezért Jiang és Li javasolta a legrövidebb k -közelítő közös szuperstring problémát. Kececioğlu és Myers egy programcsomagot dolgozott ki, amelyben számos heurisztikus algoritmust megvalósítottak a probléma megoldására.

Gyakorlatok

13.6-1. Mutassuk meg, hogy ha egy permutáció erőd, akkor legalább három szupergát van benne.

13.6-2. Legalább hány elemből kell egy erődnek állnia?

Feladatok

13-1. Konkáv Smith–Waterman

Adjuk meg a Smith–Waterman algoritmust konkáv résbűntetésekre.

13-2. Konkáv Spouge

Adjuk meg Spouge algoritmusát konkáv résbűntetésekre.

13-3. Kiszolgálás benzinkútnál

Egy benzinkútnál két sorban állnak a kocsik. Mindegyik kocsit vagy gázolajjal vagy benzinnel kell kiszolgálni. Egyszerre legfeljebb két kocsit szolgálhatunk ki, de csak akkor, ha a két kocsit különböző üzemanyagot igényel, és a két sor első kocsijairól van szó, vagy valamelyik sor első két kocsijáról. Akár egy, akár két kocsit szolgálunk ki egyszerre, a két folyamat kiszolgálási ideje ugyanakkora. Adjunk meg egy páros rejtett Markov-folyamatot, amelyre a Viterbi-algoritmus egy legrövidebb kiszolgálási tervet határoz meg.

13-4. Rejtett Markov-modellek momentumai

Adott egy szekvencia és egy rejtett Markov-modell. Számoljuk ki a rejtett Markov-modellben az adott szekvenciát kibocsátó utak valószínűségének a várható értékét, varianciáját, k -adik momentumát.

13-5. Sztochasztikus környezetfüggetlen nyelvtanok momentumai

Adott egy szekvencia és egy sztochasztikus környezetfüggetlen nyelvtan. Számoljuk ki az adott nyelvtanban a szekvenciát levezető levezetések valószínűségeinek a várható értékét, varianciáját, k -adik momentumát.

13-6. Rejtett Markov-modellek együttes kibocsátási valószínűsége

Ki lehet számolni ezt a valószínűséget $O((n_1 n_2)^2)$ időben?

13-7. Rendező inverziók

Egy adott előjeles permutációban rendező inverzióknak hívjuk azokat az inverziókat, amelyek kezdő lépései egy minimális hosszúságú rendező sorozatnak. Hogyan változtathatja meg egy rendező inverzió a töréspont gráfban a körök, töréspontok és a gátak számát?

Megjegyzések a fejezethez

Dinamikus programozási algoritmust biológiai szekvenciák hasonlóságára először Needleman és Wunch adott meg 1970-ben [41]. Tetszőleges résbűntető függvényre Waterman és munkatársai adtak algoritmust [57]. Gotoh algoritmus affín résbűntető függvényekre 1982-ben jelent meg [16], a konkáv résbűntető függvény ötlete Watermantól származik [56], mellyel később Miller és Myers [38], valamint Galil és Giancarlo [38] foglalkozott. Bár empirikus adatok alapján a konkáv résbűntető függvény biológiailag helyesebb, mégis a leggyakrabban az affín résbűntető függvényt használják, pl. a Clustal-W nevű népszerű szekvenciaillesztő programban is [52].

A többszörös szekvenciaillesztés ötlete Sankofftól származik [48], mely a bioinformatika egyik központi feladata lett [17]. Bebizonyították, hogy a többszörös szekvenciaillesztés NP-teljes probléma [55], így a gyakorlatban heurisztikus módszereket alkalmaznak. A legelterjedtebb heurisztika az iteratív illesztés, ez alapján működik a fent említett Clustal-W is.

Hirschberg algoritmusát először leghosszabb közös részszekvencia megkeresésére

használták [20], de mára számos bioinformatikai algoritmusban szerepel, pl. Doublescan [35]. A szekvenciaillesztés további algoritmikai elemzéséről kimerítően ír Gusfield [17].

A sztochasztikus nyelvtanok és bioinformatikai alkalmazásuk a központi témája Durbin, Eddy, Krogh és Mitchison népszerű könyvének, melyet számos egyetem bioinformatika oktatásában alapkönyvként használnak [10]. Formális nyelvekkel, nyelvtanokkal foglalkozó magyar nyelvű tankönyv Bach Iván [6] és Fülöp Zoltán [13] műve, valamint [8].

Az osztályozó algoritmusok összehasonlításáról részletesen olvashatunk Podani János könyvében [45]. A filogenetika és a filogenetikai algoritmusok iránt érdeklődők figyelmébe ajánljuk Felsenstein [12], valamint Semple és Steel [49] könyveit.

Kevéssé olvasmányos, de sok információt tartalmaz a genomátrendező algoritmusokról Pevzner könyve [43].

Stephen „String searching” című könyvében részletesen foglalkozik a legrövidebb szuperstring problémájával. A könyvben számos kiváló referenciát és az algoritmusok részletes leírásait is megtaláljuk [51].

Az alábbi megjegyzések a téma iránt kifejezetten érdeklődőknek szólnak.

Két string edit távolságán a minimális számú beszúrás-törlések számát értjük. Két string edit távolságának a kiszámolására van $\Theta(l^2)$ -nél gyorsabb módszer, amely a „négy orosz gyorsítása” néven híresült el, habár a négy egykori szovjetúnió-beli szerző közül csak egy volt orosz nemzetiségű [4]. Az algoritmus futási ideje $O(n^2 / \lg(n))$, azonban gyakorlati alkalmazásokban előforduló szekvenciahosszakra lassabb, mint a hagyományos dinamikus programozási algoritmusok.

A leghosszabb közös részszekvenciára használt dinamikus programozási algoritmus a két szekvencia hosszainak szorzatával arányos időben fut, hasonlóan a szekvenciák illesztéseire használt algoritmushoz. Hunt és Szymanski módszere ezzel szemben egy olyan gráfot állít elő, melynek csúcsai a két összehasonlítandó szekvencia, A és B karakterei, és a_i -t pontosan akkor köti össze él b_j -vel, ha $a_i = b_j$. Ezt a gráfot használva létezik olyan algoritmus, melynek futási ideje $\Theta((r+n) \lg(n))$, ahol r a gráf éleinek a száma, n pedig a gráf csúcsainak a száma, azaz a két szekvencia hossza [22]. Habár így az algoritmus futási ideje $O(n^2 \lg n)$, számos alkalmazásban a behúzott élek száma a szekvenciák hosszával egyenesen arányos. Ekkor a futási idő $O(n \lg n)$ lesz, ami lényegesen jobb a kvadrátikus idejű algoritmusnál.

A saroklevágási technika egyik fejlett változata az úgynevezett diagonális kiterjesztés. A diagonális kiterjesztés a dinamikus programozási táblázatot átlós irányban tölti ki, és nem igényel teszterteket. Ilyen algoritmusra példa Wu és munkatársainak az algoritmus [58]. A Unix diff utasításában használt algoritmus szintén diagonális kiterjesztésen alapul [37], melynek az átlagos számolási ideje $O(n + m + d_e^2)$, ahol n és m a két összehasonlítandó szekvencia hossza, d_e pedig a két szekvencia edit távolsága.

A Knuth–Morris–Pratt stringkereső algoritmus egy rövid P stringet keres meg egy hosszú M stringben, és $\Theta(p + m)$ időben fut, ahol p és m a két szekvencia hossza [27]. Landau és Vishkin módosított algoritmus [29] minden olyan illesztést megtalál M -ben, amely legfeljebb k helyen tér el P -től [29]. Az algoritmus futási ideje $\Theta(k(p \lg(p) + m))$, memóriáigénye pedig $\Theta(k(p + m))$.

Bár a szekvenciaillesztésre legelterjedtebb algoritmusok dinamikus programozással működnek, megadható szekvenciák optimális illesztése egész lineáris programozással is. Az ötlet Kececioğlu és munkatársaitól származik [25]. A módszert kiterjesztették tetszőleges részbüntető függvényre is [3]. Az egész értékű programozással kapcsolatos algoritmusokról írt áttekintő cikket Lancia [28]. A DiAlign szintén nem dinamikus programozáson

alapul [39].

A szekvenciák strukturális illesztése csak a fehérjék három dimenziós szerkezetét veszi figyelembe. Olyan szekvenciaillesztést keresünk, melyben a réseket büntetjük, az összeillesztett karaktereket viszont nem hasonlóság vagy távolság alapján értékeljük, hanem az alapján, hogy a három dimenziós térszerkezetben az összeillesztett aminosavak mennyire hozhatóak fedésbe a szerkezetek forgatásával. Számos algoritmust dolgoztak ki strukturális szekvenciaillesztésre, ezek közül az egyik legnépszerűbb a Kombinatorikus Kiterjesztés (CE) algoritmus [50].

Egy adott fa topológiára a Maximum Likelihood címkézés polinomiális időben megoldható [46]. Ez az algoritmus az egyik legelterjedtebb filogenetikai elemző csomagba, a PAML-ba is bekerült (<http://abacus.gene.ucl.ac.uk/software/paml.html>).

Egy adott fa topológia, élhosszak, szubsztitúciós modell és adott szekvenciák esetén lineáris időben ki lehet számolni egy fa likelihoodját Felsenstein algoritmusával. A Maximum Likelihood fa problémája az, hogy adott modell és szekvenciák esetén határozzuk meg azt a fa topológiát és élhosszakot, amelyre a likelihood maximális. Meglepő módon, még senkinek sem sikerült bizonyítani, hogy ez a probléma NP-teljes lenne, bár ez a sejtés. Azt már sikerült bizonyítani, hogy az Ancestral Maximum Likelihood probléma, amikor nem csak a fa topológiáját és élhosszait, hanem a belső csúcsok legvalószínűbb címkézését is keressük, NP-teljes [1].

A két legelterjedtebb, rejtett Markov-modellek alapján működő szekvenciaillesztő programcsomag a SAM [21] és a HMMER (<http://hmmer.wustl.edu/>). Rejtett Markov-modell alapján működő genomannotáló programot fejlesztett ki Pedersen és Hein [42], páros rejtett Markov-modellt használ szintén genomannotálásra a Double-Scan [35], (<http://www.sanger.ac.uk/Software/analysis/doublescan/>) és a Projector [36], (<http://www.sanger.ac.uk/Software/analysis/projector/>).

Olyan rejtett Markov-modellt, amely evolúciós információk alapján határozza meg a kibocsátási valószínűségeket, Goldman, Thorne és Jones publikált először [15], fehérjék másodlagos térszerkezetének jóslására. Ez a rejtett Markov-modell szekvenciák illesztett oszlopait bocsátja ki, egy illesztett oszlop kibocsátási valószínűségét egy evolúciós fa és egy időfolytonos Markov-modell határozza meg. A kibocsátási valószínűséget Felsenstein algoritmusával lehet meghatározni.

A Knudsen-Hein nyelvtant használja a PFold nevű program, amely RNS-ek másodlagos térszerkezetét határozza meg [26]. Ez a sztochasztikus környezetfüggetlen nyelvtan szintén illesztett szekvenciákat vezet le, a terminális szimbólumok ezen szekvenciaillesztés illesztett oszlopai. Az s terminális levezetési valószínűségét egy evolúciós törzsfá és egy időfolytonos Markov-modell határozza meg, a dFd levezetésben a két d terminális helyére írandó két oszlop valószínűségét pedig ugyanezen a törzsfán egy olyan időfolytonos Markov-modell adja meg, melynek állapotai dinukleotidok.

Az ELŐRE algoritmus futási ideje négyzetesen növekszik a rejtett Markov-modell állapotainak a számával. Azonban nem mindig ez a leghatékonyabb algoritmus. Egy biológiailag fontos többszörös rejtett Markov-modellben az ELŐRE algoritmus futási ideje $\Theta(5^n L^n)$, ahol n a szekvenciák száma, L pedig a szekvenciák hosszának geometriai közepe. Meg lehet adni azonban egy olyan algoritmust, amely $\Theta(2^n L^n)$ időben kiszámolja a szekvenciák kibocsátási valószínűségét [30, 31]. Nem ismert azonban, hogy erre a modellre a legvalószínűbb kibocsátás megkeresésére hasonló gyorsítás megoldható-e.

Az RNS-ek Zuker-Tinoco [53] modellje térszerkezeti elemeken definiál szabadener-

giát, és egy adott másodlagos térszerkezethez a térszerkezet részeihez rendelt szabadenergiák összegét rendeli. A Zuker–Sankoff-algoritmus [59] $\Theta(l^4)$ időben találja meg a legkisebb energiataralmú másodlagos térszerkezetet, $\Theta(l^2)$ memóriát használva, ahol l az RNS szekvencia hossza. Ugyanilyen memória- és számolásigényű a térszerkezetek Boltzman-eloszlásához tartozó partíciófüggvény kiszámolása is [34]. Egy speciális esetben mind az optimális térszerkezet, mind a partíciófüggvény számolása felgyorsítható $\Theta(l^3)$ időre is, továbbra is $\Theta(l^2)$ memóriát használva [33].

Az RNS-ek azon bázispárosodásait, melyben a párosodó nukleinsavakat összekötő, a szekvencia felett haladó ívek metszik egymást, álcsomóknak hívjuk. Az álcsomókat tartalmazó térszerkezetek általános predikciója NP-teljes. Azonban bizonyos speciális álcso-mók közül egy optimális álcso-mó megkeresésére vannak polinomiális idejű algoritmusok [2, 32, 47, 54].

Rejtett Markov-modellek és sztochasztikus környezetfüggetlen nyelvtanok együttes kibocsátási valószínűségeinek meghatározásával foglalkoztak Jagota és munkatársai [23]. A keresett valószínűséget egy kvadratikus egyenletrendszer megoldásával lehet meghatározni, amelyet csak numerikusan lehet megoldani. Az egyenletrendszerben az ismeretlenek száma Vn^2 , ahol V a nemterminálisok száma Chomsky féle normálformában, n pedig a rejtett Markov-modell állapotai. A szerzők megadtak egy iterációt, mely bizonyítottan konvergál a pontos megoldáshoz, egy iterációs lépés számolási igénye $\Theta(V^3n^5)$. Az iteráció konvergenciájának a sebességéről nem szolgálnak analitikus eredménnyel, de azt sejtik, hogy gyors.

Sztochasztikus környezetfüggetlen nyelvtanok generátorfüggvényeit használta RNS térszerkezetek predikcióinak jóságára Markus Nebel. Számos olyan statisztikát sikerült analitikusan kiszámolnia, amelyek korrelálnak a predikció jóságával [40].

Rendezett fákhhoz hasonlóan rendezett erdőket is lehet illeszteni. Rendezett erdők illesztésére adtak meg algoritmust Höchsmann és munkatársai. Megmutatták, hogy RNS térszerkezeteket lehet rendezett erdővel ábrázolni, és ezen ábrázoláson keresztül az RNS térszerkezetek összehasonlíthatóak [19].

Atteson matematikai definíciót adott a törzsfakészítő módszerek jóságára, és kimutatta, hogy bizonyos definíciók alapján a SZOMSZÉDOK EGYESÍTÉSE a lehető legjobb módszer [5].

Négy fajra három fa topológiát lehet készíteni, amelyeket quarteteknek hívunk. Ha egy fára ismerjük az összes quartet topológiát, akkor ebből az eredeti fa topológiáját is meg lehet határozni. Sőt bebizonyították, hogy nem szükséges az összes quartetet megnézni, a közeli fajok quartetjei is egyértelműen meghatározzák a fa topológiáját [11].

Egy genom állhat több DNS szekvenciából, az egyes DNS szekvenciákat kromoszómáknak hívjuk. Genomátrendeződés során nem csak kromoszómán belül, hanem a kromoszómák között is keveredhetnek a gének. Ezekre az úgynevezett transzlokációs mutációkra adott meg $\Theta(n^3)$ idejű algoritmust Hannenhalli [18]. Transzlokációs átmérővel és a probléma általánosításával foglalkozik Pisanti és Sagot [44].

A permutációk rendezésének általánosított problémája a minimális generáló szó keresése egy véges csoportban. Ez a probléma bizonyítottan NP-teljes [24]. Az előjeles permutációk inverziókkal való rendezésén és a transzlokációs távolságon kívül csak egy biológiailag kevésbé releváns problémára van polinomiális idejű algoritmus, az úgynevezett blokk átrendeződésekre [7]. Érdekességként megjegyezzük, hogy a Microsoft tulajdonosa, Bill Gates is foglalkozott permutációk rendezésével, speciálisan a prefix inverziókkal [14].

A könyvfejezetben csak a legfontosabb témákról írtunk, számos kevésbé fontos, de nagyon érdekes témakört kénytelenek voltunk kihagyni. Ilyen témakörök például a rekombi-

náció, a pedigré elemzés, a karakteralapú törzsfakészítő módszerek, a részleges emésztés, a fehérjecsavarás, DNS chipek, a tudásábrázolás, a biokémiai hálózatok. Ennek fényében Donald Knuth szavaival [9] zárjuk könyvfejezetünket: „It is hard for me to say confidently that, after fifty more years of explosive growth of computer science, there will still be a lot of fascinating unsolved problems at peoples’ fingertips, that it won’t be pretty much working on refinements of well-explored things. Maybe all of the simple stuff and the really great stuff has been discovered. It may not be true, but I can’t predict an unending growth. I can’t be as confident about computer science as I can about biology. Biology easily has 500 years of exciting problems to work on, it’s at that level.”

Irodalomjegyzék

- [1] L. Addario-Berry, B. Chor, M. Hallett, J. Lagergren, A. Panconesi, T. Wareham. Ancestral maximum likelihood of phylogenetic trees is hard. *Lecture Notes in Bioinformatics*, 2812:202–215, 2003. [585](#)
- [2] T. Akutsu. Dynamic programming algorithms for RNA secondary prediction with pseudoknots. *Discrete Applied Mathematics*, 104:45–62, 2000. [586](#)
- [3] E. Althaus, A. Caprara, H. Lenhof, K. Reinert. Multiple sequence alignment with arbitrary gap costs: Computing an optimal solution using polyhedral combinatorics. *Bioinformatics*, 18:S4–S16, 2002. [584](#)
- [4] V. [Arlazanovan](#), A. Dinic, M. Kronrod, I. Faradzev. On economic construction of the transitive closure of a directed graph. *Doklady Akademii Nauk SSSR*, 194:487–488, 1970. [584](#)
- [5] K. Atteson. The performance of the neighbor-joining method of phylogeny reconstruction. *Algorithmica*, 25(2/3):251–278, 1999. [586](#)
- [6] I. Bach. *Formális nyelvek (Formal languages)*. [Typotex](#), 2001. [584](#)
- [7] D. Christie. Sorting permutations by block-interchanges. *Information Processing Letters*, 60(4):165–169, 1996. [586](#)
- [8] T. H. [Cormen](#), C. E. [Leiserson](#), R. L. [Rivest](#), C. [Stein](#). *Introduction to Algorithms*. The [MIT Press/McGraw-Hill](#), 2003 (Második kiadás negyedik, javított utánnomása. Magyarul: [Új algoritmusok](#). [Scolar](#) Kiadó, 2003). [584](#)
- [9] D.. Doernberg. Interview with Donald Knuth. *Computer Literacy*, 1993. [587](#)
- [10] R. Durbin, S. Eddy, A. Krogh, G. Mitchison. *Biological Sequence Analysis*. University Press, 1998. [584](#)
- [11] P. [Erdős](#), M. Steel, L. [Székely](#), T. Warnow. Local quartet splits of a binary tree infer all quartet splits via one dyadic inference rule. *Computers and Artificial Intelligence*, 16(2):217–227, 1997. [586](#)
- [12] J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Inc., 2003. [584](#)
- [13] Z. [Fülöp](#). *Formális nyelvek és szintaktikus elemzésük (Formal Languages and their Syntactical Analysis)*. [Polygon](#), 2000. [584](#)
- [14] W. H. [Gates](#), C. Papadimitriou. Bounds for sorting by prefix reversals. *Discrete Mathematics*, 27:47–57, 1979. [586](#)
- [15] N. Goldman, J. Thorne, D. Jones. Using [evolutionary](#) trees in protein secondary structure prediction and other comparative sequence analyses. *Journal of Molecular Biology*, 263(2):196–208, 1996. [585](#)
- [16] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162:705–708, 1982. [583](#)
- [17] D. M. [Gusfield](#). *Algorithms on Strings, Trees and Sequences*. [Cambridge](#) University Press, 1997. [583](#), [584](#)
- [18] S. Hannenhalli. Polynomial-time algorithm for computing translocation distance between genomes. *Discrete Applied Mathematics*, 71:137–151, 1996. [586](#)
- [19] M. Höchsmann, T. Töller, R. Giegerich, S. Kurtz. Local similarity in RNA secondary structure. In *Proceedings of IEEE Bioinformatics Conference 2003*, 159–168. o., 2003. [586](#)
- [20] D. S. [Hirschberg](#). A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18:341–343, 1975. [584](#)
- [21] R. Hughey, A. Krogh. Hidden markov models for sequence analysis: Extension and analysis of the basic method. *CABIOS*, 12(2):95–107, 1996. [585](#)
- [22] J. Hunt, T. Szymanski. A fast algorithm for computing longest common subsequences. *Communications of the ACM*, 20(5):350–353, 1977. [584](#)

- [23] A. Jagota, R. B. Lyngso, C. N. S. Pedersen. Comparing a hidden Markov model and a stochastic context-free grammar. *Lecture Notes in Computer Science* 2149. kötet, 69–84. o. Springer-Verlag, 2001. [586](#)
- [24] M. R. Jerrum. The complexity of finding minimum-length generator sequences. *Theoretical Computer Science*, 36:265–289, 1986. [586](#)
- [25] J. Kececioğlu, H. Lenhof, K. Mehlhorn, P. Mutzel, K. Reinert, M. Vingron. A polyhedral approach to sequence alignment problems. *Discrete Applied Mathematics*, 104((1-3)):143–186, 2000. [584](#)
- [26] B. Knudsen, J. Hein. Pfold: RNA secondary structure prediction using stochastic context-free grammars. *Nucleic Acids Research*, 31(13):3423–3428, 2003. [585](#)
- [27] D. E. Knuth, J. H. Morris Jr., V. R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977. [584](#)
- [28] G. Lancia. Integer programming models for computational biology problems. *Journal of Computer Science and Technology*, 19(1):60–77, 2004. [584](#)
- [29] G. Landau, U. Vishkin. Efficient string matching with k mismatches. *Theoretical Computer Science*, 43:239–249, 1986. [584](#)
- [30] G. Lunter, I. Miklós, A. Drummond, J. L. Jensen, J. Hein. Bayesian phylogenetic inference under a statistical indel model. *Lecture Notes in Bioinformatics*, 2812:228–244, 2003. [585](#)
- [31] G. Lunter, I. Miklós, Y., J. Hein. An efficient algorithm for statistical multiple alignment on arbitrary phylogenetic trees. *Journal of Comp. Biology*, 10(6):869–889, 2003. [585](#)
- [32] R. Lyngso, C. N. S. Pedersen. RNA pseudoknot prediction in energy based models. *Journal of Comp. Biology*, 7(3/4):409–428, 2000. [586](#)
- [33] R. Lyngso, M. Zuker, C. Pedersen. Fast evaluation of internal loops in RNA secondary structure prediction. *Bioinformatics*, 15(6):440–445, 1999. [586](#)
- [34] J. S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29:1105–1119, 1990. [586](#)
- [35] I. Meyer, R. Durbin. Comparative ab initio prediction of gene structures using pair HMMs. *Bioinformatics*, 18(10):1309–1318, 2002. [584](#), [585](#)
- [36] I. M. Meyer, R. Durbin. Gene structure conservation aids similarity based gene prediction. *Nucleic Acids Research*, 32(2):776–783, 2004. [585](#)
- [37] W. Miller, E. Myers. A file comparison program. *Software – Practice and Experience*, 15(11):1025–1040, 1985. [584](#)
- [38] W. Miller, E. W. Myers. Sequence comparison with concave weighting functions. *Bulletin of Mathematical Biology*, 50:97–120, 1988. [583](#)
- [39] B. Morgenstern. DIALIGN 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, 15:211–218, 1999. [585](#)
- [40] M. Nebel. Identifying good predictions of rna secondary structure. In R. B. Altman, A. K. Dunker, L. Hunter, T. E. Klein (szerkesztők), *Pacific Symposium on Biocomputing*, 9. kötet, 423–434. o. PSB Online, <http://psb.stanford.edu/psb-online/>, 2004. [586](#)
- [41] S. N. Needleman, C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970. [583](#)
- [42] J. S. Pedersen, J. Hein. Gene finding with a hidden Markov model of genome structure and evolution. *Bioinformatics*, 19(2):219–227, 2003. [585](#)
- [43] P. A. Pevzner. *Computational Molecular Biology: An Algorithmic Approach*. The MIT Press, 2000. [584](#)
- [44] N. Pisanti, M. Sagot. Further thoughts on the syntenic distance between genomes. *Algorithmica*, 34(2):157–180, 2002. [586](#)
- [45] J. Podani. *Bevezetés a többváltozós biológiai adatfeldarés rejtelmeibe*. Scientia, 1997. [584](#)
- [46] T. Pupko, I. Peer, R. Shamir, D. Graur. A fast algorithm for joint reconstruction of ancestral amino acid sequences. *Molecular Biology and Evolution*, 17:890–896, 2000. [585](#)
- [47] E. Rivas, S. Eddy. A dynamic programming algorithm for RNA structure prediction including pseudoknots. *Journal of Molecular Biology*, 285(5):2053–2068, 1999. [586](#)
- [48] D. Sankoff. Minimal mutation trees of sequences. *SIAM Journal of Applied Mathematics*, 28:35–42, 1975. [583](#)
- [49] C. Semple, M. Steel. *Phylogenetics*. Number 24 in Oxford Lecture Series in Mathematics and Its Applications. Oxford Press, 2003. [584](#)
- [50] I. Shindyalov, P. Bourne. Protein structure alignment by incremental combinatorial extension (CE) of the optimal path. *Protein Engineering*, 11(9):739–747, 1998. [585](#)

- [51] G. Stephen. *String searching algorithms, Lecture Notes Series on Computing* 3. kötet. World Scientific, Singapore, 1994. [584](#)
- [52] J. D. Thompson, D. G. Higgins, T. J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 1994. [583](#)
- [53] I. Tinoco, O., Uhlenbeck M. Levine. Estimation of secondary structure in ribonucleic acids. *Nature*, 230:362–367, 1971. [585](#)
- [54] Y. Uemura, A. Hasegawa, Y. Kobayashi, T. Yokomori. Tree adjoining grammars for RNA structure prediction. *Theoretical Computer Science*, 210:277–303, 1999. [586](#)
- [55] L. Wang, T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1:337–348, 1994. [583](#)
- [56] M. S. [Waterman](#). Efficient sequence alignment algorithms. *Journal of Theoretical Biology*, 108:333–337, 1984. [583](#)
- [57] M. S. [Waterman](#), T. F. Smithand, W. A. Beyer. Some biological sequence metrics. *Advances in Mathematics*, 20:367–387, 1976. [583](#)
- [58] S. Wu, E.. W. Myers, U. Manber, W. Miller. An $O(NP)$ sequence comparison algorithm. *Information Processing Letters*, 35(6):317–323, 1990. [584](#)
- [59] M. Zuker, D. Sankoff. RNA structures and their prediction. *Bulletin of Mathematical Biology*, 46:591–621, 1986. [586](#)

Névmutató

A, Á

Addario-Berry, L., [588](#)
Akutsu, T., [588](#)
Althaus, E., [588](#)
Arlazarov, V. L., [588](#)
Atteson, K., [588](#)

B

Bach Iván, [584](#), [588](#)
Beyer, W. A., [590](#)
†Boltzman, Ludwig Eduard, [586](#)
Bourne, P. E., [589](#)

C

Caprara, A., [588](#)
Carrillo, H., [559](#)
Chomsky, N., [565](#)
Chor, B., [588](#)
Christie, D. A., [588](#)
Cocke, J., [565](#)
Cormen, Thomas H., [588](#)

D

Dinic, E., [588](#)
Doernberg, Dan, [588](#)
Drummond, A., [589](#)
Durbin, Richard, [584](#), [588](#), [589](#)

E, É

Eddy, Sean, [584](#), [588](#), [589](#)
†Erdős Pál, [588](#)

F

Faradzev, I. A., [588](#)
Felsenstein, Joseph, [561](#), [584](#), [585](#), [588](#)
Fickett, J.W., [558](#)
Fitch, J., [562](#)
Fülöp Zoltán, [584](#), [588](#)

G

Gates, William Henry, [586](#), [588](#)
Gibson, T. J., [590](#)
Giegerich, R., [588](#)
Goldman, N., [585](#), [588](#)

Gotoh, O., [588](#)
Graur, D., [589](#)
Gusfi eld, Daniel M., [588](#)

H

Hallett, M., [588](#)
Hannenhalli, S., [580](#), [588](#)
Hasegawa, A., [590](#)
Hein, J., [567](#), [585](#), [589](#)
Higgins, D. G., [590](#)
Hirschberg, D. S., [556](#), [588](#)
Höchsman, M., [588](#)
Hughey, R., [588](#)
Hunt, J. W., [588](#)

J

Jagota, A., [586](#), [589](#)
Jensen, J. L., [589](#)
Jerrum, M. R., [589](#)
Jiang, T., [590](#)
Jones, D. T., [588](#)

K

Kasami, ???, [565](#)
Kececioğlu, J. D., [582](#), [584](#), [589](#)
Knudsen, B., [567](#), [589](#)
Knuth, Donald Erwin, [584](#), [587](#), [589](#)
Kobayashi, Y., [590](#)
Krogh, Anders, [584](#), [588](#)
Kronrod, M. A., [588](#)
Kurtz, S., [588](#)

L

Lagergren, J., [588](#)
Lancia, G., [584](#), [589](#)
Landau, G. M., [584](#), [589](#)
Leiserson, Charles E., [588](#)
Lenhof, H. P., [588](#), [589](#)
Levine, M. D., [590](#)
Lipman, D., [559](#)
Lunter, G. A., [589](#)
Lyngso, R., [589](#)

M

Manber, U., [590](#)

Markov, Andrej Andrejevics, [583](#)
McCaskill, J. S., [589](#)
Mehlhorn, K., [589](#)
Meyer, I. M., [589](#)
Miklós István, [589](#)
Miller, W., [589](#), [590](#)
Mitchison, Graeme, [584](#), [588](#)
Morgenstern, B., [589](#)
Morris, James H., [584](#)
Mutzel, P., [589](#)
Myers, E. W., [589](#), [590](#)

N

Nebel, M. E., [589](#)
Needleman, S. B., [589](#)

P

Panconesi, A., [588](#)
Papadimitrou, Christos H., [588](#)
Pedersen, C. N. S., [589](#)
Pedersen, J. S., [585](#), [589](#)
Peer, I., [589](#)
Pevzner, Pavel, [580](#), [584](#)
Pisanti, N., [589](#)
Podani János, [584](#), [589](#)
Pratt, Vaughan R., [584](#)
Pupko, T., [589](#)

R

Reinert, K., [588](#), [589](#)
Rivas, E., [589](#)
Rivest, Ronald Lewis, [588](#)

S

Sagot, M. F., [589](#)
Sankoff, D., [586](#), [589](#), [590](#)
Semple, Charles, [584](#), [589](#)
Shamir, R., [589](#)
Shindyalov, I. N., [589](#)
Smithand, T. F., [590](#)
Song, Y. S., [589](#)
Spouge, J.L., [558](#)

Steel, Mike, [584](#), [588](#), [589](#)
Stein, Clifford, [588](#)
Stephen, Graham A., [584](#), [590](#)

SZ

Székely László, [588](#)
Szymanski, T. G., [588](#)

T

Thompson, J. D., [590](#)
Thorne, J. L., [588](#)
Tinoco, I., [585](#), [590](#)
Töller, T., [588](#)

U, Ú

Uemura, Y., [590](#)
Uhlenbeck, Ö. C., [590](#)
Ukkonen, Esko, [558](#)

V

Vingron, M., [589](#)
Vishkin, U., [584](#), [589](#)
Viterbi, A., [564](#), [565](#)

W

Wang, L., [590](#)
Wareham, T., [588](#)
Warnow, T., [588](#)
Waterman, Michael S., [590](#)
Wu, S., [590](#)
Wunsch, C.D., [589](#)

Y

Yokomori, T., [590](#)
Younger, ???, [565](#)

Z

Zuker, M., [585](#), [589](#), [590](#)

Tárgymutató

A, Á

additív metrika, [574](#)
affin függvény, [550](#)
átló, [556](#)

B

BACKWARD, [564](#)
BELÜLRŐL, [565](#)
BLOSUM, [554](#)

C

CENTROID, [573](#)
Chomsky-féle normálforma, [565](#)
Clustal-W, [550](#), [583](#)
CYK-algoritmus, [565](#)

CS

csendes állapot, [567](#)
CSOPORTÁTLAG, [573](#)

D

DiAlign, [556](#)
DiAlign, [584](#)
DNS, [546](#)

E, É

EGYSZERŰ-ÁTLAG, [573](#)
EGYSZERŰ-LÁNC, [573](#)
ELŐLRŐL, [565](#)
ELŐRETEKINTŐ, [551](#)
ELŐRETEKINTŐ-BINÁRISKERESŐ, [553](#)

F

Fickett algoritmus, [558](#)
Fitch-algoritmus, [562](#)_{gy}
FORWARD, [564](#)

G

gát, [581](#)
Gotoh-algoritmus, [550](#)

H

Hirschberg algoritmus, [557](#)
HMMER, [585](#)

I, Í

illesztett pár, [548](#)

K

KÍVÜLRŐL, [565](#)
Knudsen-Hein-nyelvtan, [567](#)_{gy}
kompozíció, [547](#)
konkáv résbűntető függvény, [551](#)

L

log-odds, [554](#)

M

Markov-modell, [583](#)_{fe}
MEDIÁN, [574](#)
minimális törzsfajlás, [546](#)
molekuláris óra, [574](#)

N

négycsúcs metrika, [574](#)
nem-gát, [581](#)

P

PAM, [551](#), [554](#)
PAML, [585](#)
páronkénti összeg, [559](#)
páros rejtett Markov-modell, [567](#)
PFold, [585](#)
Projector, [585](#)

R

rejtett Markov-folyamat, [563](#)
rejtett Markov-modell, [585](#)
rendezett fa, [570](#)
rés, [549](#)
résbűntetés, [549](#)
részstring, [554](#)

S

sörétes-puska nukleinsavleolvasás, [582](#)
Spouge algoritmus, [558](#)
súlyfüggvény, [547](#)

SZ
szekvencia, [546](#)

T
TELJES-LÁNC, [573](#)

U, Ú
Ukkonen algoritmus, [558](#)
ultrametrika, [572](#)

V
vezérfa, [555](#)
Viterbi-algoritmus, [565](#)

Tartalomjegyzék

VI. ALKALMAZÁSOK	545
13. Bioinformatika (Miklós István)	546
13.1. Algoritmusok szekvenciákon	546
13.1.1. Két szekvencia távolsága lineáris részbüntetés mellett	546
13.1.2. Dinamikus programozás tetszőleges részbüntetés mellett	549
13.1.3. Gotoh algoritmus affín részbüntetéssel	550
13.1.4. Konkáv részbüntetés	550
13.1.5. Két szekvencia hasonlósága, Smith–Waterman algoritmus	554
13.1.6. Többszörös szekvenciaillesztés	554
13.1.7. Memóriaredukció Hirschberg algoritmusával	556
13.1.8. Memóriaredukció saroklevágással	557
13.2. Algoritmusok fákon	560
13.2.1. A takarékosági elv kis problémája	560
13.2.2. Felsenstein algoritmus	561
13.3. Algoritmusok sztochasztikus nyelvtanokon	562
13.3.1. Rejtett Markov-modellek: előre, hátra és Viterbi algoritmus	563
13.3.2. Sztochasztikus környezetfüggetlen nyelvtanok: belülről, kívülről és a CYK algoritmus	565
13.4. Szerkezetek összehasonlítása	568
13.4.1. Címkézett, gyökeres fák illesztése	568
13.4.2. Két rejtett Markov-modell együttes kibocsátási valószínűsége	569
13.5. Törzsfakészítés távolságon alapuló algoritmusokkal	570
13.5.1. Osztályozó algoritmusok	572
13.5.2. Szomszédok egyesítése	574
13.6. Válogatott témák	580
13.6.1. Genomok átrendeződése	580
13.6.2. Sörétes-puska nukleinsavleolvasás	582
Irodalomjegyzék	588
Névmutató	591
Tárgymutató	593