7 SAROKVÁGÁS TESZTÉRTÉK NÉLKÜL

7.1 Problémafelvetés

A tesztértékek segítségével történő szekvencia összehasonlító algoritmusok rendkívül praktikusak akkor, amikor adatbázisokból kell kikeresni egy adott szekvenciával nagyfokú rokonságot mutató szekvenciákat (Spouge, 1991). Ezek az algoritmusok kevésbé hatékonyak akkor, ha szükségképpen meg kell állapítani a két szekvencia közötti hasonlóságot, például, ha a szekvenciák evolúciós leszármazási viszonyaira vagyunk kíváncsiak.

Statisztikus szekvencia analízis esetén nem egyetlen 'optimális' illesztést keresünk, hanem az illesztések összességéből próbálunk következtetéseket levonni az adott szekvenciákra. A teszt érték segítségével történő sarokvágási algoritmusok ebben a problémakörben tehát közvetlenül nem alkalmazhatóak. Hein és munkatársai algoritmusában (Hein et al, 2000) az ε érték tölt be a tesztértékhez hasonló funkciót (ld. 4.7 fejezet). Alacsony ε érték esetében a likelihood számítás pontatlan lesz, ezzel párhuzamosan a metódus pontatlan becsléseket ad a maximum likelihood paraméterértékekre is.

Ebben a fejezetben bemutatom, hogy hogyan lehet a diagonális kiterjesztést (Myers, 1986; Wu et al., 1990) felhasználni tesztérték nélküli sarokvágási algoritmusok konstruálására.

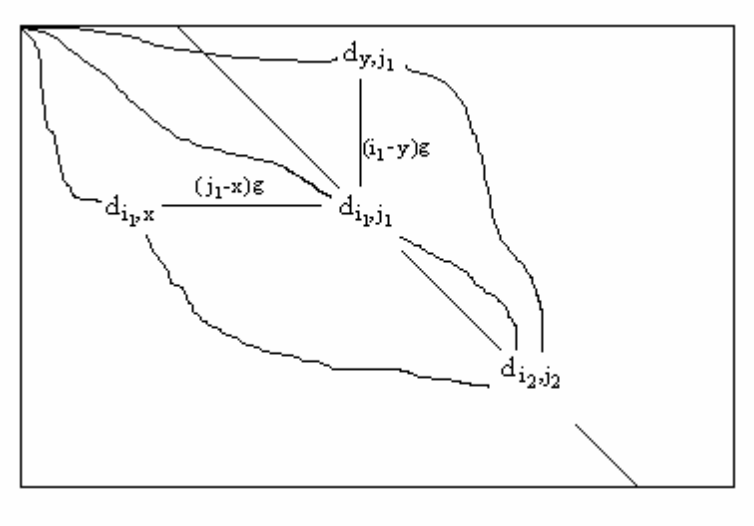
7.2 Sarokvágás többszörös indelek engedélyezése nélkül

Ebben a fejezetben azt mutatom be, hogy hogyan lehet a minimális súlyú illesztést gyorsan megkeresni tesztérték nélkül. Az algoritmus ötletét a lehető legegyszerűbb súlyfüggvényen mutatom be, a bonyolultabb eseteket és a statisztikus szekvencia analízisben való alkalmazás lehetőségét a következő fejezetekre hagyom.

Az egyszerűség kedvéért az algoritmus csak kétféle súllyal dolgozik. Két tetszőleges, de nem azonos illesztett karakter súlyát *mis*-sel jelölöm, egy karakter beszúrását vagy törlését pedig *g* értékkel büntetem. Először a *k*-átló fogalmát definiálom.

DEFINÍCIÓ A **D** mátrix *k*-átlója a mátrix azon $d_{i,j}$ elemeinek a halmaza, amelyekre $i-j=k$. A *k*-átló indexe *k*.

Bármely *k*-átló tehát a mátrix főátlójával párhuzamos, a főátló a 0-átló. Az *A* és *B* szekvenciák összehasonlítása esetén, amelyek rendre *n* és *m* hosszúak, a dinamikus programozási táblázatban az átlók indexei -*m*-től *n*-ig terjednek. Az algoritmus ötlete a következő tételben rejlik.



7.2.1 ábra Az A_{i_2} és B_{j_2} szekvenciák optimális illesztését ábrázoló utak három lehetséges esete. Az optimális út vagy átmegy d_{i_2,j_2} átlójának egy rögzített elemén, vagy az optimális út ezen rögzített d_{i_1,j_1} elem előtt vagy fölött halad. Mindhárom esetben bizonyítható, hogy $d_{i_1,j_1} \le d_{i_2,j_2}$.

LEMMA7.2.1 A dinamikus programozási táblázat bármely átlójában az értékek monoton növekszenek felülről lefelé.

BIZONYÍTÁS Legyen d_{i_1,j_1} és d_{i_2,j_2} a **D** mátrix két eleme, $i_1<i_2$, és ekképpen $j_1<j_2$. A következő három eset lehetséges (ld. **7.2.1** ábra):

- Az A_{i_2} és B_{j_2} szekvenciák optimális illesztését reprezentáló út tartalmazza d_{i_1,j_1}-t. Mivel bármely optimális illesztést tartalmazó p_1, p_2, … p_n útra $p_k \le p_{k+1}$, ezért nyilvánvalóan $d_{i_1,j_1} \le d_{i_2,j_2}$.
- Az A_{i_2} és B_{j_2} szekvenciák optimális illesztését reprezentáló út tartalmazza $d_{i_1,x}$-t, ahol $x<j_1$. A $d_{i_1,x}$-től d_{i_2,j_2}-ig terjedő út súlya legalább $(j_1-x)g$, ezért $d_{i_1,x}+(j_1-x)g \le d_{i_2,j_2}$. Mivel A_{i_2} és B_{j_2} szekvenciák optimális illesztése nem szükségképpen megy át $d_{i_1,x}$-en, ezért $d_{i_1,j_1} \le d_{i_1,x}+(j_1-x)g$. A két egyenlőtlenséget összekapcsolva kapjuk, hogy $d_{i_1,j_1} \le d_{i_1,x}+(j_1-x)g \le d_{i_2,j_2}$
- Az A_{i_2} és B_{j_2} szekvenciák optimális illesztését reprezentáló út tartalmazza d_{y,j_1}-t, ahol $y<i_1$. Az előző esethez hasonlóan $d_{i_1,j_1} \le d_{y,j_1}+(i_1-y)g \le d_{i_2,j_2}$.

Az algoritmusnak szüksége van két egydimenziós tömbre. Az egyik tömb minden egyes átlóra tartalmazza az utolsó kiszámított elem első koordinátáját, ezt *pos*(*k*) jelöli, a másik tömb pedig az utolsó kiszámított elem értékét tartalmazza minden átlóra, ezt *val*(*k*) jelöli.

Az algoritmus részletes leírása a következő (A I., II., stb. számok a **7.2.2** ábrán szereplő számokra utalnak):

I. Az algoritmus a **D** mátrixot átlók szerint tölti ki. Az algoritmus egymás után számolja ki egy átló értékeit, figyelembe nem véve a szomszédos átlók ki nem számított elemeit, és addig halad az átlón, amíg az így kiszámított értékek (jelölése *newval*) jósága kétségtelen.

LEMMA 7.2.2 Tegyük fel, hogy *newval* jelölt a *k* átló $d_{i,j}$ elemére. Ha $val(k\text{-}1)+g \geq newval$ és $val(k+1)+g \geq newval$, akkor $neval = d_{i,,j}$.

BIZONYÍTÁS $newval \neq d_{i,,j}$, ha $d_{i\text{-}1,,j}$-t még nem számolta ki az algoritmus, és $newval > d_{i\text{-}1,,j}+g$. De ha $d_{i\text{-}1,,j}$-t még nem számolta ki az algoritmus, akkor a **7.2.1** lemmából adódóan $val(k\text{-}1) \leq d_{i\text{-}1,,j}$, így ha $newval \leq val(k\text{-}1)+g$, akkor $newval \leq d_{i\text{-}1,,j}+g$. Ugyanez igaz $d_{i,,j\text{-}1}$-re is.

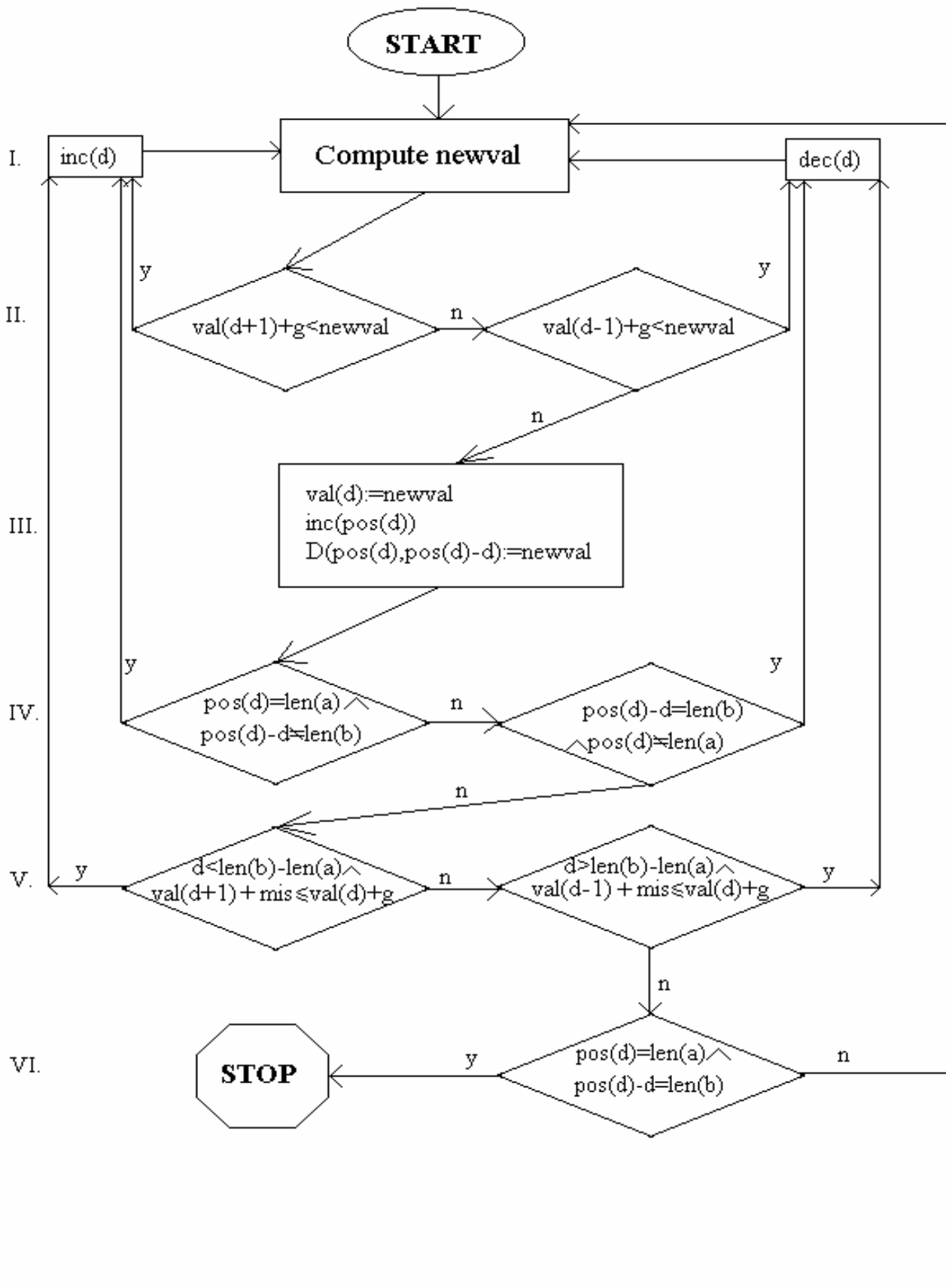
II. Ha *neval* kétséges jelöltje $d_{i,,j}$-nek, azaz $val(k+1)+g<newval$, vagy $val(k\text{-}1)+g< newval$, akkor az algoritmus átlót cserél. Vegyük észre, hogy ez a csere mindig visszalépést jelent, azaz, ha $val(k+1)+g<newval$, akkor $pos(k)>pos(k+1)$, és ha $val(k\text{-}1)+g< newval$, akkor $pos(k)>pos(k\text{-}1)\text{-}1$. Az észrevétel szintén a **7.2.1** lemmából adódik, hiszen ha valamelyik átlón előrébb járnánk, akkor a *newval*-nak értéket adó átlóelem nem lehetne nagyobb az átló utolsó kiszámított elemének az értékénél.

III. Ha bizonyos, hogy $newval=d_{i,j}$, akkor ezt az értéket az algoritmus beírja a dinamikus programozási táblázatba.

IV. Ha az algoritmus átlót cserél, ha elérte az átló végét.

V. Az algoritmus átlót cserél, ha az aktuális átló nem akadályozza meg a jobb alsó sarokhoz tartozó átlóhoz közelebb eső szomszédos átlón az előrehaladást, azaz, ha $val(k)+g \leq val(k\text{-}1)+mis$ vagy $val(k)+g \leq val(k+1)+mis$.

VI. Az algoritmus megáll, ha elérte a jobb alsó sarkot.



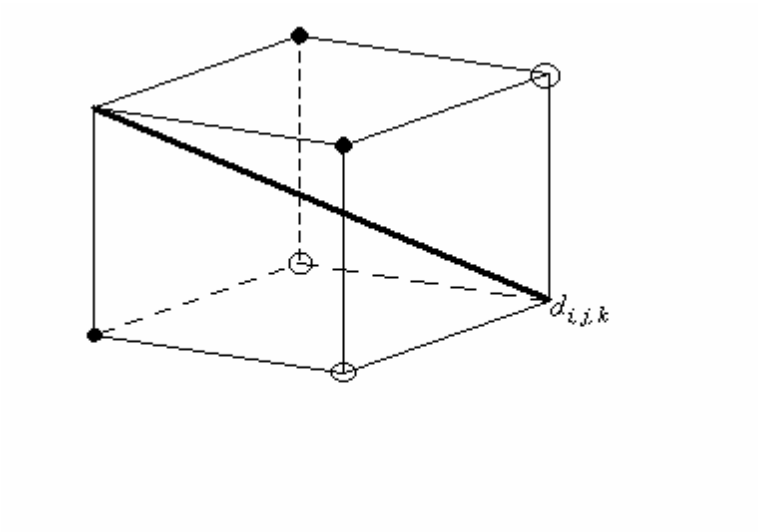
7.2.2 ábra A 7.2 fejezetben bemutatott algoritmus folyamatábrája.

A **7.2.2** lemma bizonyítja, hogy az algoritmus csak pontos értékeket ír be a dinamikus programozási táblázatba. Az algoritmus nem esik végtelen ciklusba, mert a teljesen kiszámított átlók nem akadályozzák meg az algoritmust a szomszédos átlón való előrehaladásban.

A rendre *x*, *y* és *z* hosszú *A*, *B* és *C* szekvenciák együttes illesztésére hasonló algoritmust lehet készíteni A többszörös illesztést lehet értékelni az illesztett szekvenciák páronkénti eltéréseinek az összegzésével (Carillo & Lipman, 1988) vagy a konszenzus-hiba értékelő függvénnyel (Gusfield, 1997). Ez utóbbit a

$$CE = \sum_{i=1}^{n} \min\left(\sum_{j=1}^{m} d(a_{k,i}, a_{j,i}) \,\middle|\, k = 1,2..m \right) \tag{7.2.1}$$

képlettel lehet kiszámítani, ahol *n* az illesztés hossza, *m* az illesztett szekvenciák száma, $a_{k,i}$ és $a_{j,i}$ az *i*-edik szimbólumok a *k*-adik és *j*-edik illesztett szekvenciákban, *d*(,) pedig a távolságfüggvény $\Omega \cup \{-\}$-on..



7.2.3 ábra A három dimenziós dinamikus programozási táblázatban a $d_{i,jk}$ elem kiszámításakor a fekete körrel jelzett elemeket tartalmazó átlók a vastag vonallal jelölt átló távoli szomszédjai, az üres körrel jelzett elemeket tartalmazó átlók pedig a közeli szomszédjai.

. A háromdimenziós dinamikus programozási táblázat egy átlója azon $d_{i,j,k}$ elemek halmaza, amelyre $i\text{-}j=c_1$, és $i\text{-}k=c_2$, ahol c_1 és c_2 konstans számok az átló indexei.. Egy átlónak hat szomszédos átlója lehet, ebből három közelebbi, három pedig távolabbi (**7.2.3** ábra). Meg lehet mutatni, hogy a **7.2.1** lemma fennáll háromszoros illesztések esetében is, azaz a háromdimenziós dinamikus programozási táblázatban is minden egyes átlón az értékek monoton nőnek. Így a *newval* értéke helyes jelölt $d_{i,j,k}$-ra, ha minden szomszédos átló értéke legalább *g*-vel kisebb (konszenzus-hiba értékelő függvény esetében), illetve közelebbi szomszédok esetében *g*-vel, távolabbi szomszédok esetében 2*g*-vel kisebb (páronkénti

eltérések összegzése esetén). A két és a három dimenziós algoritmus között a IV. és az V. pontban van lényegi eltérés. A háromdimenziós algoritmus akkor cserél átlót, ha *bármely* szomszédos átlón való előrehaladást az aktuális átló nem gátolja meg, és ezen szomszédos átlón az utolsó kiszámított érték koordinátái nagyobbak, mint az aktuális átlón az utolsó kiszámított érték koordinátái. Amikor az algoritmus eléri egy átló végét, az átló értéke végtelenre állítódik, ez megakadályozza az algoritmust, hogy visszatérjen erre az átlóra. Ezen változtatás oka a következő. A két dimenziós esetben egyféleképpen lehet egy adott átlóról eljutni az (*n*-*m*)-átlóra, nevezetesen, az összes közti átlót érinteni kell. A három dimenziós esetben több úton is el lehet érni $d_{x,y,z}$ átlójához, ezek között lehetnek görbe utak is. Egy olyan algoritmus esetén, amely megpróbál mindig egyenesen haladni $d_{x,y,z}$ átlója felé, a görbén levő átlók megakadályozhatnák ebben, és így végül a görbe teljes konvex burkát ki kellene számítani ahhoz, hogy az algoritmus elérje $d_{x,y,z}$ t. Empirikus eredmények mutatják, hogy ez utóbbi algoritmus lassabb.

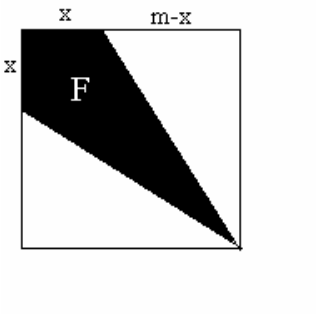
A két dimenziós algoritmust összehasonlítottam Spouge algoritmusával, a teszt érték egyik esetben $t=d_{n,m}$ a másik esetben $t=d_{n,m}$*1.2 volt. Négy elemű ABC feletti 100 hosszúságú véletlen szekvenciákat generáltam 80, illetve 0 százalék hasonlósággal (Miller & Myers, 1988) A 80% hasonlóság azt jelenti, hogy 20-20 új véletlen karakter lett beszúrva ugyanabba a véletlen, 80 hosszú szekvenciába. Két függetlenül generált szekvenciát 0%-os hasonlóságúnak tekintettem. Az illesztés értékelése *g*=3, *mis*=1 értékekkel történt. A dinamikus programozási táblázat kitöltött hányadát, illetve a szükséges iterációk számát (elképzelhető, hogy *newval* nem helyes, ezért az iterációk száma nagyobb, mint a táblázatba beírt értékek száma) a **7.2.1** táblázat tartalmazza.

A szekvenciák hasonlósága	Spouge algoritmusa $t=d_{n,m}$		Spouge algoritmusa $t=d_{n,m}$*1.2		A tesztérték nélküli algoritmus	
	F	# Iteráció	F	# Iteráció	F	# Iteráció
0%	14.61%	1662	20.09%	2206	14.22%	2205
80%	10.77%	1279	14.76%	1675	10.49%	1485

7.2.1 táblázat A két dimenziós teszt érték nélküli algoritmus hatékonysága, Spouge algoritmusával (Spouge, 1991) összehasonlítva. F jelöli a dinamikus programozási algoritmus kiszámított hányadát, # Iteráció pedig a szükséges iterációk számát, 200 futás átlagában. Az algoritmusok 0% és 80% hasonlóságú, 100 hosszúságú szekvenciákat hasonlítottak össze.

Mint látható, teszt érték nélkül is majdnem olyan gyors szekvencia összehasonlító algoritmust lehet készíteni, mint tesztérték segítségével. Amikor Spouge algoritmusában a

teszt érték pontosan a szekvenciák közötti távolságra lett állítva, a két algoritmus (Spouge és a teszt érték nélküli) nagyjából ugyanakkor hányadát számolta ki a dinamikus programozási táblázatnak. Ebben az esetben a teszt érték nélküli algoritmust valamelyest lassabb, mivel nem mindegyik *newval* pontos, ezért a dinamikus programozási táblázat egyes elemeit a teszt érték nélküli algoritmus több iterációban számítja ki. Gyakorlatban azonban nem ismerjük a szekvenciák közötti távolságot, ha ismernénk, akkor nem kellene kiszámítani. Ha 20% pontossággal meg tudjuk becsülni ezt az értéket, és a 20% felső határt választjuk teszt értéknek, akkor a teszt érték nélküli algoritmus már kevesebb iterációs lépést igényel, mint Spouge algoritmusa.



7.2.4 ábra Végtelen nagy ABC feletti *m* hosszúságú szekvenciák esetében a teszt érték nélküli algoritmus által kiszámított része a dinamikus programozási táblázatnak (F). *x* az utolsó olyan átló indexe, amelynek legalább egy elemét kiszámítja az algoritmus.

Különböző hosszúságú szekvenciák összehasonlításából kiderül, hogy a teszt érték nélküli algoritmus futási ideje is $O(l^2)$, ahol l a szekvenciák átlagos hosszúsága. Azaz a teszt érték nélküli algoritmus csak konstans gyorsítása az eredeti dinamikus programozási algoritmusnak (Seller, 1974). Az iterációs lépésekből kiderül, hogy kb. 5-8-szor gyorsabb a tesztérték nélküli algoritmus, mint az eredeti dinamikus programozási algoritmus, függve attól, hogy mennyire hasonló szekvenciák kerültek összehasonlításra. Hasonló szekvenciák esetében a szükségesen kiszámított hányada a dinamikus programozási táblázatnak kisebb, így ilyenkor az algoritmus gyorsabb. Azonban kevésbé hasonló szekvenciák esetében sem kell a teljes dinamikus programozási táblázatot kitölteni. Meg lehet mutatni, hogy végtelen nagy méretű ABC (tehát amikor két véletlen szekvencia csupa különbözőbetűből áll) és két egyforma hosszú szekvencia esetében $F=mis/2g$. Ugyanis ebben az esetben a kiszámított terület egy deltoid (**7.2.4** ábra). Legyen a két szekvencia hossza m. Ha az x-átló az utolsó, amelyik legalább egy elemet tartalmaz, akkor

$$x\ g=m\ mis-xg \tag{7.2.2}$$

Tehát

$$x=m\ mis/2g \tag{7.2.3}$$

A ki nem számított terület $(m-x)m$, tehát a kiszámított terület $m^2-(m-x)\ m= m^2\ mis/2g$. Ez az eredmény azt is mutatja, hogy mis/g növekedése F növekedésével jár együtt. A három dimenziós esetben a teszt érték nélküli algoritmus a dinamikus programozási táblázatnak nagyobb hányadát töltötte ki a konszenzus hiba értékelő függvénnyel, mint a páronkénti összegzés esetében (**7.2.2** táblázat). Ezt meg lehet magyarázni azzal, hogy a konszenzus hiba értékelés kevésbé bünteti a beszúrásokat és a törléseket, mint a páronkénti összegző értékelés.

Három szekvencia esetében mindkét ismertetett értékelő metódussal vizsgáltam a teszt érték nélküli algoritmus hatékonyságát. g és mis ugyanakkora volt, mint két szekvencia esetében. Spouge algoritmusát egyszerűen ki lehet terjeszteni a három dimenziós esetre. Ez az algoritmus csak azokat a $d_{i,j,k}$ elemeket számítja ki, amelyek teljesítik a $d_{i,j,k}+q\ g\leq t$ feltételt, ahol $q=2\cdot\max\{abs(i-j),abs(i-k),abs(j-k)\}$ a páronkénti összegző értékelés esetén és $q=\cdot\ \max\{abs(i-j),abs(i-k),abs(j-k)\}$ a konszenzus hiba értékelő függvény esetében. A teszt érték szintén a pontos távolság, illetve annak a 20%-os felső becslése volt. A **7.2.2** táblázat tartalmazza 200 futás átlagait

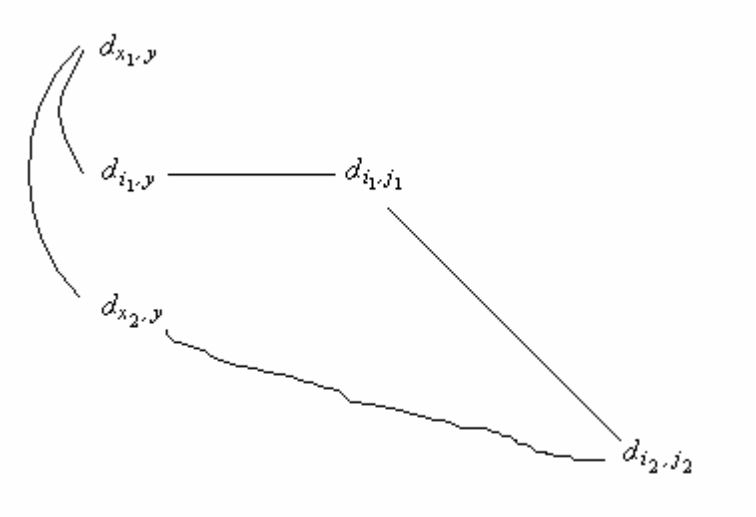
Értékelő függvény	A szekvenciák	Spouge algoritmusa $t=d_{x,y,z}$		Spouge algoritmusa $t=d_{x,y,z}$*1.2		A teszt érték nélküli algoritmus	
		F	# Iteráció	F	# Iteráció	F	# Iteráció
Páronkénti összeg	0%	11.09%	120293	18.6289%	197794	12.59%	151134
	80%	5.145%	57598	8.445%	92328	5.545%	62719
Konszenzus hiba	0%	13.46%	144786	22.31%	235163	18.67%	236833
	80%	5.496%	61206	8.951%	97497	7.434%	86722

7.2.2 táblázat A három dimenziós teszt érték nélküli algoritmus hatékonysága Spouge algoritmusával összehasonlítva. F jelöli a dinamikus programozási algoritmus kiszámított hányadát, # Iteráció pedig a szükséges iterációk számát, 200 futás átlagában. Az algoritmusok 0% és 80% hasonlóságú, 100 hosszúságú szekvenciákat hasonlítottak össze, mind a páronkénti összeg, mind a konszenzus hiba értékelő függvény segítségével.

A **7.2.2** táblázat eredményei hasonlóak a **7.2.1** táblázat eredményeihez. A konklúzió ugyanaz, Spouge algoritmusa csak akkor jobb a teszt érték nélküli algoritmusnál, ha a végeredményt előre nagyon pontosan meg lehet becsülni.

7.3 Sarokvágás konkáv gap függvény esetében

Konkáv gap függvény esetében is igaz az, hogy egy átlón az értékek szigorúan monoton növekednek. Legyen ugyanis d_{i_1,j_1} és d_{i_2,j_2} egy átló két eleme, $i_1<i_2$ és ekképpen $j_1<j_2$. A 7.2.1 lemma bizonyításának három esetében az állítás nyílván konkáv gap függvényekre is fennáll. A két megvizsgálandó eset az, amikor a d_{i_2,j_2} -be menő optimális út átugorja az i_1 sort vagy a j_1 oszlopot. Tegyük fel, hogy az optimális út átugorja az i_1 sort (**7.3.1** ábra). Ekkor az optimális úton van két egymást követő elem, $d_{x_1,y}$ és $d_{x_2,y}$, ahol $x_1<i_1$, $x_2>i_1$, és $y<j_1$. Van egy út d_{i_1,j_1} -be $d_{x_1,y}$ -ból $d_{i,y}$ -on át, két ugrással. Mivel nem biztos, hogy ez az optimális út d_{i_1,j_1} -be, ezért $d_{i_1,j_1} < d_{x_1,y} + g_{i_1-x_1} + g_{j_1-y}$ .Mivel $d_{x_2,y}$ távolabb van d_{i_2,j_2} átlójától, mint $d_{i,y}$, ezért $d_{x_1,y} + g_{i_1-x_1} + g_{j_1-y} < d_{i_2,j_2}$, és így $d_{i_1,j_1} < d_{i_2,j_2}$. A j_1 oszlop átugrása esetén a bizonyítás az előbbivel analóg.



7.3.1 ábra Egy olyan eset, amikor a d_{i_2,j_2} -ba benő út átugorja az i_1 sort. Megmutatható, hogy ekkor is $d_{i_1,j_1} < d_{i_2,j_2}$.

A sarokvágó algoritmus ciklusonként tölti ki a dinamikus programozási táblázatot. a ciklusok indexelése 0-tól kezdődik. Az algoritmus az *i*-edik ciklusban az *i* és a -*i*-átlón előrehalad addig, amíg eléri a $g_{i+1}+g_1$ értéket, aztán az *i*-1 és a -*i*+1 átlón addig, amíg eléri a $g_{i+1}+g_2$ értéket, és így tovább, kívülről befelé haladva, a *j* és -*j*-átlón előrehalad addig, amíg el nem éri a $g_{i+1}+g_{i-j+1}$ értéket. Legvégül a 0-átlón előrehalad addig, amíg el nem éri a $2g_{i+1}$ értéket.

LEMMA 7.3.1 Az előző paragrafusban megadott algoritmus nem ír be hibás értéket a dinamikus programozási táblázatba akkor, ha az addig beírt minden lehetséges értéket figyelembe veszi.

BIZONYÍTÁS Felhasználom, hogy konkáv gap függvény esetében is az átlókon monoton növekednek az értékek. Hibás érték akkor kerülhetne be a dinamikus programozási táblázatba, ha valamely, az adott pontig ki nem számolt pozíció jobb értéket küldene. A bizonyításban azt mutatom meg, hogy ez nem lehetséges. Négy esetet kell elkülöníteni

- A még el nem kezdett, azaz az *i*-átlón és a -*i*-átlón kívüli átlók nyílván nem küldhetnek kisebb értékeket.
- A 0-átlón túli, már megkezdett átlókról sem érkezhet kisebb érték. A *j*-átlón az *i*-edik körben kiszámított értékek nem nagyobbak, mint $g_{i+1}+g_{j-i+1}$. A túloldali -*k*-átlón a még ki nem számolt értékek legalább g_i+g_{i-k} értékűek. A -*k*-átlóról a *j*-átlóra átjutni legalább g_{j+k+1} súlyú úton lehetséges. És valóban $g_i+g_{i-k}+ g_{j+k+1}\geq g_{i+1}+g_{j-i+1}$, hiszen a konkáv függvény tulajdonságaiból adódóan $g_i+g_{i-k}\geq g_{i+1}$ és $g_{j+k+1}\geq g_{j-i+1}$.
- A 0-átlóval el nem választott nagyobb abszolút értékű átló még ki nem számolt értékei sem küldhetnek kisebb értéket. Legyen ugyanis $k>j$! Ekkor a *k*-átló még ki nem számolt értékei nem kisebbek, mint $g_{i+1}+g_{i-k+1}$, a *j*-átló értékei pedig nem nagyobbak, mint $g_{i+1}+g_{i-j+1}$. A *k*-átlóról eljutni a *j*-átlóra legalább g_{k-j} súlyú úton lehetséges. És megint $g_{i+1}+g_{i-k+1}+ g_{k-j}\geq g_{i+1}+ g_{i-j+1}$, hiszen a konkáv gap függvényből adódóan $g_{i-k+1}+ g_{k-j}\geq g_{i-j+1}$.
- A 0-átlóval el nem választott kisebb abszolút értékű átló még ki nem számolt értékei sem küldhetnek kisebb értéket. Legyen ugyanis $k<j$! Ekkor a *k*-átló még ki nem számolt értékei nem kisebbek, mint g_i+g_{i-k}, a *j*-átló értékei pedig nem nagyobbak, mint $g_{i+1}+g_{i-j+1}$. A *k*-átlóról eljutni a *j*-átlóra legalább g_{j-k} súlyú úton lehetséges. És megint $g_i+g_{i-k}+ g_{j-k}\geq g_{i+1}+ g_{i-j+1}$, hiszen a konkáv gap függvényből adódóan $g_i+ g_{j-k}\geq g_{i+1}$ és $g_{i-k}\geq g_{i-j+1}$.

Egyetlen részlet maradt még vissza. Ha $d_{i,j}$ kiszámításakor az *i*-edik sor és a *j*-edik oszlop minden már kiszámolt elemére összehasonlítást végeznénk, akkor az algoritmus egy elem kiszámítására $O(n+m)$ összehasonlítást végezne, így az algoritmus futási ideje nagyobb lenne $O(nm(\log n+\log m))$-nél. Szerencsére a 3.5 fejezetben bemutatott pointer technika itt is alkalmazható, így az algoritmus egy elem kiszámítására csak $\log n+\log m$ időt fordít, ugyanúgy, mint a 3.5 fejezetben ismertetett algoritmus. És mivel a sarokvágó algoritmus nem

számolja ki a dinamikus programozási táblázat összes elemét, egy konstans faktorral gyorsabb, mint a 3.5 fejezetben bemutatott algoritmus.

7.4 Sarokvágás a statisztikus szekvencia analízisben

Hein és munkatársai által megadott átformulázás (Hein et al., 2000) lehetővé teszi, hogy felső becslést adjunk két szekvencia teljes likelihoodjára. Rövidítsük $p_0'(t)$-t p-vel, $\lambda\beta(t)\pi_{\max}$-ot L-lel, ahol $\pi_{\max}$ a leggyakoribb karakter gyakorisága, és $p_1(t)f_{\max} + p_1'(t)\pi_{\max}$-ot F-fel, ahol $f_{\max}$ az f függvény maximuma. A 4.7.6 képletből adott $d_{i-1,j}$, $d_{i,j-1}$ és $d_{i-1,j-1}$ esetén $d_{i,j}$ maximális értéke

$$\max d_{i,j} = pd_{i-1,j} + Ld_{i,j-1} + (F - pL)d_{i-1,j-1} \tag{7.4.1}$$

azon a paramétertartományon, ahol

$$p_1'(t) - \lambda\beta(t)p_0'(t) > 0 \tag{7.4.2}$$

de ez $0<\lambda t,\mu t<1$ közötti paramétertartományra mindig teljesül. Teljes indukcióval megmutatható, hogy

$$\max d_{i,j} = \sum_{k=0}^{\min(i,j)} \binom{i}{k}\binom{j}{k} p^{i-k} L^{j-k} F^k \tag{7.4.3}$$

Hasonlóképpen adhatunk alsó becslést két szekvencia teljes likelihoodjára, csak akkor a fenti képletbe a minimális gyakoriságú karakter gyakoriságát és a legvalószínűtlenebb átmenet valószínűségét kell írni.

Ha a rendre n és m hosszú A és B szekvenciák teljes likelihoodját 1-α pontossággal szeretnénk meghatározni, akkor nem kell kiszámítani azokat a cellákat, melyekre

$$\left[\sum_{k=0}^{\min(i,j)} \binom{i}{k}\binom{j}{k} p^{i-k} L^{j-k} F^k\right]\left[\sum_{k=0}^{\min(n-i,m-j)} \binom{n-i}{k}\binom{m-j}{k} p^{n-i-k} L^{m-j-k} F^k\right] \le \frac{\alpha \min d_{n,m}}{nm} \tag{7.4.4}$$

ahol min $d_{n,m}$ a két szekvencia likelihoodjának az alsó becslése.

Kettő, 100 hosszúságú szekvencia esetében a **7.4.1** táblázat tartalmazza azt, hogy adott paraméterek mellett a dinamikus programozási táblázatnak legfeljebb hányadrészét kell kitölteni ahhoz, hogy 1-α pontossággal megkapjuk két szekvencia teljes likelihoodját. Adott λ esetében a μ érték úgy lett beállítva, hogy a szekvenciák hosszúság eloszlásának várható

értéke éppen 100 legyen, a szubsztitúciós modell a Jukes-Cantor modell volt (Jukes & Cantor, 1969). A táblázatból kiderül, hogy a kiszámítandó rész nagyobb mértékben függ a mutációk paramétereitől, mint α-tól. Ez azt sugallja, hogy a megadott felső határ nem erős becslése a tényleges felső határnak. Ez tényleg valószínűsíthető, hiszen egy adott cella hozzájárulását a teljes likelihoodhoz azzal a feltételezéssel becsültem, hogy a két szekvencia minden karaktere megegyezik, a két szekvencia teljes likelihoodjának alsó becslésekor pedig azt feltételeztem, hogy a két szekvenciának egyetlen közös karaktere sincs.

α = 0.01

s\λ	0.01	0.02	0.03
0.5	47.57%	53.85%	57.93%
0.7	41.93%	48.08%	52.23%
0.9	37.42%	43.18%	47.22%

α = 0.0001

s\λ	0.01	0.02	0.03
0.5	48.28%	54.48%	58.59%
0.7	42.95%	48.91%	52.95%
0.9	38.5%	44.34%	48.26%

7.4.1 táblázat Elméleti felső határok arra vonatkozóan, hogy a dinamikus programozási táblázatnak maximum hányadrészét kell kitölteni ahhoz, hogy a teljes likelihood 1-α részét megkapjuk.

A bemutatott sarokvágási technika jobban hasonlít Ukkonnen algoritmusához (Ukkonen, 1985), mint Spouge algoritmusához (Spouge, 1991). Ukkonen algoritmusa csak azokat a $d_{i,j}$ elemeket számolja ki, amelyre $|i - j|*g + |(n-i)-(m-j)|g<t$, ahol g az indelek büntetése, t a teszt érték, n és m pedig a két összehasonlítandó szekvencia hossza. Spouge algoritmusában $|i - j|*g$ helyett $d_{i,j}$ szerepel, ezáltal ez utóbbi algoritmus kevesebb értéket számít ki, mint az előbbi, mivel $|i - j|*g \leq d_{i,j}$ Ukkonen algoritmusa előre eldönti, hogy melyik értékeket számítja ki először, Spouge algoritmusa dinamikusan változtatja ezt a döntést, függve a már kiszámított értékek nagyságától. Ilyen dinamikus algoritmust akkor lehetne írni a statisztikus szekvencia illesztésben, ha a (7.4.4) egyenlőtlenség bal oldalát sikerülne kiszámítani konstans idő alatt. Jelen formában a bemutatott ötletet egy programban lehet alkalmazni, amely tartalmaz egy olyan táblázatot, amely megmutatja, hogy adott paraméterek és α esetén mely $d_{i,j}$ értékeket kell kiszámítani. A kiszámítandó értékek úgy helyezkednek el, hogy ezt egyszerűen egy $|i-j|<k_{\alpha,\theta}$ egyenlőtlenséggel lehet megadni ahol θ a paraméterek halmaza. Természetesen θ végtelen sok értéket vehet fel, de tudható, hogy a kiszámítandó terület nő λ növelésével és s csökkentésével, így elég véges sok $k_{\alpha,\theta}$-t megadni.

8. TOVÁBBI PERSPEKTÍVÁK

8.1 A kombinatorikus és analitikus módszerek egyesítése

Az 5.2 és 5.3 fejezetekben ismertetett illesztési eljárás lehetővé teszi a Thorne-Kishino-Felsenstein modell analitikus és kombinatorikus továbbfejlesztéseinek az egyesítését, illetve kiterjesztését kettőnél több szekvenciákra. Pl.: megadható egy olyan modell, amelyben a szekvenciák Poisson szekvencia hosszúságból (vagy tetszőleges egyéb eloszlásból) evolválódtak a többszörös beszúrásokat engedélyező modell alapján. A bonyolultabb módszerek számolási ideje persze a modell bonyolultságával növekszik, melyek esetében közelítő számításokat érdemes alkalmazni. A lehetséges metódusok a sarokvágási technika valamint Markov Chain Monte Carlo módszer.

A modellek számának növekedésével szükségessé válik goodness-of-fit statisztikák (Hein et al., 2000) megalkotása is, amelyek azt hivatottak eldönteni, hogy egy adott modell mennyire jól írja le az adott szekvenciák evolúcióját.

8.2 Markov Chain Monte Carlo

A Thorne-Kishino-Felsenstein modell leírható úgy, mint egy rejtett Markov folyamat (Hidden Markov Model, HMM) (Durbin et al, 1998, Metzler et al., 2001; Hein, 2001; Holmes & Bruno, 2001). A rejtett Markov folyamat azt jelenti, hogy magát a Markov folyamatot nem látjuk, csupán az egyes állapotok emisszióját. Két rokon szekvencia esetében páros rejtett Markov folyamatról (pair-HMM) beszélünk. A lehetséges állapotok a `Start`, $\begin{matrix}A\\-\end{matrix}$, $\begin{matrix}-\\A\end{matrix}$, $\begin{matrix}A\\B\end{matrix}$, `End` állapotok, ahol `A` és `B` tetszőleges betűi azon ABC-nek, amely fölötti szekvenciákat tekintünk. A Markov folyamat a `Start` állapotból indul, és az $\begin{matrix}A\\-\end{matrix}$, $\begin{matrix}-\\A\end{matrix}$, $\begin{matrix}A\\B\end{matrix}$ állapotokon keresztül az `End` állapotba érkezik. Az $\begin{matrix}A\\-\end{matrix}$ állapot az első, a $\begin{matrix}-\\A\end{matrix}$ állapot a második, az $\begin{matrix}A\\B\end{matrix}$ állapot pedig mindkét szekvenciába kibocsát egy karaktert. A lehetséges állapotok közötti átmenetek valószínűségeit a **8.2.1** táblázat tartalmazza (Hein, 2001)

	- A	A B	A -	End
Start	$\lambda\beta(t)$	$\frac{\lambda}{\mu}(1-\lambda\beta(t))e^{-\mu t}$	$\frac{\lambda}{\mu}(1-\lambda\beta(t))\left(1-e^{-\mu t}\right)$	$\left(1-\frac{\lambda}{\mu}\right)(1-\lambda\beta(t))$
- A	$\lambda\beta(t)$	$\frac{\lambda}{\mu}(1-\lambda\beta(t))e^{-\mu t}$	$\frac{\lambda}{\mu}(1-\lambda\beta(t))\left(1-e^{-\mu t}\right)$	$\left(1-\frac{\lambda}{\mu}\right)(1-\lambda\beta(t))$
A B	$\lambda\beta(t)$	$\frac{\lambda}{\mu}(1-\lambda\beta(t))e^{-\mu t}$	$\frac{\lambda}{\mu}(1-\lambda\beta(t))\left(1-e^{-\mu t}\right)$	$\left(1-\frac{\lambda}{\mu}\right)(1-\lambda\beta(t))$
A -	$\frac{1-e^{-\mu t}-\mu\beta(t)}{1-e^{-\mu t}}$	$\frac{\lambda\beta(t)e^{-\mu t}}{1-e^{-\mu t}}$	$\lambda\beta(t)$	$\frac{\beta(t)(\mu-\lambda)}{1-e^{-\mu t}}$

8.2.1 táblázat A Thorne-Kishino -Felsenstein modell tekinthető egy Markov folyamatnak. A táblázat az egyes állapotok közötti átmenetek valószínűségeit tartalmazza.

A táblázat csak a beszúrások és törlések Markov folyamatának a valószínűségeit írja le, ehhez természetesen a szekvenciák teljes evolúciójának a leírásakor még figyelembe kell venni a szubsztitúciók valószínűségeit is. A szemlélő természetesen a Markov folyamatot nem látja, csak a folyamat emisszióját, azaz a két homológ szekvenciát. A **8.2.1** táblázat alapján bármely illesztésnek a valószínűsége kiszámítható, a maximum likelihood szekvencia illesztés feladata éppen az, hogy dinamikus programozási algoritmus segítségével az összes lehetséges illesztés valószínűségét összeadja, majd megkeresse azokat a paramétereket, amelyek segítségével ezen valószínűségek összege maximális. Kettőnél több szekvencia evolúcióját többszörös rejtett Markov folyamattal (multiple-HMM) (Holmes & Bruno, 2001) lehet leírni. Ebben az esetben is a teljes valószínűség kiszámítható dinamikus programozási algoritmussal (Hein, 2001, Steel & Hein 2001; Miklós 2001b), de az algoritmus futási ideje a szekvenciák számával exponenciálisan nő, így legfeljebb három-négy szekvencia esetében lehetséges a pontos teljes likelihood értéket kiszámítani.

Négynél több szekvencia esetében reménytelen az összes lehetséges leszármazás valószínűségét kiszámítani. (Egy 466 MHz-es gépen körülbelül 100000 óra alatt futna le 5 szekvencia statisztikus illesztése, maximum likelihood paraméterek megkeresésével együtt.) Ekkor válik rendkívül hasznossá a HMM megközelítés. A teljes likelihood kiszámítása helyett a lehetséges leszármazásokból veszünk mintákat, úgy, hogy az egyes minták választásának a valószínűsége egyezzen meg a leszármazás likelihoodjának a teljes likelihoodban való arányával. Az ilyen mintavételt hívjuk Markov Chain Monte Carlo technikának (Gamerman, 1997). Egy adott fa mentén többszörös illesztések mintavételezését a TKF91 modell alapján először Holmes és Bruno adott meg (Holmes & Bruno, 2001). Két szekvencia esetén

illesztések és evolúciós paraméterek párhuzamos mintavételezésével lehetőség van arra is, hogy vizsgáljuk azt, hogy egy adott illesztés vagy adott paraméterhalmaz mennyire elfogadható, azaz mennyire reprezentálja két szekvencia leszármazási viszonyait (Metzler et al. 2001).

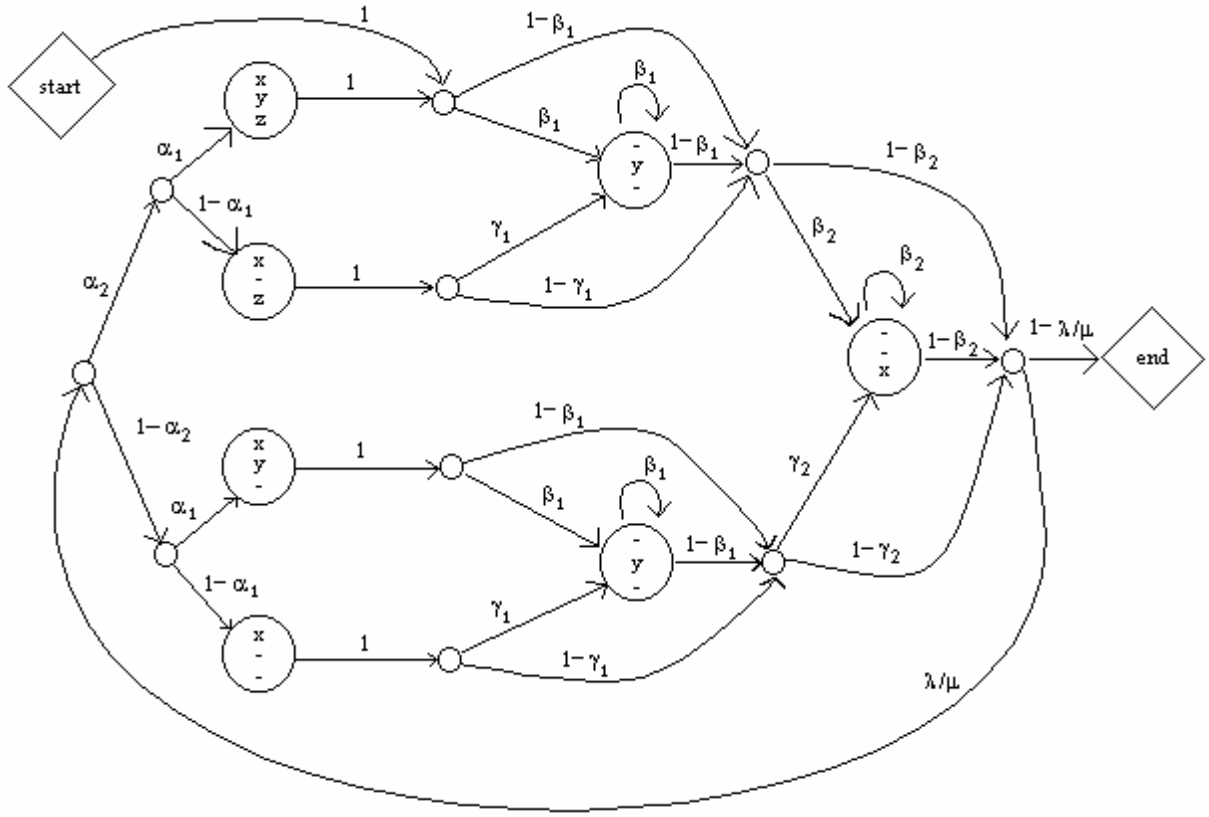
A többszörös beszúrások modellje (6.2 fejezet) szintén tekinthető rejtett Markov folyamatnak. Ekkor persze a rejtett állapotok száma is végtelen nagy lesz, hiszen tetszőleges hosszúságú beszúrások lehetségesesek, és ezek mind külön folyamatok, amelyeket egy-egy rejtett állapot reprezentál. Hasonlóan a többszörös törlések modellje is tekinthető rejtett Markov folyamatnak. A 6.4 fejezetben bemutatott modell viszont nem fogalmazható át az előbbiekben ismertetettekhez hasonló rejtett Markov modellé, mert egyes linkek születésének a rátája függ a szekvencia aktuális hosszától. Viszont ez a modell is átírható az összes szekvencia összes lehetséges illesztéseinek a rejtett Markov modelljévé. Ebben a modellben a rejtett állapotok a lehetséges illesztések, az emissziók továbbra is a vizsgálni kívánt szekvenciák. Természetesen HMM-ek használatára csak akkor kerülhet sor, ha az egyes átmenetek valószínűségei ismertté válnak, ehhez viszont ismerni kellene az összes lehetséges leszármazás valószínűségét.

8.3 Dinamikus programozás *versus* MCMC

A **8.3.1** ábra azt a többszörös rejtett Markov folyamatot mutatja be, amely segítségével egy ősi *X* szekvenciához a *Y* és *Z* modern szekvenciákat lehet illeszteni (Holmes & Bruno, 2001). Az ábra alján van egy nagy ciklus, amely csak az x/-/- emittáló állapoton halad át (az üres körök nem emittáló állapotok, csak az ábra egyszerűbb lerajzolását segítik elő). Mivel az ősi szekvenciát nem ismerjük, ezen a körön való végighaladás valószínűsége független a két modern szekvenciától. A dinamikus programozásban az $O(l^{2n})$ futási időnek $O(l^n)$-re való csökkentését az az ötlet tette lehetővé, hogy az egymás után következő kihalt ősi linkek egyetlen állapotként lettek kezelve. Ez az ötlet az illesztések mintavételezésénél is használható. Ahelyett, hogy az előbb említett körön cirkulálnánk, egyetlen egy lépésben el lehet dönteni, hogy ezen a körön

1. hány kört teszünk meg
2. melyik állapotnál hagyjuk el a kört.

Az ötletet valóban alkalmazták is a program megalkotói is, és a *Null cycle elimination* nevet adták neki (Holmes, személyes közlés).



8.3.1 ábra Többszörös rejtett Markov folyamat, amely az ősi *X* szekvenciához illeszti az *Y* és *Z* modern szekvenciákat. A kis körök nem emittáló állapotokat jelölnek

A fenti példa is rávilágít a dinamikus programozás és a MCMC technika közötti kapcsolatra. Napjainkban a legtöbb kutató a MCMC technikát preferálja a dinamikus programozással szemben. Valóban az MCMC technika látványosabb, mint a dinamikus programozás, hiszen amíg a dinamikus programozás számítási igénye exponenciálisan növekszik a vizsgálatba bevont szekvenciák számával, addig egyetlen többszörös illesztés mintavételezéséhez szükséges idő csak lineárisan növekszik a szekvenciák számával. Ami az MCMC hátránya lehet, az az, hogy jelenleg nem tudjuk pontosan megmondani azt, hogy hány mintavételezés szükséges ahhoz, hogy az egyes illesztések likelihoodjainak az eloszlásáról pontos képet kapjunk. Előfordulhat, hogy a szükséges mintavételezések száma exponenciálisan növekszik, és ekkor az MCMC technika sem hoz lényeges áttörést a többszörös szekvencia illesztésben.

A Spouge által definiált *computational volume* a dinamikus programozási táblázat azon része, amely az összes optimális illesztést tartalmazza (Spouge, 1991). Empirikus eredmények azt mutatják, hogy két szekvencia esetében ez a terület a szekvenciák hosszának másfeledik hatványával arányos. Ennek ellenére a létező sarokvágási technikák mindegyike $O(l^2)$ futási idejű. Paradox helyzetnek tűnik a végtelen nagy ABC feletti szekvenciák

illesztése. Ha tudjuk, hogy a két azonos hosszúságú szekvenciának nincsen közös karaktere, akkor dinamikus programozás nélkül is tudható, hogy az optimális illesztés egyetlen egy gap-et sem tartalmaz, a főátló *computational volume*, mégis ezen információ hiányában a sarokvágási technikát alkalmazó algoritmusok ebben az esetben igénylik a legtöbb számolási időt. A háromdimenziós algoritmus tesztelésekor kiderült, hogy az összes optimális illesztés konvex burka jelentős mennyiségű szuboptimális illesztést tartalmazhat.

Az illesztések statisztikus mintavételezésénél is könnyen előfordulhat, hogy $O(l^n)$ nagyságú részt kell mintavételezni ahhoz, hogy elég pontosan meghatározzuk a szekvenciák teljes likelihoodját. Egyenlőre azok a mintavételezési próbálkozások, amelyek arra irányulnak, hogy dinamikus programozási táblázat minél nagyobb részéből vegyünk mintát, nem jártak eredménnyel (Holmes & Bruno, 2001).

A MCMC technikának még fontosabb szerepe lehet olyan modellek esetében, amelyekre a dinamikus programozási eljárás $O(l^n)$-nél több időt igényel, ugyanis ezekben az esetekben is lehetséges lineáris idejű mintavételezés. A dinamikus programozás ezekben az esetekben nem azért lassabb, mert a lehetséges illesztések száma megnövekszik, hanem azért, mert az illesztések likelihoodjait nem lehet olyan hatékonyan összegezni. Mivel ilyen modellek a közeljövőben tért fognak hódítani a statisztikus szekvencia analízisben, ezért az MCMC technika jelentősége semmiképpen nem elhanyagolható.

9. ÖSSZEFOGLALÁS

A biológiai szekvenciák evolúciós vizsgálata statisztikus módszereket igényel. Illesztett szekvenciák esetében statisztikus módszerek már húsz éve ismertek (Felsenstein, 1981), és azóta a szubsztitúciók modellezése kielégítő módon fejlődött (Hillis et al., 1996). A szekvenciák statisztikus illesztésére kevesebb figyelmet szenteltek az elmúlt időkig. Pedig a statisztikus illesztés rendkívül fontos, mint ahogy Jotun Hein megállapította (Hein et al., 2000):

> It is an inconsistent approach to first use parsimony/similarity and then halfway in the analysis switch to a statistical approach. In addition, the alignment created by parsimony/similarity can create unknown biases in the estimation of substitution parameters, as these procedures will align to create as much identity within each column as possible.

Az első próbálkozás statisztikus szekvencia illesztésre Bishop-tól és Thompson-tól származik (Bishop & Thompson, 1986). Ez a módszer még csak közelítő értéket adott két szekvencia likelihoodjára. Az első pontos számolást Thorne és munkatársai adták meg (Thorne et al., 1991). Az azóta TKF91 néven emlegetett modellnek azonban több biológiailag irreleváns tulajdonsága is van. Egyrészt a modell feltételezi, hogy a szekvenciák hosszúságának egyensúlyi eloszlása geometriai, ami ellentmond a biológiai megfigyeléseknek (Zhang, 2000). A másik gyenge pontja a modellnek az, hogy nem engedélyezi több karakter (aminosav vagy nukleotid) beszúrását egyetlen evolúciós eseményként. Hein és munkatársai által bemutatott goodness-of-fit statisztika segítségével megmutatható, hogy egy ilyen modell nem írja le jól a biológiai szekvenciák evolúcióját (Hein et al., 2000). Thorne és munkatársai megpróbálták a TKF91 modellt úgy továbbfejleszteni, hogy az engedélyezze több karakter beszúrását és törlését (Thorne et al., 1992). Azonban számolási okokból a hosszú beszúrásokat egyetlen széttörhetetlen fragmentumként kezelték, és csak egész fragmentumok törlődhetnek, ami megint biológiailag nem megalapozott feltételezés (Thorne et al., 1992; Miklós & Toroczkai, 2001).

Az értekezésem első felében olyan új modelleket mutattam be, amelyek megpróbálják a fent említett hibákat kiküszöbölni. A bemutatott modellek két nagy csoportra oszthatóak.

1. A TKF91 modell kombinatorikus továbbfejlesztései.
2. A TKF91 modell analitikus továbbfejlesztései.

A TKF91 modell kombinatorikus továbbfejlesztésein olyan modelleket értek, amelyekben a tranziens viselkedést továbbra is az eredeti TKF91 modell írja le. Ilyen értelemben a TKF92 modell is a TKF91 modell kombinatorikus továbbfejlesztésének tekinthető. A modellek alapja egy olyan új illesztés típus, amely az ismeretlen ősi szekvenciához illeszti hozzá a modern szekvenciákat. Ennek a segítségével sikerült több modellt felállítani, és az adott modellek alapján a kapcsoltsági valószínűségeket kiszámító gyors algoritmusokat írni. Megmutattam, hogy

1. Létezik $O(l^3)$ futási idejű algoritmus, amely Poisson eloszlású szekvencia hosszúságokból evolválódó szekvenciák kapcsoltsági valószínűségeit számolja ki (Miklós, 2001c).
2. Létezik $O(l^2)$ futási idejű algoritmus, amely olyan modell alapján számítja ki a kapcsoltsági valószínűségét két szekvenciának, amely modell megengedi az egyes ágakon az átfedő törléseket (Miklós, 2001a).. Ezt a TKF92 modell nem engedélyezte.
3. Létezik $O(l^n)$ futási idejű algoritmus, amely kiszámítja n darab olyan szekvencia kapcsoltsági valószínűségét, amelyek egy csillag alakú fa mentén evolválódtak (Miklós, 2001b). Ez az algoritmus jelentősen gyorsabb, mint az eddig ismert (Steel & Hein, 2001)

A TKF91 modell analitikus továbbfejlesztésein olyan modelleket értek, amelyekben már megváltozik a tranziens viselkedés is a TKF91 modellhez képest. Megmutattam, hogy a TKF91 modell átírható a sztochasztikus sorban állási rendszerek nyelvére. A sorban állási rendszerek elméletében alkalmazott generátorfüggvények segítségével egy olyan modellt készítettünk, amely engedélyezi széttörhető hosszú fragmentumok beszúrását (Miklós & Toroczkai, 2001). Ezenkívül két további modellt állítottam fel. Ezeknek a modelleknek még nem ismerjük a tranziens viselkedését, mert a generátorfüggvényüket leíró parciális differenciálegyenleteknek a megoldásai nem ismertek. Az egyik modell a hosszú beszúrásokon kívül engedélyez hosszú törléseket is, a másik modellben a szekvenciák hosszúságának az egyensúlyi eloszlása Poisson, ami lényegesen jobb közelítés, mint a TKF modellek geometriai eloszlása.

Az értekezésem második felében a sarokvágási technikák továbbfejlesztésiben elért eredményeimet tettem közzé. A sarokvágás olyan technika, amely lehetővé teszi, hogy a dinamikus programozási táblázat jobb felső és bal alsó sarkának a kiszámítása nélkül kapjuk meg két szekvencia optimális illesztését. Az eddig publikált sarokvágási technikák egy ún. teszt értéket igényelnek, és az algoritmusok csak akkor adnak helyes végeredményt, ha az

optimális illesztés súlya ennél az értéknél kisebb. A statisztikus illesztésben nem is világos, hogy milyen teszt értéket kellene megadni, hiszen itt nem a teljes likelihood alapján határozzák meg a szekvenciák evolúciós távolságát, hanem azt a maximum likelihood paraméterekkel írják le, amihez az adott paraméterek mellet ki kell számolni a szekvenciák teljes likelihoodját. Az értekezésemben megmutattam, hogy mind a távolságalapú szekvencia illesztésben, mind a statisztikus szekvencia illesztésben lehetőség van teszt érték nélküli sarokvágásra. A statisztikus szekvencia analízisben elért eredmények rosszabbak, mint a távolságalapú illesztésben elértek. Ennek az az oka, hogy a statisztikus illesztésben csak egy durva becslést sikerült adnom a dinamikus programozási táblázat azon részére, amely bizonyosan nem járul hozzá jelentős mértékben a teljes likelihoodhoz, míg a távolságalapú módszereknél ezt a becslést folyamatosan lehet finomítani a már kiszámított értékek ismeretében. A statisztikus illesztésben is lehetőség van a becslés folyamatos finomítására, de ehhez olyan egyenlőtlenségek szükségesek, amelyeket nem lehet konstans idő alatt kiszámítani. Így amit nyerünk a réven, elveszítjük a vámnál.

Az értekezésem befejező részében a további perspektívákat vázoltam fel. A statisztikus szekvencia illesztés további fejlődésében valószínűleg nagy szerepet fognak játszani a statisztikus mintavételezési módszerek. Az értekezésemben bemutatott modellek és a MCMC és hasonló technikák kombinálása újabb utat nyithat meg a bioinformatika fejlődésében.