STATISZTIKUS SZEKVENCIA ILLESZTÉS

Elméleti biológia és ökológia program

Dr. Szathmáry Eörs
programvezető

Dr. Podani János
témavezető

ELTE TTK Növényrendszertani és Ökológiai Tanszék

TARTALOMJEGYZÉK

"Biology is so digital, and incredibly complicated, but incredibly useful…
It is hard for me to say confidently that after fifty more years of explosive growth of computer science, there would still be a lot of fascinating unsolved problems […] 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…"

Donald E. Knuth

1. BEVEZETÉS

A biológiai makromolekulák vizsgálata gyökeres változást hozott a biológiai tudományok területén. Needleman és Wunsch cikke (Needleman & Wunsch, 1970) óta az elmúlt harminc év alatt egy új diszciplína született, a bioinformatika. Ma már több tudományos folyóirat jelenik meg, amely kifejezetten ezen tudományág eredményeit publikálja (Bioinformatics, Journal of Computational Biology), és ezeken kívül is számos újság — az Algorithmica-tól kezdve a Bulletin of Mathematical Biology-n át a Journal of Molecular Biology-ig — közöl ilyen témájú cikkeket. Mára az évenként publikált cikkek terjedelme meghaladja a 10000 oldalt. Felmerül a kérdés, hogy miért alakult ki egy új tudományág, miért kellett egy új matematikai eszközrendszert kialakítani a biológiai makromolekulák vizsgálatára, mely diszciplínákkal és milyen mértékben rokon a bioinformatika. A válasz a nukleinsavak és fehérjék szekvenciális adatszerkezetében rejlik.

A DNS szerkezetének a felfedezése James Watson és Francis Crick nevéhez fűződik (Watson & Crick, 1953). Az általuk javasolt — és később többszörösen igazolt — kettős spirál modell szerint a DNS két egymást átfogó szál, amelyek szabályos kettős spirális alakban csavarodnak egymásra. A DNS szálak építőkövei a nukleotidok, melyek önmagukban is összetett vegyületek: egy nukleotid egy cukor foszfát és egy nukleozid bázis egységből épül fel. A DNS szálak gerincét az egymáshoz kapcsolódó cukor-foszfát molekulák adják, ezekről ágaznak le a bázisok. A DNS-ben négyféle bázis — és így négyféle nukleotid — található. Minden bázis hidrogénkötésekkel kapcsolódik a szemközti szál bázisához. Az adenin csak a timinnel, a guanin csak a citozinnal kapcsolódhat. Így az egyik szál sorrendje a másik szál sorrendjét is meghatározza. Az előbbiek alapján világos, hogy egy DNS szerkezetét egyértelműen meg lehet adni az egyik szál bázisainak a sorrendjével.

Az RNS molekula szintén nukleotidokból épül fel. A nukleotidok cukor-foszfát alegysége nem dezoxiribózt, hanem ribózt tartalmaz. A négy bázis sem teljesen azonos, az RNS-ben timin helyett uracil található. Az RNS csak egyetlen nukleotid láncból épül fel, s a lánc visszakanyarodva rövidebb-hosszabb szakaszokon önmagával alkothat bázispárokat. Így az RNS molekuláknak sajátos térszerkezetük lehet.

A fehérjék építőkövei az aminosavak. A természetben előforduló, fehérjéket felépítő húsz különböző aminosav az α szénatomon elhelyezkedő oldalláncokban különbözik. Az aminosavak peptidkötéseket kialakítva kapcsolódnak egymáshoz. A peptidkötés bármely két

aminosav esetében azonos, így egy fehérje szerkezetét alapvetően a benne található aminosavak sorrendje határozza meg.

Mint látható, a három típus — DNS, RNS, fehérje — bármelyikébe is tarozik egy makromolekula, egyértelműen megadható a benne levő egyszerű kémiai építőkövek (nukleotidok vagy aminosavak) sorrendjével. Ez az adattípus alapvetően különbözik a cönológiában és ökológiában előforduló sokváltozós adattípusoktól (Podani, 1997). A biológiai makromolekulák nem csak szubsztitúcióval (az egyes építőkövek cseréjével) változhatnak meg, hanem hosszabb-rövidebb láncok törlődhetnek, illetve ékelődhetnek be. A törlés (deletion) és beszúrás (insertion) helyei két rokon biológiai szekvencia összehasonlításakor nem ismertek, a szekvenciák illesztése (alignment) a vizsgálat szerves részét kell hogy képezze.

A lehetséges illesztések száma exponenciálisan nő a szekvenciák hosszával. Ez azt jelenti, hogy minden egyes illesztéssel külön-külön nem lehet foglalkozni, mert egy ilyen módszerrel hosszabb makromolekulák vizsgálata még a leggyorsabb számítógépekkel is nagyon lassú lenne. Needleman és Wunsch fent idézett korszakalkotó cikkében található az első olyan algoritmus, amely egy előre megadott szempontrendszer alapján gyorsan — a szekvenciák hosszának polinomiális kifejezésével arányos időben — megtalálja két szekvenciának a legoptimálisabb illesztését. Ebben a cikkben a megadott szempontrendszer egy hasonlósági függvény volt, és az algoritmus azt az illesztést adja meg, amely mentén a kiszámított hasonlósági index a maximális. A bemutatott algoritmus az ún. dinamikus programozási algoritmusok családjába tartozik (Bellman, 1957). Ezen algoritmusokra az jellemző, hogy egy bonyolult problémát több részre bontanak fel, és az egyes részproblémák megoldásával jutnak el az eredeti probléma megoldásához. Hamarosan hasonló ötlet alapján olyan algoritmusokat publikáltak, amelyek egy távolságfüggvényt (Sellers, 1974) vagy transzformációs súlyok összegét (Waterman et al., 1976) minimalizálják. Később megmutatták, hogy a távolság és a hasonlóság alapú módszerek duális problémák (Smith et al., 1981).

Mint látható, a bioinformatika egyik központi problémája a szekvenciák illesztése. A fent említett algoritmusok azonban nem oldották meg teljes mértékben a szekvencia-illesztés problémáját, csupán megmutatták azt az utat, amely mentén a biológiának és az informatikának az interdiszciplínája döbbenetes fejlődésnek indult. Az alapalgoritmusokkal szemben ugyanis a következő kritikák merülnek fel:

- Az előre megadott szempontrendszer, legyen az akár hasonlóság, akár távolságfüggvény, szubjektívvé teszi a makromolekulák vizsgálatát (Thorne et al., 1991). Az előre megadott értékeket változtatva a legjobb illesztés más és más lesz. Inkorrekt paraméterek mentén inkorrekt illesztést kapunk, amelyből a helyes paraméterek nem állapíthatóak meg (Fleiβner et al., 2000)
- Távoli szekvenciák esetében csak egyes konzervatív régiókat lehet összehasonlítani, a variábilisabb régiók hasonlósága nem nagyobb, mint két véletlen úton generált szekvenciáé. Ekkor a szekvenciáknak a leginkább hasonló részeit kell lokálisan illeszteni (Smith & Waterman, 1981), és a többi részt kihagyni a vizsgálatból.
- A fehérjék aminosav sorrendje gyakorlatilag egyértelműen meghatározza a fehérje térbeli szerkezetét (Anfinsen, 1973), habár bizonyos fehérjék esetében chaperon fehérjék is közrejátszanak a térbeli szerkezet kialakításában (Barnhart et al, 2000; Normark, 2000). Azonban a mai napig nincs olyan algoritmus, amely a kémiai építőkövek sorrendjéből tökéletesen megállapítaná a térbeli szerkezetet.
- A térszerkezet-predikció egyik leghatékonyabb módszere a mai napig az ismeretlen szekvenciák ismert térszerkezetű szekvenciákkal való összehasonlítása. Mint említettem, a dinamikus programozási technikával viszonylag gyorsan lehet összehasonlítani két biológiai makromolekulát. Azonban ezen gyors technika és az egyre gyorsabb processzorokkal rendelkező számítógépek ellenére is ún. információs deficit alakult ki: az ismert fehérje térszerkezetek száma jóval kisebb, mint az ismert fehérje-szekvenciák száma, és a kettő aránya évről évre egyre kisebb.
- A dinamikus programozási technika csak két szekvencia összehasonlításakor polinomiális idejű, a szekvenciák számával a futási idő exponenciálisan növekszik. A távolság vagy hasonlóság alapú többszörös illesztés (multiple alignment) ráadásul még a páronkénti illesztésnél is szubjektívebb: nincs a két szekvencia illesztésénél használt szempontrendszereknek objektív kiterjesztése a többszörös illesztésre.

Végül, a bevezetés végén mindenképpen meg kell említeni, hogy igazságtalan az az állítás, hogy a bioinformatika csak a szekvencia-illesztésről szól. Négy témakört kell megemlíteni, amely szintén a bioinformatikához tartozik, bár jelen értekezés ezekkel a témakörökkel egyáltalán nem foglalkozik:

- Egy hosszú szekvencia összerakása sok rövid szekvenciából (shortest superstring) Jelentősége a DNS ún. shotgun analízisében van.
- Evolúciós távolságok számítása kromoszómamutációk alapján.
- A rekombináció témakörével foglalkozó vizsgálatok
- Összetett biokémiai rendszerek modellezése. Néhány kutató szerint ez a témakör néhány éven belül nagymértékű fejlődésnek fog indulni.

2. CÉLKITŰZÉSEK

Tekintsük a következő két illesztést:

```
- A                    A
C -                    C
```

A bal oldali illesztés azt szimbolizálja, hogy egy szekvenciába egy adott helyen egy C karaktert (citozin) szúrtunk be, a beszúrás melletti pozícióban álló A karaktert (adenozin) pedig kitöröltük. Azaz ez az illesztés egy beszúrás és egy törlés mutációját írja le. A jobb oldali illesztés egyetlen szubsztitúciót ír le. A távolság/hasonlóság alapú módszerek a minimális súlyú, illetve maximális hasonlóságú illesztéseket keresik meg. A távolság alapú módszerek esetében a bal oldali illesztést $w(- \rightarrow C)+w(A \rightarrow -)$ súllyal büntetjük, a jobb oldali illesztést pedig $w(A \rightarrow C)$ súllyal. Az egyes mutáció típusok súlyai a mutációk gyakoriságából adódnak, és így általában $w(- \rightarrow C)+w(A \rightarrow -) > w(A \rightarrow C)$, mivel a bal oldali illesztés által reprezentált mutáció sorozat ritkább esemény, mint a jobb oldali által reprezentált. Ekképpen az optimális illesztésben soha nem találunk a bal oldali illesztéshez hasonló részt (azaz egy karakter beszúrását és törlését közvetlenül egymás mellett). De az az állítás, hogy ez a mutáció sorozat ritkábban következik be, mint egy szimpla szubsztitúció, nem azt jelenti, hogy ez a mutáció sorozat soha nem következik be! Így a távolság/hasonlóság alapú illesztő algoritmusok által megadott optimális illesztések alulbecsülik a beszúrások és törlések gyakoriságát. (Illetve, ha a kezdeti transzformációs súlyokat úgy határozzuk meg, hogy $w(- \rightarrow C)+w(A \rightarrow -) < w(A \rightarrow C)$ legyen, akkor az optimális illesztésben soha nem látunk $A \rightarrow C$ szubsztitúciót, és így az optimális illesztés alulbecsüli a szubsztitúciók gyakoriságát.). A transzformációs súlyokra nem lehet egy kezdő objektív értéket adni, és mint a fentiekből kiderül, egyetlen optimálisnak vélt illesztésből ezeket a súlyokat pontosan nem lehet megbecsülni.

A biológiai szekvenciáknak a távolság vagy hasonlóság alapú elemzésénél objektívebb vizsgálatára ad lehetőséget a maximum likelihood módszer, amely az összes lehetséges illesztést figyelembe veszi. A módszer három lépésből áll:

- Az első lépés egy sztochasztikus modell felállítása a makromolekulák evolúciójára.

- Meg kell adni egy gyors algoritmust, amely a modellben szereplő paraméterek bármely lehetséges értékei mellett kiszámolja kettő vagy több szekvencia kapcsoltsági valószínűségét, azaz annak a valószínűségét, hogy a szekvenciák egy közös ősszekvenciából evolválódtak. Kettőnél több szekvencia esetében a bemenő adatok nem csak a szekvenciákat és az evolúciós folyamatokat leíró paramétereket tartalmazzák, hanem a leszármazási kapcsolatokat is.
- Meg kell adni azokat a paraméterértékeket, — kettőnél több szekvencia esetében a leszármazási kapcsolatokat is — amelyek mentén a kapcsoltsági valószínűség maximális. Az evolúciós paraméterek a mutációs ráták és az evolúciós idő szorzatai, és ezek a szorzatok jól írják le a szekvenciák rokonsági viszonyát (Thorne et al. 1991).

A bioinformatikai közösségben egyre növekvő igény van statisztikailag megalapozott szekvencia összehasonlító módszerekre (Hein et al., 2000). A maximum likelihood módszer hátránya az, hogy az eddig ismert beszúrás és törlés modellek (Thorne et al., 1991, 1992) leegyszerűsítve írják le a makromolekulák evolúcióját. Az értekezés célja jobb evolúciós modellek felállítása és olyan gyors algoritmusok kidolgozása, amelyek az adott modell alapján kiszámolják a kapcsoltsági valószínűséget.

Az értekezés felépítése a következő: Először a távolság és hasonlóság alapú páronkénti és többszörös szekvencia illesztési módszereket ismertetem. Bemutatom az illesztésben előforduló beszúrások és törlések értékelési módszereit, illetve az ún. sarokvágási (corner cutting) technikát. Az itt elhangzó definíciók és leírt algoritmusok nagymértékben elősegítik a maximum likelihood módszer jobb megértését, illetve az itt szereplő technikákat felhasználom a statisztikus szekvencia vizsgálatban is.

A következő fejezetben mutatom be a statisztikus szekvencia illesztés eddig elért eredményeit. Az első modellt Thorne, Kishino és Felsenstein publikálta (Thorne et al., 1991), amely mostanában annyira közismerté vált, hogy csak TKF91 rövidítéssel szerepel a szakirodalomban. Ennek egy változata, mely hosszú részszekvenciák törlését és beszúrását is lehetővé teszi, egy évvel később készült (Thorne et al., 1992). A TKF91 és TKF92 modellek két szekvencia kapcsoltsági valószínűségét adják meg. Napjainkban a TKF91 modellt kiterjesztették több szekvenciára is (Steel & Hein, 2001, Hein, 2001), valamint egy gyorsabb algoritmust közöltek a kapcsoltsági valószínűség kiszámítására (Hein et al., 2000).

Az ezt követő fejezetekben mutatom be a doktori programban elért eredményeket. Először a TKF91 és TKF92 modellek kombinatorikus továbbfejlesztései kerülnek tárgyalásra.

Az itt elért eredmények alapján lehetőség nyílik olyan algoritmus megadására, amely kettőnél több szekvencia kapcsoltsági valószínűségét az eddig ismertnél több nagyságrenddel gyorsabban számolja ki. Ezután megmutatom, hogy a TKF91 modell leírható a sztochasztikus sorban állási rendszerek nyelvén. A sorban állási rendszerek elméletében kidolgozott matematikai eszköztár segítségével új evolúciós modelleket konstruálok. Megfogalmazok a sorban állási rendszerek nyelvén olyan modelleket, amelyek esetében egyenlőre még nem létezik gyors algoritmus a kapcsoltsági valószínűség kiszámítására. Végül megmutatom, hogy hogyan lehet a sarokvágási technikát a maximum likelihood módszerben alkalmazni.

A befejező részben a további lehetséges perspektívák kerülnek felvázolásra. Egy lehetséges továbblépés a kombinatorikus és az analitikus módszerek egyesítése. Megpróbálok kapcsolatot teremteni a sarokvágási technika és egy másik új, a napjainkban sok kutató által preferált, MCMC (Markov Chain Monte Carlo) technika között.

3. TÁVOLSÁG ÉS HASONLÓSÁG ALAPÚ SZEKVENCIAÖSSZEHASONLÍTÓ MÓDSZEREK

3.1 Problémafelvetés

Az információt hordozó DNS molekulák a sejt kettéosztódása előtt replikálódnak, 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, amikből 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.

3.2 A páronkénti szekvencia illesztés alapalgoritmusa

Tegyük fel, hogy az egyes mutációk egymástól függetlenek, így egy mutáció sorozat 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 nyilvánvalóan jó választás a valószínűség logaritmusának a mínusz egyszerese. Ekkor egy mutáció sorozat súlya az egyes mutációk súlyainak az összege. Feltételezzük, hogy egy mutációs változás és a megfordítottja — például egy timin szubsztitúciója adeninre és vissza — ugyanakkora valószínűséggel fordul elő. Így a két szekvencia közös ősből való leszármazása helyett azt kell vizsgálni, hogy hogyan alakulhatott ki az egyik szekvencia a másikból. A minimális evolúció elméletét feltételezve, azt a minimális súlyú mutáció sorozatot keressük, amely az egyik szekvenciát a másikba alakítja. Fontos kérdés, hogy hogyan lehet egy minimális súlyú mutáció sorozatot (lehet, hogy több minimális értékű sorozat van) hatékonyan megkeresni. A naiv algoritmus megkeresi az összes lehetséges mutáció sorozatot, és kiválasztja belőle a minimális súlyút. Ez

nyilvánvalóan nagyon lassú, mert a lehetséges sorozatok száma exponenciálisan nő a szekvenciák hosszával.

Az alábbiakban Sellers algoritmusát mutatom be (Sellers, 1974). 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 különböző transzformációk hajthatók végre:

- Egy a szimbólum beszúrása egy szóba egy adott pozíciónál. Ezt a transzformációt $-\rightarrow a$ jelöli.
- Egy a szimbólum törlése egy szóból egy adott pozíciónál. Ezt a transzformációt $a\rightarrow -$ jelöli.
- Egy a szimbólum cseréje egy b szimbólumra egy adott pozícióban. Ezt a transzformációt $a\rightarrow b$ jelöli.

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

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

$$T_1\circ T_2=S \tag{3.2.1}$$

akkor

$$w(T_1)+w(T_2)=w(S) \tag{3.2.2}$$

Két szekvencia, A és B transzformációs távolságán az A-t B-be vivő minimális súlyú transzformációk súlyát értjük, azaz

$$\delta(A,B) = \min\{w(T)|T(A) = B\} \tag{3.2.3}$$

Ha w-ről feltesszük, hogy

$$w(a\rightarrow b) = w(b\rightarrow a) \tag{3.2.4}$$

$$w(a\rightarrow a) = 0 \tag{3.2.5}$$

$$w(a\to b) + w(b\to c) \le w(a\to c) \tag{3.2.6}$$

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

Már itt fel szeretném hívni a figyelmet arra, hogy habár a súlyfüggvénnyel való számolást egy valószínűségelméleti okoskodás motiválta, nincsen egy egzakt és objektív származtatása a w transzformációs súlyoknak. Például, annak a valószínűsége, hogy egy karakter — aminosav vagy nukleotid — nem változott meg az evolúció során, nem 1, így ha transzformációs súlynak a valószínűség logaritmusának a mínusz egyszeresét választjuk, akkor $w(a\to a)$ semmiképpen sem lehet 0. Másfelől a mutációk valószínűségei függenek az eltelt időtől, de éppen ez az, amit szeretnénk meghatározni. Ennek ellenére a távolságalapú módszerek az egyszerűségük és gyorsaságuk miatt a ami napig elterjedtek.

Egy transzformáció sorozatot a szekvenciák illesztésével reprezentálunk. 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 szubsztitúció 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 transzformáció sorozat súlya az egyes pozíciókban történt mutációk súlyainak az összege. Látható, hogy minden mutáció sorozathoz egyértelműen megadható egy illesztés, amely azt reprezentálja, és egy illesztésből a mutációk sorrendjétől eltekintve egyértelműen megadhatóak azok a mutációk, amelyeket az illesztés reprezentál. Mivel az összeadás kommutatív, ezért a teljes súly független a mutációk sorrendjétől. Optimális illesztésnek nevezzük azt az illesztést, amelyhez rendelt mutáció sorozat súlya minimális. Jelöljük $\alpha^*(A_n,B_m)$-mel A_n és B_m optimális illesztéseinek a halmazát, $w(\alpha^*(A_n,B_m))$-mel jelöljük ezen illesztésekhez tartozó mutáció sorozatok súlyát.

A gyors algoritmus az optimális illesztéseket adja meg. Az ötlet az, hogy ha ismerjük $w(\alpha^*(A_{n-1},B_m))$-et, $w(\alpha^*(A_n,B_{m-1}))$-et, valamint $w(\alpha^*(A_{n-1},B_{m-1}))$-et, akkor ebből konstans idő alatt kiszámítható $w(\alpha^*(A_n,B_m))$. Tekintsük ugyanis A_n és B_m egy optimális illesztésének az utolsó illesztett párját! Ezt elhagyva vagy A_{n-1} és B_m vagy A_n B_{m-1} vagy A_{n-1} és B_{m-1} egy

optimális illesztését kapjuk, rendre attól függően, hogy az utolsó pár egy törlést, beszúrást vagy szubsztitúciót reprezentál. Így

$$\begin{aligned} w(\alpha^*(A_n,B_m)) = \min\{ & w(\alpha^*(A_{n-1},B_m))+w(a_n \rightarrow -); \\ & w(\alpha^*(A_n,B_{m-1}))+w(- \rightarrow b_m); \\ & w(\alpha^*(A_{n-1},B_{m-1}))+w(a_n \rightarrow b_m)\} \end{aligned} \tag{3.2.7}$$

A optimális illesztéshez tartozó súlyokat egy ún. dinamikus programozási mátrix, **D** segítségével lehet megadni. Ez egy *n* és egy *m* hosszúságú szekvencia összehasonlítása esetén egy (*n*+1)x(*m*+1)-es táblázat, a sorok és oszlopok indexelése hagyományosan 0-tól *n*-ig illetve *m*-ig megy. A kezdeti feltételek a nulladik sorra és oszlopra:

$$\begin{aligned} d_{0,0} &= 0 \\ d_{0,j} &= \sum_{l=1}^{j} w(- \rightarrow b_l) \\ d_{i,0} &= \sum_{l=1}^{i} w(a_l \rightarrow -) \end{aligned} \tag{3.2.8}$$

A táblázat belsejének a kitöltése a (3.2.7) képlet alapján történik.

A táblázat kitöltésének az ideje O(*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 sarkig, 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 polinomiális időben reprezentálható. Ehhez egy olyan irányított gráfot kell készíteni, amelynek a pontjai a dinamikus programozási táblázat elemei, és egy él megy egy v_1 pontból 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.

3.3 Tetszőleges gap függvény

Mivel egy beszúrást ugyanakkora súllyal értékelünk, mint egy törlést, ezeket közös névvel indel-eknek vagy gap-eknek szokás nevezni, a hozzá tartozó súlyt pedig gap penalty-nek. Általában egy súlyt szoktak használni minden karakter beszúrására és törlésére. Az alapalgoritmus úgy tekinti a szekvenciák változását, hogy egy hosszú gap kialakulását egyes indelek egymásutánjának képzeli. Ez biológiailag helytelen, mert tudjuk, hogy egy hosszabb részszekvenvcia beszúrása vagy törlése egyetlen mutációval is megtörténhet. Így az alapalgoritmus a hosszú indeleket túlságos mértékben bünteti. Ez a felismerés motiválta a gap függvények bevezetését a biológiai szekvenciák analízisében (Waterman et al.,1976). Ez az algoritmus abban különbözik az alapalgoritmustól, hogy egy k hosszúságú gap-et g_k értékkel bünteti. Például az alábbi illesztés súlya $g_2+w(\mathrm{A}\rightarrow\mathrm{G})+g_3+w(\mathrm{G}\rightarrow\mathrm{A})+w(\mathrm{C}\rightarrow\mathrm{G})$

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

Továbbra is azt a minimális súlyú mutáció sorozatot keressük, amely az egyik szekvenciát a másikba alakítja. A dinamikus programozási algoritmus abban különbözik az előző algoritmustól, hogy az illesztés végén állhat egy hosszú gap, így $w(\alpha^*(A_n,B_m))$ kiszámításához nem csak $w(\alpha^*(A_{n-1},B_m))$, $w(\alpha^*(A_n,B_{m-1}))$ és $w(\alpha^*(A_{n-1},B_{m-1}))$ ismeretére van szükség, hanem $w(\alpha^*(A_{n-1},B_{m-1}))$-en kívül minden $w(\alpha^*(A_i,B_m))$, $0\le i<n$-re és minden $w(\alpha^*(A_n,B_j))$, $0\le j<m$-re. Az előbbiekhez hasonlóan belátható, hogy

$$\begin{aligned} w(\alpha^*(A_n,B_m)) = \min\{ & w(\alpha^*(A_{n-1},B_{m-1}))+ w(a_n\rightarrow b_m); \\ & \min_{0\le i<n}\{w(\alpha^*(A_i,B_m))+g_{n-i}\}; \\ & \min_{0\le j<m}\{w(\alpha^*(A_n,B_j))+g_{m-j}\}\} \end{aligned} \tag{3.3.1}$$

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

$$\begin{aligned} d_{0,0}&=0 \\ d_{0,j}&=g_j \\ d_{i,0}&=g_i \end{aligned} \tag{3.3.2}$$

A táblázat belsejét a (3.3.1)-es képlet alapján lehet meghatározni.

A táblázat $d_{i,j}$ elemének a kiszámításához O(i+j) időre van szükség, így a táblázat kitöltése — és így egy optimális illesztés súlya — O(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 gap függvényre bizonyos megkötéseket teszünk, akkor lehetőség van gyorsabb algoritmusok konstruálására. A következő két fejezetben ilyen algoritmusokra mutatok példát.

3.4 Affin gap függvény

Egy gap függvényt affin függvénynek hívunk, ha

$$g_k = uk + v,\ u \geq 0,\ v \geq 0 \tag{3.4.1}$$

Ilyen gap függvény esetén létezik olyan algoritmus, amelynek a futási ideje O(nm) (Gotoh, 1982). Emlékeztetőül, Waterman és munkatársai algoritmusában

$$d_{i,j} = \min\{d_{i-1,j-1} + w(a_n \rightarrow b_m);\ p_{i,j};\ q_{i,j}\} \tag{3.4.2}$$

ahol

$$\begin{aligned} p_{i,j} &= \min_{1 \leq k < i} \{d_{n-k,m} + g_k\} \\ q_{i,j} &= \min_{0 \leq k < j} \{d_{n,m-k} + g_k\}\} \end{aligned} \tag{3.4.3}$$

Az algoritmus ötlete a következő átindexelésben rejlik

$$\begin{aligned} p_{i,j} &= \min\{d_{i-1,j} + g_1,\ \min_{2 \leq k < i} \{d_{n-k,m} + g_k\}\} \\ &= \min\{d_{i-1,j} + g_1,\ \min_{1 \leq k < i-1} \{d_{n-1-k,m} + g_{k+1}\}\} \\ &= \min\{d_{i-1,j} + g_1,\ \min_{1 \leq k < i-1} \{d_{n-1-k,m} + g_k\} + u\} \\ &= \min\{d_{i-1,j} + g_1,\ p_{i-1,j} + u\} \end{aligned} \tag{3.4.4}$$

És hasonlóan

$$q_{i,j} = \min\{d_{i,j-1} + g_1,\ q_{i,j-1} + u\} \tag{3.4.5}$$

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 O(nm) marad, és csak egy konstans faktorral lesz lassabb, mint az alapalgoritmus, amely nem enged meg hosszú indeleket egyetlen mutációs lépésben.

3.5 Konkáv gap függvény

Az affin gap függvény helyességére nincs semmilyen biológiai magyarázat (Gonet et al, 1992; Benner et al, 1993), elterjedt használatát (Thompson et al., 1994) a hozzá tartozó gyors algoritmusnak köszönheti. Meg lehet szabni azonban egy sokkal reálisabb feltételt a gap függvényre (Waterman, 1984), melyre Gotoh algoritmusánál egy kicsit lassabb, de a Waterman és munkatársai köbös idejű algoritmusánál mégis lényegesen gyorsabb algoritmus létezik (Miller and .Myers,1988; Galil and Giancarlo,1989)

Egy gap függvényt konkávnak nevezünk, ha bármely i-re $g_{i+1} - g_i \leq g_i + g_{i-1}$. Magyarán: az egyre növekvő indeleket egyre kevésbé büntetjük. Nyílván, ha a függvény egy adott ponttól csökkenni kezd, akkor elég nagy indelekre 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 (Benner et al., 1993), ha két szekvencia d PAM egység (Dayhoff et al., 1978; Schwarz & Dayhoff, 1979) óta evolválódott, akkor egy q hosszúságú indel súlya

$$35.03 - 6.88 \log_{10} d + 17.02 \log_{10} q \qquad (3.5.1)$$

ami szintén konkáv függvény.

Konkáv gap függvényekre létezik O(nm(log n +log m)) idejű algoritmus. Ez az algoritmus az ún. előretekintő — 'forward looking' — algoritmusok családjába tartozik. Az előretekintő algoritmus tetszőleges gap függvény esetén a következő módon számítja ki a dinamikus programozási algoritmus i-edik sorát:

```
For j:=1 to m do
 begin
  q1[i,j]:=d[i,0]+g[j];
  b[i,j]:=0;
 end;
For j:=1 to m do
 begin
  q[i,j]:=q1[i,j];
  d[i,j]:=min[ q[i,j],p[i,j],d[i-1,j-1]+w(a_i→b_j)];
      {Ennél a lépésnél feltesszük, hogy p[i,j]-t és d[i-1,j-1]-et már kiszámoltuk}
  for j1:=j to m do  {belső ciklus}
      if q1[i,j1]<d[i,j]+g[j1-j] then
         begin
          q1[j1]:=d[i,j]+g[j1-j];
          b[i,j1]:=j;
         end;
 end;
```

ahol $g[]$ a gap függvény, b pedig egy pointer, aminek a jelentése a későbbiekben derül ki. 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 gap értékeket hozzáadva, melyik $q_{i,j1}$-re lehet optimális (belső ciklus). Az aktuális legjobb jelölt $q1[i,j1]$-ben tárolódik, és mire a $j1$-edik cellához érünk, minden szükséges összehasonlítást elvégeztünk. Tehát az előretekintő algoritmus nem gyorsabb az eredeti algoritmusnál, de a koncepció segít a gyorsításhoz.

A gyorsítás alapja a következő észrevétel:

LEMMA 3.5.1 Legyen j az aktuális cella. Ha $d_{i,j}+g_{j1-j} \geq q1[i,j1]$, akkor minden $j2>j1$-re $d_{i,j}+g_{j2-j} \geq q1[i,j2]$.

BIZONYÍTÁS Legyen ugyanis, $k<j<j1<j2$. A feltételből

$$d_{i,j} + g_{j1-j} \geq d_{i,k} + g_{j1-k}. \tag{3.5.2}$$

Adjunk az egyenlőtlenség mindkét oldalához $g_{j2-k} - g_{j1-k}$ -t!

$$d_{i,j} + g_{j1-j} + g_{j2-k} - g_{j1-k} \geq d_{i,k} + g_{j2-k}. \tag{3.5.3}$$

A konkáv gap függvény tulajdonságából

$$g_{j2-j} - g_{j1-j} \geq g_{j2-k} - g_{j1-k} \tag{3.5.4}$$

És ebből átrendezve és felhasználva (3.5.3)-at

$$d_{i,j} + g_{j2-j} \geq. d_{i,j} + g_{j1-j} + g_{j2-k} - g_{j1-k} \geq d_{i,k} + g_{j2-k}. \tag{3.5.5}$$

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 $O(m)$ értéket kellene felülírni minden egyes pozíciónál. Az előző észrevétel egy folyománya azonban már elvezet a kívánt gyorsításhoz.

FOLYOMÁNY 3.5.1 Mielőtt a j-edik cella jelölteket küldene előre (az algoritmus belső ciklusa) a j-edik pozíciótól a pozíciók blokkokra oszthatóak, minden blokknak a b pointere ugyanakkora, és ezek a pointer értékek blokkonként csökkennek balról jobbra haladva.

Az algoritmus a következőképpen működik: minden sorra fenn kell tartani egy változót, amely az aktuális blokkok számát tárolja. Minden blokkra csak egy pointert 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(log m) idő minden egyes cellára.

Az oszlopok esetében teljesen hasonlóan járunk el. Egyetlen rész maradt vissza, ami a teljes algoritmus precíz 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 eltárolunk. Így az algoritmus teljes futási ideje valóban O(nm (log n + log m)).

3.6 Többszörös szekvencia illesztés

Kettőnél több szekvencia egyszerre történő illesztését először Sankoff ismertette (Sankoff, 1975). Az első alkalmazása egy lokális optimalizálás háromszoros illesztéssel volt, mellyel egy bináris fa belső pontjaira adtak meg konszenzus szekvenciákat (Sankoff et al, 1976). Mára a bioinformatika egyik kulcskérdésévé vált a gyors és adekvát többszörös szekvencia illesztés, Dan Gusfield a bioinformatika Szent Gráljának nevezi (Gusfield, 1997). 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 (Hubbard et al., 1996): Amit két homológ szekvencia suttog azt egy többszörös illesztés hangosan kiáltja.

A többszörös illesztés egymás alá írt n-eseit illesztett n-eseknek hívjuk. A többszörös illesztés dinamikus programozási algoritmusa egyszerű általánosítása a páronkénti illesztés algoritmusának: n szekvencia illesztéséhez egy n 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 indelt, és a koordinátatengelyekkel párhuzamos hipersíkok minden kisebb indexű elemét, ha többszörös indeleket megengedünk.

A többszörös szekvencia illesztéssel két alapvető probléma van. Az egyik algoritmuselméleti probléma: az egzakt megoldáshoz szükséges idő a szekvenciák számával exponenciálisan nő. Bebizonyították, hogy a többszörös illesztés NP-teljes probléma (Wang & Jiang, 1994). A másik metodikai probléma: nem világos, hogy 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 (Pages & Holmes, 1998)

Mindkét problémára heurisztikus megoldást ad a fa mentén való iteratív páronkénti illesztés (Feng & Doolittle, 1987; Corpet, 1988; Thompson et al, 1994). Ez a módszer először egy fát konstruál (guide tree, vezérfa) páronkénti távolságokból kiindulva (Hartigan, 1975), majd ezt használja fel többszörös illesztésre. Először a fa alapján szomszédos szekvenciákat illeszti, majd a már illesztett szekvencia párokhoz, hármasokhoz, stb. illeszti az újabb szekvenciákat, szekvencia párokat, hármasokat, stb. úgy, hogy a már illesztett szekvenciák illesztett *n*-eseit nem lehet megbontani, csak egy csupa gap jelekből álló oszlopot beilleszteni. Így *n*-1 páronkénti illesztéssel kapjuk meg *n* szekvencia többszörös illesztését. Sokszor a már illesztett szekvenciákat csak egy ún. profile-lal ábrázolják (Gribskov et al., 1987). Egy profile egy (|Ω|+1)x *l*-es táblázat, ahol *l* az illesztés hossza. Az egyes oszlopokban az adott pozíciójú illesztett *n*-esről készült statisztika található. Az egyes értékek azt mutatják, hogy az ABC adott betűje hány százalékban szerepel az illesztés adott illesztett *n*-esében. Az oszlop utolsó helyén a gap 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 kör 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 alábbi két optimális illesztést:

```
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 másiknál. Például:

```
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 - -
```

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. A közelmúltban megjelent néhány cikk, amely adott felső korlátú közelítést ad többszörös illesztésre (Gusfield, 1993; Ravi & Kececioglu, 1998) Azonban az adott felső határok elfogadhatatlanul nagyok, és sokszor az ezekkel a módszerekkel kapott eredmények lényegesen rosszabbak, mint az egyszerű heurisztikus módszerek eredményei.

Végül meg kell említeni egy új módszert a többszörös illesztésre, amelyik nem dinamikus programozáson alapszik. A DIALIGN gap-mentes homológ részeket keres szekvenciák páronkénti összehasonlításával, majd ezeket a homológ párokat fűzi össze többszörös illesztéssé (Morgenstern et al.,1996; Morgenstern et al., 1998; Morgenstern, 1999). A módszer hátránya az, hogy néha indokolatlanul nagy gap-eket tesz a többszörös illesztésbe, mivel egyáltalán nem bünteti a gap-eket.

3.7 A sarokvágási technika

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 ún. 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 közlöm. Spouge algoritmusa (Spouge, 1989, 1991) általánosítása Fickett (Fickett, 1984), valamint Ukkonen algoritmusainak (Ukkonen, 1984, 1985). A másik algoritmus Gusfieldtől származik (Gusfield, 1997), 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 $d_{i,j} + |(n-i) - (m-j)|*g \leq t$, ahol t a teszt érték, g az indelek büntetése (hosszú indelek 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 $|(n-i) - (m-j)|*g$ értékkel növeli meg az illesztés súlyát. Így, ha t kisebb, 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 $d_{i,j} \leq t$, míg Ukkonen algoritmusában az egyenlőtlenség $|i - j|*g + |(n-i) - (m-j)|*g \leq t$ 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 (Spouge, 1991). Az algoritmus kiterjeszthető konkáv gap függvényekre is.

A k-eltérésű globális illesztés probléma (Gusfield, 1997) 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 bármely út $d_{n,m}$-ből $d_{0,0}$-ba, amelyik legfeljebb k súlyú, nem tartalmaz olyan $d_{i,j}$ pozíciót, amelyre $|i-j|>k/g$. Ekképpen az algoritmus csak azokat a $d_{i,j}$ mátrix elemeket számolja ki, amelyekre $|i-j|<k/g$, 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 $d_{n,m}<k$, 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 $|i-j|<k/g$ 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 n-est a páronkénti összeg — sum-of-pairs — sémával értékelünk (Carillo & Lipman, 1988), azaz

$$SP_k = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d(a_{k,i}, a_{k,j}) \tag{3.7.1}$$

ahol SP_k a k-adik illesztett n-es értékelése, $d(,)$ az $\Omega\cup\{-\}$ halmazon definiált távolságfüggvény, n az illesztett szekvenciák száma. Egy szekvencia k-suffix-e a k-adik betűtől a szekvencia végéig terjedő részszekvencia. Jelöljük $w_{i,j}(k,l)$-lel az i-edik és a j-edik szekvencia k- illetve l-suffix-ének a távolságát. Carillo és Lipman algoritmusa csak azokat a $d_{i_1 i_2,\ldots i_n}$ pozíciókat számolja ki, amelyre

$$d_{i_1,i_2\ldots i_n} + \sum_{j=1}^{n} \sum_{k=j}^{n} w_{j,k}(i_j, i_k) \le t \tag{3.7.2}$$

ahol t a teszt érték. Az algoritmus helyességét az bizonyítja, hogy a szekvenciák még nem illesztett suffix-einek a sum-of-pairs sémával adott optimális illesztésének a súlya nem lehet nagyobb mint a páronkénti illesztésből adódó súlyok összege. A $w_{i,j}(k,l)$-ek a páronkénti illesztésekből számíthatóak, így az algoritmus praktikus idő alatt ki tudja számolni hat darab 200 hosszúságú szekvencia optimális illesztését (Lipman et al., 1989).

4. Maximum likelihood módszerek

4.1 A likelihood függvény

A magyar 'valószínűség' szónak két megfelelője van az angol nyelvben, a probability és a likelihood. Az angol tudományos nyelvben ez a két szó két jól elkülöníthető fogalmat takar. Az értekezés konvenciója az, hogy a 'valószínűség' szót a probability megfelelőjeként használom, a likelihood szót pedig — jobb híján — nem fordítom le.

Tekintsünk egy egyparaméteres sűrűségfüggvényt, például az exponenciális eloszlás sűrűségfüggvényét! Ez az

$$f(x) = \begin{cases} 0 & \text{ha } x \le 0 \\ ke^{-kx} & \text{ha } x > 0 \end{cases} \tag{4.1.1}$$

függvény, ahol $k>0$. Az

$$f(x,k) = ke^{-kx}, \qquad (x,k) \in (0,\infty)\times(0,\infty) \tag{4.1.2}$$

kétváltozós függvény bármely rögzített k-ra egy k paraméterű exponenciális eloszlás sűrűségfüggvényét adja meg, és így

$$\int_0^\infty f(x,k_0)dx = 1 \tag{4.1.3}$$

Ha azt a kérdést tesszük fel, hogy egy adott x_0 esetén melyik k paraméterű eloszlásra lesz a legnagyobb annak a valószínűsége, hogy a valószínűségi változó az $[x_0,x_0+dx]$ intervallumba esik, akkor az $f(x_0,k)$ függvény maximumát kell megkeresni. Ez a függvény azonban nem lehet sűrűségfüggvénye egy valószínűségi változónak, kivéve x_0=1-t, mert

$$\int_0^\infty f(x_0,k)dk = \frac{1}{x_0^2} \tag{4.1.4}$$

és $f(x_0,k)$ nem annak a valószínűségét adja meg, hogy az eloszlás paramétere k volt, precízebben

$$f(x_0,k)dk \neq P(k < \xi < k+dk) \tag{4.1.4}$$

Az $f(x_0,k)$ függvényt hívjuk likelihood függvénynek. Általában azonban nem egyetlen mintavételünk van, n elemű mintára a likelihood függvény

$$L(k;x_1,x_2,\ldots x_n) = \prod_{x_i\in\{x_1,x_2,\ldots x_n\}} f(x_i,k) \tag{4.1.4}$$

A likelihood függvény ebben az esetben is egyváltozós függvény, a változó a k, x_1, x_2,…x_n csak paraméterek, ezért vannak pontosvesszővel elválasztva. A maximum likelihood paraméter érték, azaz az a k, amelyre az $L(k; x_1,x_2,\ldots x_n)$ függvény a maximumát veszi fel, általában konzisztens becslése az eloszlás paraméterének, azaz bármely dk>0-ra

$$\lim_{n\mapsto\infty} P\left(\hat{k}_n - dk < \xi < \hat{k}_n + dk\right) = 1 \qquad (4.1.5)$$

ahol $\hat{k}_n$ egy n elemű mintából számított maximum likelihood becslés.

Általánosságban tehát ha $f(x|\theta)$ függvényben x-et tekintjük változónak, akkor valószínűségről beszélünk, ha a θ paramétert vagy paraméterhalmazt tekintjük változónak, akkor likelihood függvényről beszélünk.

4.2 Szubsztitúciók modellezése

A maximum likelihood módszer igényli a szekvenciák evolúciójának a modellezését: az eljárás során ki kell számolni, hogy mekkora valószínűséggel evolválódott egy szekvencia a másikba adott idő alatt.

A szubsztitúciókat folytonos idejű Markov modellekkel szokták leírni. A Markov modell feltételezi, hogy egy *a* karakter *b*-re való cseréjének a valószínűsége csak *a*-tól és *b*-től függ, és attól nem, hogy az adott site *a* előtt milyen állapotban volt. Mivel a biológiai szekvenciák esetében véges sok karakter van (nukleinsavak esetében 4, aminosavak esetében 20), a Markov modell véges állapotú. Az állapotok számát k jelöli. Egy véges állapotú Markov modellt egy lineáris differenciálegyenlet-rendszerrel lehet megadni, mátrix-vektor formában

$$\frac{d\mathbf{x}}{dt} = \mathbf{Q}\mathbf{x} \qquad (4.2.1)$$

ahol **Q** a modellt leíró mátrix, és **x** tartalmazza az egyes állapotok valószínűségeit. **Q**-ról ki kell kötni, hogy a főátló kivételével minden eleme nem negatív legyen, és minden egyes oszlop összegének 0-nak kell lennie (így a főátlóban nem pozitív számok állnak). Ez a feltétele annak, hogy a valószínűségek összege minden egyes időpontban 1 legyen. Kezdeti érték feltételnek azt a vektort választva, amelynek az i-edik sora 1 és az összes többi eleme 0, a differenciálegyenlet-rendszer megoldása megadja annak a valószínűségét, hogy az i-edik állapotból kiindulva t idő elteltével melyik állapotnak mekkora a valószínűsége. Annak a valószínűségét, hogy i állapotból kiindulva t idő elteltével a j állapotba jutunk, $f_{ij}(t)$ jelöli. A differenciálegyenlet-rendszer megoldása

$$\mathbf{x}(t) = e^{\mathbf{Q}t}\mathbf{x}(0) = \sum_{n=0}^{\infty} \frac{(\mathbf{Q}t)^n}{n!}\mathbf{x}(0) \tag{4.2.2}$$

A megoldás megkereséséhez érdemes a **Q** mátrixot diagonizálni, ha $\mathbf{Q}=\mathbf{V}^{-1}\mathbf{D}\mathbf{V}$, akkor

$$e^{\mathbf{Q}t} = \mathbf{V}^{-1}e^{\mathbf{D}t}\mathbf{V} \tag{4.2.3}$$

ezt pedig már könnyen számítható, mivel

$$e^{\mathbf{D}t} = \begin{pmatrix} e^{\lambda_1 t} & 0 \cdots & 0 \\ 0 & e^{\lambda_2 t} & \\ \vdots & \ddots & \\ 0 & & e^{\lambda_k t} \end{pmatrix} \tag{4.2.4}$$

ahol $\lambda_1, \lambda_2, \ldots \lambda_k$ a **Q** mátrix sajátértékei.

Mivel a **Q** mátrix sorvektorainak az összege a **0** vektor, **Q** rangja nem lehet *k*, így van 0 sajátértéke. Pozitív sajátértéke nem lehet, mert ez ellentmondana a $\sum_{i=1}^{k} x_i = 1$ feltételnek. Ha **Q**-nak csak egyetlen 0 sajátértéke van, akkor létezik egyetlen egyensúlyi állapot, amely globálisan stabilis, azaz bármely $\mathbf{x}(0)\geq\mathbf{0}$-ra, amely teljesíti a $\sum_{i=1}^{k} x_i = 1$ feltételt, a folyamat ebbe az állapotba konvergál. Az *i*-edik állapot egyensúlyi valószínűségét π_i jelöli.

Egy folyamatot reverzibilisnek nevezünk, ha bármely *i*-re és *j*-re

$$\pi_i f_{ij}(t) = \pi_j f_{ji}(t) \tag{4.2.5}$$

A **Q** mátrix túl sok paramétert tartalmaz ahhoz, hogy minden egyes paraméterét a maximum likelihood módszerrel becsüljük meg. Általában $\mathbf{Q}=\mathbf{Q}_0 s$ alakban írják fel, ahol $\mathbf{Q}_0$-ra teljesül, hogy

$$\sum_{i=1}^{k} \pi_i q_{ij} = 1 \tag{4.2.6}$$

azaz egyensúlyi állapotban egységnyi idő alatt átlagosan egy mutáció történik Ezután *st*-t becsülik a maximum likelihood módszerrel. Általános jelenség, hogy az egyes folyamatok rátáját és az evolúciós időt nem lehet külön-külön becsülni, csupán csak a szorzatukat. Ha az evolúciós rátákat a felére csökkentjük, az eltelt időt pedig a kétszeresére növeljük, az immár kétszeres idő eltelte után az egyes állapotok valószínűsége ugyanakkora, mint az eredeti esetben. Úgy is lehetne szemléltetni ezt a jelenséget, hogy egy filmnek az utolsó képkockája nem változik meg attól, hogy felére lassítva játsszuk le, csupán kétszer annyi idő alatt érünk a film végére.

$\mathbf{Q}_0$ mátrix választására sokféle lehetőség van. Az általános Q mátrix DNS adatok esetében (Tavaré, 1986; Rodríguez et al., 1990)

$$\begin{pmatrix} x & \mu a\pi_A & \mu b\pi_A & \mu c\pi_A \\ \mu a\pi_C & y & \mu d\pi_C & \mu e\pi_C \\ \mu b\pi_G & \mu d\pi_G & z & \mu f\pi_G \\ \mu c\pi_T & \mu e\pi_T & \mu f\pi_T & q \end{pmatrix} \tag{4.2.7}$$

ahol *x*, *y*, *z* és *q* olyan értékeket vesz fel, hogy minden oszlop összege 0 legyen. Ekkor az egyensúlyi gyakoriságok valóban π_A, π_C, π_G és π_T. Például a mátrix első sorát ellenőrizve, $x = -\mu(a\pi_C + b\pi_G + c\pi_T)$ és valóban

$$\begin{aligned} &\left\langle \left(x, \mu a\pi_A, \mu b\pi_A, \mu c\pi_A,\right) \middle| \left(\pi_A, \pi_C, \pi_G, \pi_T,\right) \right\rangle = \\ &= -\mu(a\pi_C + b\pi_G + c\pi_T)\pi_A + \mu a\pi_A\pi_C + \mu b\pi_A\pi_G + \mu c\pi_A\pi_T = 0 \end{aligned} \tag{4.2.8}$$

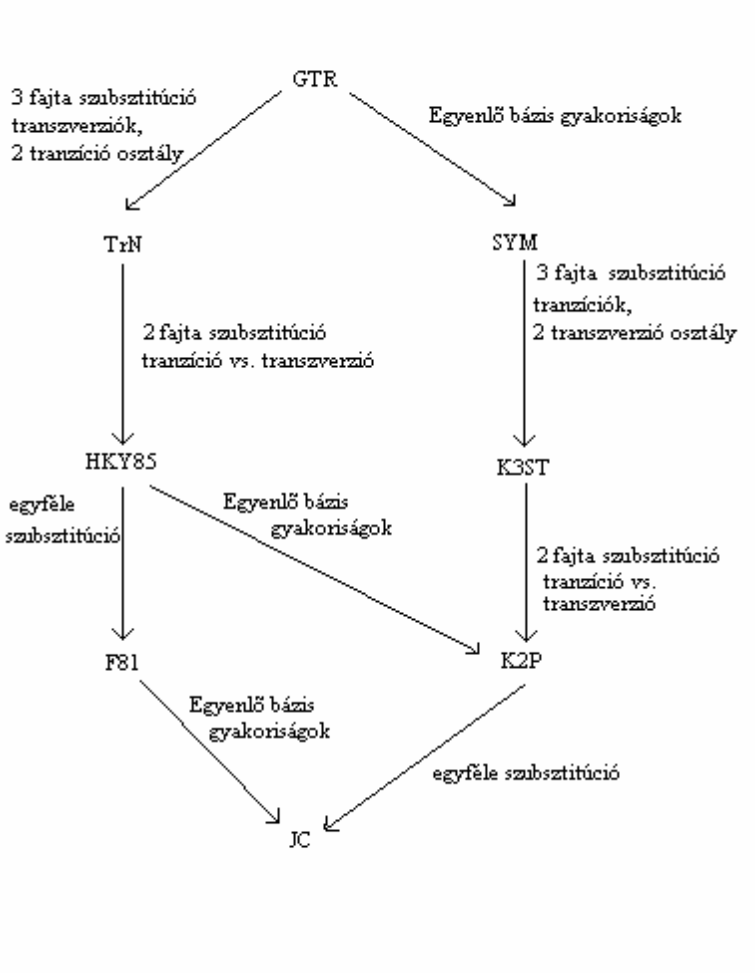
Szintén lehet ellenőrizni, hogy a modell reverzibilis is, a transzponált helyeken álló értékek az egyensúlyi gyakoriságokkal szorozva valóban egyenlők.

Az általános modellre egyre több megkötést téve fokozatosan eljuthatunk a Jukes-Cantor modellhez (Jukes & Cantor, 1969), (**4.2.1** ábra). Ha egyenlő egyensúlyi gyakoriságokat tételezünk fel, a SYM modellt kapjuk (Zarkikh, 1994). Erre tovább feltételezve, hogy a tranzíciók rátája, valamint az A↔T és C↔G, illetve az A↔C és G↔T transzverziók rátája megegyezik, Kimura három paraméteres modelljét kapjuk (Kimura, 1981). Ha minden transzverzió rátáját azonosnak tekintjük, Kimura két paraméteres modelljéhez jutunk (Kimura, 1980), végül a tranziciók rátáját a transzverziók rátájával egyenlővé téve eljutunk a Jukes-Cantor modellhez.

Az általános modellből a Tamura-Nei modellhez jutunk, ha feltesszük, hogy minden transzverzió rátája azonos, *a=c=d=f*, (Tamura & Nei, 1993). A Hasegawa-Kishino-Yano modellben az összes tranzíció rátája is azonos (Hasegawa et al., 1985). Felsenstein 1981-es modelljében minden szubsztitúciós ráta azonos típusú, de az egyensúlyi gyakoriságok különbözőek (Felsenstein, 1981).

Aminosav szekvenciák esetében a Jukes-Cantor modellhez analóg modell a Poisson modell (Kishino et al., 1990), Felsenstein modelljének az analógja pedig a Hasegawa-Fujiwara modell (Hasegawa & Fujiwara, 1993). Bonyolultabb modellekben nem lehetséges a szubsztitúciók típusainak olyan elkülönítése, mint a nukleotidok esetében a tranzíciók és a transzverziók megkülönböztetése. Mégis, empirikus adatok azt mutatják, hogy fizikai-kémiailag hasonló aminosavak szubsztitúciója sokkal gyakoribb, mint különböző aminosavak szubsztitúciója (pl.: egy hidrofób aminosav cseréje hidrofilre) (Dayhoff et al., 1978). A Dayhoff féle mátrixokat sok kutató felhasználja szubsztitúciós modellek készítésére (Kishino et al., 1990; Adachi & Hasegawa, 1992; Hein et al, 2000). Kevéssé rokon fehérjék esetében a

Dayhoff mátrixokat további empirikus adatok segítségével módosítják (Jones et al., 1992; Cao et al., 1994).



4.2.1 ábra Nukleotid szubsztitúciós modellek osztályozása (Hillis et al., 1996). Az általános modellből (GTR, Tavaré, 1986; Rodrígez et al, 1990) egyre több megszorítással jutunk el a Jukes-Cantor modellig (JC, Jukes & Cantor, 1969). További rövidítések: TrN: (Tamura & Nei, 1993), HKY85: (Hasegawa et al., 1985), F81: (Felsenstein, 1981), SYM: (Zharkikh, 1994), K3ST: Kimura 3 szubsztitúció-típusú modellje, (Kimura, 1981), K2P: Kimura 2 paraméteres modellje, (Kimura, 1980)

A modellek jóságát többféle statisztikával lehet ellenőrizni, pl. χ^2 statisztikával, ilyen statisztikák segítségével lehet eldönteni, hogy érdemes-e egy vagy több újabb paramétert bevonni a modellbe (Kishino & Hasegawa, 1990)

4.3 Felsenstein algoritmusa Maximum Likelihood fa meghatározására

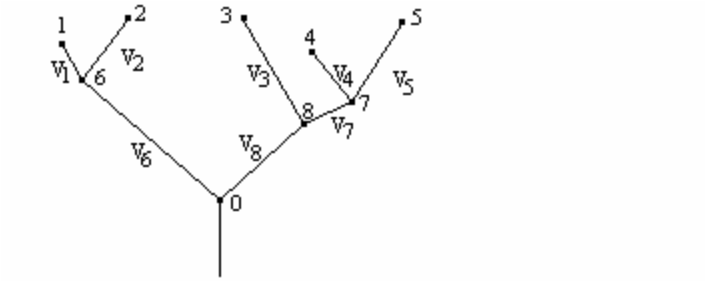
Az alábbiakban Felsenstein egy cikkét ismertetem (Felsenstein, 1981). A bemenő adat DNS szekvenciák többszörös illesztése. Csak azokat a site-okat tekintjük, ahol az illesztésben nincs gap. Feltesszük, hogy az egyes site-ok egymástól függetlenül evolválódtak, így egy evolúciós folyamat valószínűsége az egyes pontokon történt események valószínűségeinek a szorzata. Legyen adva egy fa topológiája, amely reprezentálja a szekvenciák leszármazási sorrendjét. A kérdés, hogy mely élhosszak a maximum likelihood paraméterek. Egy adott paraméterhalmaz mellett a fa likelihood-jának a kiszámítását egy adott fa topológián mutatom meg (**4.3.1** ábra). Elég azt megmutatni, hogy hogyan kell a likelihood-ot egy site-ra kiszámítani, a fa teljes likelihood-ja a site-ok likelihood-jának a szorzata. Az adott site-ra s_i jelöli az i-edik nódusz karakterét, v_j pedig az j-edik él evolúciós ideje, pontosabban a mutációs ráta és az idő szorzata (st). A belső nóduszok állapotait persze általában nem ismerjük, ezért minden lehetséges állapotra összegezni kell

$$L = \sum_{s_0}\sum_{s_6}\sum_{s_7}\sum_{s_8} \pi_{s_0} f_{s_0 s_6}(v_6) f_{s_6 s_1}(v_1) f_{s_6 s_2}(v_2) \\ f_{s_0 s_8}(v_8) f_{s_8 s_3}(v_3) f_{s_8 s_7}(v_7) f_{s_7 s_4}(v_4) f_{s_7 s_5}(v_5) \tag{4.3.1}$$

Ha négyelemű ABC-t, 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 szorzatra, aminek a számítási igénye jóval kisebb

$$L = \sum_{s_0} \pi_{s_0} \left\{ \sum_{s_6} f_{s_0 s_6}(v_6) \left[f_{s_6 s_1}(v_1) \right] \left[f_{s_6 s_2}(v_2) \right] \right\} \\ \left\{ \sum_{s_8} f_{s_0 s_8}(v_8) \left[f_{s_8 s_3}(v_3) \right] \left[\sum_{s_7} f_{s_8 s_7}(v_7) \left(f_{s_7 s_4}(v_4) \right) \left(f_{s_7 s_5}(v_5) \right) \right] \right\} \tag{4.3.2}$$

Vegyük észre, hogy (4.3.2)-ben a zárójelezések pontosan leírják a fa topológiáját. Minden összegzés külön elvégezhető, és ezeket az összegeket szorozzuk össze, így a számítási idő lecsökken $O(|\Omega|n)$-re egy pozícióra, a teljes fa likelihood-jának a kiszámítása $O(|\Omega|nl)$-re, ahol l a pozíciók száma.



4.3.1 ábra A fa, amin Felsenstein algoritmusát bemutatom. Az élekre írt v-k az élek hosszait jelölik.

Ha a szubsztitúciót leíró modell reverzibilis, akkor érvényben van az ún. emelő elv (Pulley Principle). Ez azt mondja ki, hogy ha a fa gyökeréből kiinduló két él közül az egyik hosszát lecsökkentjük, a másikat pedig megnöveljük, akkor a fa likelihood-ja nem változik meg. Valóban, bármely s_6-ra és s_8-ra, ha alkalmazzuk a reverzibilitást — (4.2.5) — valamint a teljes valószínűség tételét

$$L = \sum_{s_0} \pi_{s_0} f_{s_0 s_6}(v_6) f_{s_0 s_8}(v_8) = \sum_{s_0} \pi_{s_6} f_{s_6 s_0}(v_6) f_{s_0 s_8}(v_8) = \tag{4.3.3}$$
$$= \pi_{s_6} f_{s_6 s_8}(v_6 + v_8)$$

tehát az adott összegzés nem függ v_6-tól és v_8-tól, csak a kettő összegétől. Az elnevezés onnan származik, hogy a gyökér mozgása végig a $v_6 + v_8$ hosszúságú élen olyan, mint egy emelő alátámasztása.

Az emelő elv folyománya, hogy egy fa likelihood-ja független attól, hogy hol gyökereztetjük le, és egy legyökereztetett fa likelihood-ja megegyezik a gyökerezetlen fa likelihoodjával.

Egy adott topológia esetén a maximum likelihood paramétereket egyszerű numerikus optimalizálás segítségével meg lehet határozni, de továbbra is fennáll az a probléma, hogy a lehetséges topológiák száma nagyon gyorsan nő a szekvenciák számával. Nevezetesen, a gyökerezetlen topológiák száma $(2n\text{-}5)!/[(n\text{-}3)!2^{n-3}]$ (Cavalli-Sforza & Edwards, 1967) Ez már tíz faj esetén is több mint 2 millió lehetőség. Sajnos a mai napig nem találtak az optimális fa topológiájára gyors algoritmust, az ismert gyors algoritmusok mindegyike heurisztikus, és csak egy lokális optimumot képes megtalálni. A filogenetika körébe tartozó legtöbb probléma NP nehéz (Day, 1983; Foulds & Graham, 1982).

4.4 A Thorne-Kishino-Felsenstein modell

A szekvenciák statisztikus illesztéséhez szükség van a beszúrások és törlések evolúciós modellezésére is. Az első publikált modell erre a folyamatra a Thorne-Kishino-Felsenstein modell volt (Thorne et al., 1991).

A módszer megkeresi azokat az evolúciós paramétereket, amelyre két szekvencia kapcsoltsági valószínűsége maximális. Két szekvencia kapcsoltsági valószínűsége definíció szerint

$$P_t(A,B) = \sum_C P_\infty(C) P_t(A|C) P_t(B|C) \qquad (4.4.1)$$

ahol $P_t(A|C)$ annak a valószínűsége, hogy a C szekvencia t idő alatt A-vá evolválódott., $P_\infty(C)$ pedig a C szekvencia egyensúlyi valószínűsége. A folyamatról feltesszük, hogy egyensúlyban van, így a C szekvencia egyensúlyi valószínűsége megegyezik a C szekvencia t idővel ezelőtti gyakoriságával.

Mivel az alábbiakban leírt folyamat reverzibilis, a (4.3.3) levezetéssel analóg módon kapjuk, hogy

$$P_t(A,B) = P_\infty(A) P_{2t}(B|A) \qquad (4.4.2)$$

azaz a kapcsoltsági valószínűség kiszámításához nem kell az összes lehetséges ősszekvencián összegezni. Megállapodunk, hogy a továbbiakban azt az időt, ami alatt az A szekvencia B-vé evolválódott, t-vel fogjuk jelölni, bár tudjuk, hogy ez a két szekvencia szétválása óta eltelt idő kétszerese.

Az egyszerűség kedvéért a beszúrás és törlés folyamatát nem a karakterek segítségével, hanem ún. képzeletbeli linkek segítségével írjuk le. A szekvencia minden egyes karaktere asszociálva van egy halandó (mortal) linkkel. Ez a link mindig a karakter jobb oldalán található. Ezenkívül a szekvencia rendelkezik egy halhatatlan (immortal) linkkel is, amely a szekvencia bal végén található. Például, ha a halandó link jele $*$, a halhatatlané pedig o, akkor a GACTTAA szekvencia a következőképpen néz ki az adott modellben

o G * A * C * T * T * A * A *

Minden egyes link független a másik linktől, egy link születése vagy halálozása nem befolyásolja egy másik link születésének vagy halálozásának a valószínűségét. Mind a halandó, mind a hallhatatlan link képes halandó linkek szülésére. Az újszülött link mindig a

szülője jobb oldalán helyezkedik el. Egy link születése össze van kapcsolva egy karakter születésével, amihez az újszülött link asszociálódik. Az új karakter mindig az újszülött link bal oldalán helyezkedik el. Az új karakter típusát az egyensúlyi eloszlás szabja meg, azaz egy karaktert akkora valószínűséggel választunk, amekkora az egyensúlyi gyakorisága. A szubsztitúciós folyamatról feltesszük, hogy egyensúlyban van, és az újszülött karakterek ilyen módon történő választása nem zavarja meg ezt az egyensúlyt. A halandó linkek meghalhatnak, míg a halhatatlan nem. Ha egy link meghal, akkor a hozzá asszociált karakter is automatikusan törlődik. Hagyományosan a születési rátát λ-val, a halálozási rátát pedig μ-vel jelöljük.

Többszörös beszúrások és törlések nem lehetségesek ebben a modellben. Egy n hosszúságú szekvencia növekedési rátája $(n+1)\lambda$, mivel egy n hosszúságú szekvencia $n+1$ linkkel rendelkezik (beleszámítva a halhatatlant is). Hasonlóan, egy n hosszúságú szekvencia lecsökken n-1 hosszúságúra $n\mu$ rátával, mivel n halandó linket tartalmaz.

A likelihood kiszámítása nem csak a tranzíciók valószínűségeit kéri, hanem az ősi szekvenciák egyensúlyi valószínűségét is. A Thorne-Kishino-Felsenstein modellben egy adott n hosszúságú szekvencia valószínűsége az egyes karakterek valószínűségeinek a szorzata szorozva annak a valószínűségével, hogy a szekvencia éppen n hosszúságú. Könnyen belátható, hogy a megadott születési-halálozási modell egyensúlyi eloszlása egy eggyel balra elcsúsztatott geometriai eloszlás (a szekvencia 0 hosszú is lehet), ha γ_n jelöli az n hosszúságú szekvencia egyensúlyi gyakoriságát, akkor

$$\gamma_n = \left(1 - \frac{\lambda}{\mu}\right)\left(\frac{\lambda}{\mu}\right)^n \qquad (4.4.3)$$

Az halhatatlan linkre két okból van szükség. Egyrészt ennek a segítségével modellezzük a szekvencia elején történő beszúrásokat, másrészről e nélkül a szekvencia hosszaknak nem lenne egyensúlyi eloszlásuk.

Képzeljünk el két szekvenciát, az A szekvencia legyen TGTC, a B szekvencia pedig GCACA. Számos lehetőség van arra, hogy az A szekvencia B-vé alakuljon. Egy lehetőséget az alábbi α illesztés reprezentál

```
- T G T - C -
G - C - A C A
```

azaz az első, ötödik és hetedik pozícióban egy-egy beszúrás történt, a másodikban és a negyedikben egy-egy törlés, a harmadikban pedig egy szubsztitúció. Jelölje az α illesztésben az egyes karakterek meglétének vagy meg nem létének az információját α'. Ha α'-t a linkek segítségével reprezentáljuk, akkor a következő illesztést kapjuk

```
o - * * * - * -
o * - * - * * *
```

A linkek csoportosíthatóak a fenti reprezentációban, azzal a céllal, hogy meghatározzuk $P(\alpha'|\theta)$-t, ahol θ a paraméterek halmaza, $\theta=\{st, \mu t, \lambda t\}$. Egy tetszőleges tranzíció reprezentálható egy α illesztéssel. Az illesztés valószínűsége a tranzíció valószínűsége szorozva az ősi szekvencia valószínűségével. Az α illesztés valószínűsége két komponensre bontható. Mivel α tartalmazza az összes információt α'-ről, ezért

$$P(\alpha \mid \theta)=P(\alpha, \alpha' \mid \theta)=P(\alpha \mid \alpha', \theta)\, P(\alpha' \mid \theta) \qquad (4.4.4)$$

Általában, ha egy szekvencia n hosszúságú, $P(\alpha' \mid \theta)$ n+2 kifejezésnek a szorzata. Az első kifejezés az ősi szekvencia valószínűsége, a következő kifejezés azt írja le, hogy a halhatatlan linknek hány utódja van, a többi kifejezés pedig a halandó linkek sorsát írja le. Egy adott linkre ez a kifejezés függ attól, hogy a link életben maradt vagy meghalt, valamint a link utódjainak a számától. Mindkettő könnyen leolvasható az α' illesztésből. Egy link utódjainak a száma egy (ha a link túlélt) plusz a jobb oldala és a következő ősi link bal oldala között található leszármazottak száma.

Figyelemmel kísérve az egyes linkek sorsát, három fajta valószínűségi függvényt kell számításba venni: $p_n(t)$ adja meg a valószínűségét annak, hogy egy halandó link t idő elteltével túlél, és saját magát is beleszámítva n utódja van a t-edik időpillanatban. $p_n'(t)$ a valószínűsége annak, hogy egy link meghal t idő alatt, de n utódot hagy maga után, és végül $p_n''(t)$ annak a valószínűsége, hogy a halhatatlan linknek a t-edik időpillanatban n utódja van, önmagát is beleértve. A fenti illesztésre a valószínűségek

$$\begin{aligned} P(\alpha' \mid \theta) &= \gamma_4\ p_2''(t)\ p_0'(t)\ p_1(t)\ p_1'(t)\ p_2(t) \\ P(\alpha \mid \alpha', \theta) &= \pi_G\ \pi_T\ \pi_G\ f_{GC}(t)\ \pi_T\ \pi_A\ \pi_C\ f_{CC}(t)\pi_A \end{aligned} \qquad (4.4.5)$$

Definíció szerint $p_0^{''}(t) = p_0(t) = 0$. A többi valószínűségi függvényt a következő differenciálegyenlet-rendszer írja le

$$\begin{aligned}
&\frac{dp_n(t)}{dt} = \lambda(n-1)p_{n-1}(t) - (\lambda+\mu)np_n(t) + \mu np_{n+1}(t) \qquad n>0\\
&\frac{dp_n^{'}(t)}{dt} = \lambda(n-1)p_{n-1}^{'}(t) - (\lambda+\mu)np_n^{'}(t) + \mu(n+1)p_{n+1}^{'}(t) + \mu p_{n+1}(t) \qquad n>0\\
&\frac{dp_0^{'}(t)}{dt} = \mu p_1^{'}(t) + \mu p_1(t)\\
&\frac{dp_n^{''}(t)}{dt} = \lambda(n-1)p_{n-1}^{''}(t) - \left[\lambda n + \mu(n-1)\right]p_n^{''}(t) + \mu np_{n+1}^{''}(t) \qquad n>0
\end{aligned} \tag{4.4.6}$$

a kezdeti feltételek a következők:

$$\begin{aligned}
p_1(0) &= p_1^{''}(t) = 1\\
p_n(0) &= p_n^{''}(t) = 0, \qquad n = 2, 3, \ldots\\
p_n^{'}(0) &= 0, \qquad n=0, 1, \ldots
\end{aligned} \tag{4.4.7}$$

A (4.4.6) differenciálegyenlet-rendszer a (4.4.7) kezdeti érték feltételekkel megoldható (levezetését lásd a 6.1 fejezetben). A megoldások

$$\begin{aligned}
&p_n(t) = e^{-\mu t}[1-\lambda\beta(t)][\lambda\beta(t)]^{n-1} \qquad n>0\\
&p_n^{'}(t) = [1-e^{-\mu t}-\mu\beta(t)][1-\lambda\beta(t)][\lambda\beta(t)]^{n-1} \qquad n>0\\
&p_0^{'}(t) = \mu\beta(t)\\
&p_n^{''}(t) = [1-\lambda\beta(t)][\lambda\beta(t)]^{n-1} \qquad n>0
\end{aligned} \tag{4.4.8}$$

ahol

$$\beta(t) = \frac{1-e^{(\lambda-\mu)t}}{\mu-\lambda e^{(\lambda-\mu)t}} \tag{4.4.9}$$

Fontos megjegyezni egy lényeges különbséget a konvencionális illesztés és az itt bemutatott illesztés között. Az illesztésre az előbbi példa a következő volt

```
- T G T - C -
G - C - A C A
```

Ha a negyedik és az ötödik pozíciót felcseréljük, a következő illesztést kapjuk

```
- T G - T C -
G - C A - C A
```

Nem világos, hogy a hagyományos értelemben ez a két illesztés miben különbözik egymástól, de az itt bemutatott modell alapján a két illesztés világosan különbözik. Az első illesztésben a negyedik pozícióban található T-hez asszociált link szülte azt a linket, amely A-hoz van

asszociálva. Magyarán A beszúrása mindenképpen megelőzte T törlését. A második illesztésben a negyedik pozícióban található A-hoz asszociált link a harmadik pozícióban található G-hez asszociált link leszármazottja. Ekképpen A beszúrása nem feltétlenül előzte meg T törlését.

Legyen adott két szekvencia, *A* és *B*, rendre *n* és *m* hosszúsággal. Az evolúciós paraméterek az

$$L_\theta(A,B) = \prod_{x\in\Omega} \pi_x^{r_x} \gamma_n P_t(B|A,\theta) \tag{4.4.10}$$

likelihood függvény maximalizálásával határozhatóak meg, ahol r_x az *x* karakterek száma az *A* szekvenciában. A likelihood függvényt dinamikus programozási algoritmussal számítjuk ki. Ez három szempont miatt lehetséges.

- Az egyes linkek egymástól függetlenül evolválódnak. Így $P(\alpha' \mid \theta)$ felbontható az egyes linkek sorsát leíró függvényekre, ahogy az a (4.4.5) képletben be lett mutatva.
- $k \geq 1$-től a (4.4.8)-ban bemutatott képletek felírhatóak az első tag és $\lambda\beta(t)$ segítségével. Azaz

$$\begin{aligned} p_k(t) &= p_1(t)\left[\lambda\beta(t)\right]^{k-1} \\ p_k'(t) &= p_1'(t)\left[\lambda\beta(t)\right]^{k-1} \\ p_k''(t) &= p_1''(t)\left[\lambda\beta(t)\right]^{k-1} \end{aligned} \tag{4.4.11}$$

  Ez a felírás lehetőséget ad O(*nm*) idejű algoritmus készítésére.
- A szekvencia hosszak eloszlása geometriai, ezért minden egyes mortális link az *A* szekvenciában egy $\frac{\lambda}{\mu}$ faktorral járul hozzá a likelihoodhoz. Meg kell azonban jegyezni, hogy ez nem feltétlenül szükséges két szekvencia kapcsoltsági valószínűségének a kiszámításához.

Legyen $S(A_i,B_j)$ A_i és B_j összes lehetséges illesztésének a halmaza. Ezt a halmazt három diszjunkt részhalmazra bontjuk, az utolsó link sorsa alapján

- $S^0(A_i,B_j)$ azon illesztések halmaza, amelyben az A_i szekvencia utolsó linkjének nincs leszármazottja B_j-ben
- $S^1(A_i,B_j)$ azon illesztések halmaza, amelyben az A_i szekvencia utolsó linkjének pontosan egy leszármazottja van B_j-ben

- $S^2(A_i,B_j)$ azon illesztések halmaza, amelyben az A_i szekvencia utolsó linkjének legalább két leszármazottja van B_j-ben

Egy adott θ paraméterhalmaz mellett $\alpha(A_i,B_j)$ illesztés likelihoodját $l_\theta[\alpha(A_i,B_j)]$ jelöli. Az egyes S^i részhalmazok likelihood-ját az alábbiakban definiáljuk

$$L_\theta^k(A_i,B_j)=\sum_{\alpha(A_i,B_j)\in S^{ki}(A_i,B_j)} l_\theta\left[\alpha(A_i,B_j)\right] \qquad k=0,1,2 \tag{4.4.12}$$

Ezek után a dinamikus programozási algoritmus az alábbi szabályokat követi

$$\begin{aligned}
&a)\quad L_\theta^0(A_i,B_j)=\frac{\lambda}{\mu}\pi_{a_i}p_0^{'}(t)\sum_{k=0}^{2}L_\theta^k(A_{i-1},B_j)\\
&b)\quad L_\theta^1(A_i,B_j)=\frac{\lambda}{\mu}\pi_{a_i}\left[f_{a_i,b_j}(t)p_1(t)+\pi_{b_j}p_1^{'}(t)\right]\sum_{k=0}^{2}L_\theta^k(A_{i-1},B_{j-1})\\
&c)\quad L_\theta^0(A_i,B_j)=\lambda\beta(t)\pi_{b_j}\sum_{k=1}^{2}L_\theta^k(A_i,B_{j-1})
\end{aligned} \tag{4.4.13}$$

A kezdeti feltételek

$$\begin{aligned}
&a)\quad L_\theta^0(A_0,B_0)=L_\theta^2(A_0,B_0)=0\\
&b)\quad L_\theta^1(A_0,B_0)=\gamma_0 p_1^{''}(t)\\
&c)\quad L_\theta^0(A_i,B_0)=\gamma_i p_1^{''}(t)\prod_{k=1}^{i}\left(\pi_{a_k}p_0^{'}(t)\right), \qquad i\geq 1\\
&d)\quad L_\theta^1(A_i,B_0)=L_\theta^2(A_i,B_0)=0, \qquad i\geq 1\\
&e)\quad L_\theta^2(A_0,B_j)=\gamma_0 p_{j+1}^{''}(t)\prod_{k=1}^{j}\pi_{b_j}, \qquad j\geq 1\\
&f)\quad L_\theta^0(A_0,B_j)=L_\theta^1(A_0,B_j)=0, \qquad j\geq 1
\end{aligned} \tag{4.4.14}$$

A (4.4.13) és (4.4.14) képletek helyessége könnyen ellenőrizhető, végiggondolva a lehetséges illesztéseket.

- (4.4.13a) A_{i-1} és B_j bármilyen illesztéséhez a_i-t hozzátéve egy olyan illesztést kapunk, amelyben A_i utolsó linkjének nincs leszármazottja. Eközben A-t eggyel megnöveltük
- (4.4.13b) A_{i-1} és B_{j-1} bármilyen illesztéséhez a_i-t és b_j-t hozzátéve egy olyan illesztést kapunk, amelyben A_i utolsó linkjének egy leszármazottja van. Ez vagy A utolsó, túlélt linkje, vagy ennek a valódi leszármazottja. Közben A-t szintén eggyel megnöveltük
- (4.4.13c) Ha egy illesztésben az utolsó linknek legalább két leszármazottja van, akkor az utolsó leszármazottat elhagyva egy olyan illesztést kapunk, amelyben az

utolsó linknek legalább egy leszármazottja van. Az *A* szekvencia hossza nem változik meg, viszont ez a részlikelihood egy $\lambda\beta(t)$ szorzót kap a (4.4.11)-es képlet értelmében.

- (4.4.14a,b) Két üres szekvencia egyetlen módon illeszthető: egymás alá írjuk a két halhatatlan linket. Eszerint az utolsó linknek az illesztésben pontosan egy leszármazottja van.
- (4.4.14c,d) Egy nem üres szekvenciához egy üres szekvenciát egyféleképpen illeszthetünk. A halhatatlan link egyetlen leszármazottja az üres szekvencia halhatatlan linkje, minden halandó linknek — így az utolsónak is — nulla leszármazottja van.
- (4.4.14e,f) Egy üres szekvenciához egy nem üres szekvenciát egyféleképpen illeszthetünk, a leszármazott szekvencia összes linkje az üres szekvencia halhatatlan linkjének a leszármazottja. Így az utolsó linknek legalább két leszármazottja van.

Két szekvencia likelihoodja egyszerűen a

$$L_\theta(A,B) = \sum_{k=1}^{2} L_\theta^k(A,B) \tag{4.4.15}$$

képlettel adható meg.

A maximum likelihood paraméterek kiszámításához nem elegendő egyetlen paraméterlistára kiszámítani két szekvencia likelihoodját, meg kell keresni $L_\theta(A,B)$ függvény maximumát. A három paramétert (st, λt, μt) le lehet csökkenteni kettőre, a

$$\lambda = \frac{\mu(n+m)}{n+m+2} \tag{4.4.16}$$

képlet segítségével. Ez a képlet közvetlenül adódik abból, hogy a minták számtani közepe az eloszlás várható értékének a maximum likelihood becslése.

4.5 A fragmentum modell

A Thorne-Kishino-Felsenstein modell (TKF91) nem enged meg többszörös beszúrásokat és törléseket. Az eredeti modell egy javított változatát egy évvel később publikálták (Thorne et al., 1992), amely megenged többszörös beszúrásokat és törléseket (TKF92). Ez a modell ugyanazt a születési és halálozási folyamatot feltételezi, mint a TKF91 modell, de lehetőséget ad arra, hogy egy link ne egy, hanem tetszőleges számú karakterrel

legyen asszociálva. Ekkor azonban a szekvencia széttörhetetlen fragmentumokból fog állni: ha egy link a születésekor több karakterrel asszociálódott, akkor nincs lehetőség arra, hogy ezek a karakterek külön-külön törlődjenek.

Mégis megmutatható, hogy ez a modell jobb, mint a TKF91 modell. Képzeljük el a következő három szekvencia illesztését:

```
A C G T C G T A T G
A C G T C - - - T G
A C G - - - - A T G
```

Ha a legfelső szekvencia volt az alsó kettő közös őse, akkor a TKF91 modell legalább öt különböző törlési folyamattal tudja leírni ezt a tranzíciót, a TKF92 modellnek elég csak három. Hogy még sem tökéletes ez a modell, azt az mutatja, hogy széttörhetetlen fragmentumok nélkül két törléssel le lehet írni ezt a tranzíciót.

A fragmentumok hossza r paraméterű geometriai eloszlást követ. Megmutatható, hogy a szekvenciák hosszának egyensúlyi eloszlása majdnem geometriai:

$$\gamma_0 = 1 - \frac{\lambda}{\mu}$$
$$\gamma_n = \left(1 - \frac{\lambda}{\mu}\right)\frac{\lambda}{\mu}(1-r)\left[\frac{\lambda}{\mu}(1-r) + r\right]^{n-1} \qquad n \geq 1 \tag{4.5.1}$$

A modell továbbra is reverzibilis, de egy ősi szekvencia valószínűsége függ a szekvencia fragmentáltságától, egy tranzíció valószínűsége — sőt, egyáltalán a lehetősége — függ az ősi szekvencia fragmentáltságától. Két szekvencia likelihoodjának a kiszámításához ezért összegezni kell az összes lehetséges fragmentálódáson és az összes lehetséges tranzíción. Dinamikus programozással ezt is meg lehet tenni. A fragmentumok geometriai eloszlása biztosítja az O(nm) futási idejű algoritmus létezését. Az algoritmus mégis bonyolultabb, mint a TKF91 algoritmusa, a lehetséges illesztéseket hat részre kell osztani, az utolsó link sorsától függően. A dinamikus programozási algoritmus további részleteitől eltekintek.

4.6 A TKF91 modell általánosítása kettőnél több szekvenciára

Kettőnél több szekvencia kapcsoltsági valószínűségének a kiszámításához nem lehet felhasználni a (4.4.2) képletet, szükségképpen összegezni kell az összes lehetséges ősi szekvenciára. Az első ilyen algoritmust Steel és Hein adta meg (Steel & Hein, 2001). Az ő

algoritmusuk tetszőleges számú olyan szekvencia kapcsoltsági valószínűségét számolja ki, amelyek egy csillag alakú fa mentén evolválódtak. Az algoritmust három szekvenciára mutatom be, az általánosítás egyértelmű. Az algoritmus ismertetéséhez az alábbi rövidítések ismertetése szükséges

- Az A szekvencia $A[m]$ prefixén az $a_1 \ldots a_{n-m}$ szekvenciát értjük, ahol n az A szekvencia hossza. Hasonlóan $A(m)$ jelöli az A szekvencia $a_{n-m} \ldots a_n$ részstringjét.
- Az A^1, A^2, A^3 szekvencia hármast röviden **A** jelöli. Az $\mathbf{n}=(n_1, n_2, n_3)$ vektorra $\mathbf{A}[\mathbf{n}]$ jelöli az $A^1[n_1]$, $A^2[n_2]$, $A^3[n_3]$ hármast.
- Egy A szekvenciára $\pi(A)=\prod_{i=1}^{l(A)} \pi(a_i)$.
- $\mathbf{N}=(N_1, N_2, N_3)$, ahol $N_i \in \{0, 1, \ldots l(A^i)\}$ valószínűségi változók mutatják meg, hogy az X ősi szekvencia jobbszélső linkjének hány leszármazottja van az i-edik szekvenciában. Az $U \subseteq \{1,2,3\}$halmaz pedig azt mutatja, hogy az X ősi szekvencia jobbszélső linkje melyik szekvenciákban élt túl.
- Egyszerűen **n**-nel és u-val jelöljük az $\mathbf{N} = \mathbf{n}$ és $U = u$ eseményeket

Először az üres szekvenciák észlelésének a valószínűségét határozzuk meg

$$P(\varnothing,\varnothing,\varnothing) = \left(1-\frac{\lambda}{\mu}\right)\frac{\prod_{i=1}^{3}(1-\lambda\beta(t_i)}{1-\lambda\mu^2\beta(t_1)\beta(t_2)\beta(t_3)} \tag{4.6.1}$$

Ez következik abból, hogy

$$P(\varnothing,\varnothing,\varnothing) = \sum_{l=0}^{\infty} P(\varnothing,\varnothing,\varnothing|l(X)=l)P(l(X)=l) \tag{4.6.2}$$

(4.6.2) egy végtelen geometriai sor, aminek az összege éppen (4.6.1). (4.6.2) persze tetszőleges **A**-ra fennáll

$$P(\mathbf{A}) = \sum_{l=0}^{\infty} P(\mathbf{A}|l(X)=l)P(l(X)=l) \tag{4.6.3}$$

Az $l(X)=0$ esetet külön kezeljük

$$P(\mathbf{A}|l(X)=0) = \prod_{i=1}^{3}\pi(A^i)p''_{l(A^i)}(t_i) \tag{4.6.4}$$

Ezután tegyük fel, hogy $l \geq 1$. A teljes valószínűség tételéből adódóan

$$P(\mathbf{A}|l(X)=0) = \sum_{(\mathbf{n},u)\in I(\mathbf{A})} P(\mathbf{A},\mathbf{n},u|l) \tag{4.6.5}$$

ahol

$$I(\mathbf{A}) := \{((n_1,n_2,n_3),u)\} : 0 \leq n_i \leq l(A^i), i=1,2,3; u \subseteq \{1,2,3\}; n_i = 0 \Rightarrow i \notin u\} \tag{4.6.6}$$

azaz $I(\mathbf{A})$ az a részhalmaza $(\mathbf{n},u)$-nak, amelyre $P(\mathbf{A},\mathbf{n},u|l)$ pozitív értéket vesz fel. A feltételes valószínűség tulajdonságát felhasználva

$$P(\mathbf{A},\mathbf{n},u|l) = P(\mathbf{A}|l,\mathbf{n},u) \times P(\mathbf{n},u|l) \tag{4.6.7}$$

Mivel a TKF91 modellben az egyes linkek sorsa független a többitől,

$$P(\mathbf{A}|l,\mathbf{n},u) = P(\mathbf{A}[\mathbf{n}]|l-1) \times w_1(\mathbf{A},\mathbf{n},u) \tag{4.6.8}$$

ahol

$$w_1(\mathbf{A},\mathbf{n},u) = \left(\prod_{i \notin u, n_i \geq 1} \pi(x_i)\, P(\{i,x_i\} : i \in u) \right) \times \prod_{1 \leq i \leq 3, n_i \geq 2} \pi(A^i(n_i - 2)) \tag{4.6.9}$$

ahol $x_i = a^i_{l(A^i)-n_i+1}$ és $S \subseteq \{1,2,3\}$ esetén $P(\{(i,x_i) : i \in S\})$ annak a valószínűsége, hogy az i-edik nóduszon x_i-t észlelünk, minden $i \in S$-re, használva Felsenstein algoritmusát (Felsenstein, 1981).

Továbbá, mivel $l \geq 1$

$$P(\mathbf{n},u|l) = \prod_{i=1}^{3} p_{n_i}^{|u \cap \{i\}|}(t_i) \tag{4.6.10}$$

ahol $p_k^0(t) := p_k'(t)$ és $p_k^1(t) := p_k(t)$.

Legyen

$$w(\mathbf{A},\mathbf{n},u) = w_1(\mathbf{A},\mathbf{n},u) \prod_{i=1}^{3} p_{n_i}^{|u \cap \{i\}|}(t_i) \tag{4.6.11}$$

Ezután egyesítve a (4.6.5), (4.6.7), (4.6.8) és (4.6.10) képleteket

$$P(\mathbf{A}|l) = \sum_{(\mathbf{n},u) \in I(\mathbf{A})} w(\mathbf{A},\mathbf{n},u)\, P(\mathbf{A}[\mathbf{n}]|l-1) \tag{4.6.12}$$

ebből, valamint a (4.6.3) és (4.6.4) képletekből kapjuk hogy

$$P(\mathbf{A}) = \left(1 - \frac{\lambda}{\mu}\right) \left[\prod_{i=1}^{3} \pi(A^i)\, p''_{l(A^i)}(t_i) \right] + \frac{\lambda}{\mu} \sum_{(\mathbf{n},u) \in I(\mathbf{A})} w(\mathbf{A},\mathbf{n},u)\, P(\mathbf{A}[\mathbf{n}]) \tag{4.6.13}$$

(4.6.13) mindkét oldalán tartalmazza $P(\mathbf{A})$-t, rendezve az egyenletet $P(\mathbf{A})$-ra

$$P(\mathbf{A}) = \frac{\left(1 - \frac{\lambda}{\mu}\right) \left[\prod_{i=1}^{3} \pi(A^i)\, p''_{l(A^i)}(t_i) \right] + \frac{\lambda}{\mu} \sum_{(\mathbf{n},u) \in I^*(\mathbf{A})} w(\mathbf{A},\mathbf{n},u)\, P(\mathbf{A}[\mathbf{n}])}{\left(1 - \frac{\lambda}{\mu} w(\mathbf{A},\mathbf{0},\varnothing)\right)} \tag{4.6.13}$$

ahol $I^*(\mathbf{A}) = I^*(\mathbf{A}) \setminus (\mathbf{0},\varnothing)$. A (4.6.13) képlet lehetőséget ad egy dinamikus programozási algoritmusra, amelynek a futási ideje $O(l^6)$.

A fentiekben leírt algoritmus ki lett terjesztve tetszőleges bináris leszármazási fákra is (Hein, 2001).

4.7 A TKF91 modell gyorsításai

Habár a TKF91 modell alapján végzett statisztikus szekvencia analízis futási ideje O(*nm*), mégis jóval lassabb, mint a távolság/hasonlóság alapú optimális illesztő algoritmusok. Ennek az egyik oka az, hogy a valós számokkal végzett műveletek a legtöbb számítógépen lényegesen lassabbak, mint az egész számokkal végzett műveletek. Az alábbiakban két olyan ötletet mutatok be, amely segítségével a statisztikus szekvencia analízis felgyorsítható (Hein et al., 2000)

Az első ötlet az, hogy a maximum likelihood számításokat lehet a maximálisan szimiláris illesztés körüli régióra korlátozni. Először tehát hasonlóság alapú illesztést végzünk (ehhez lehet egész számokkal számolni, és az lényegesen gyorsabb), majd meg kell határozni egy adott ε-ra a dinamikus programozási táblázatnak azt a részét, amely magában foglalja az összes olyan illesztést, amelynek a hasonlósági értéke legalább (1-ε) része az optimális illesztés értékének. A maximum likelihood számítások erre a régióra vannak korlátozva. Nyílván túl kicsi ε érték választása esetén pontatlanná válik a módszer. Egy kisebb ε értékhez vékonyabb sáv tartozik, amibe nagyobb valószínűséggel nem tartozik bele olyan illesztés, amely likelihood értéke szignifikánsan járul hozzá a teljes likelihoodhoz. Ez különösképpen akkor következhet be, ha a magas likelihood értékeket adó illesztésekben sok beszúrás és törlés található. Az előbbiekből adódóan a módszer várhatóan alul fogja becsülni a λ és μ paraméterek maximum likelihood értékeit alacsony ε érték esetén, amit Hein és munkatársai empirikus eredményei is igazolnak.

A második ötlet az eredeti TKF91 algoritmus átformulázása. Míg az eredeti algoritmusban az összes illesztések halmazát három részre kellett felbontani, és minden egyes részhalmazon külön kell kiszámolni a teljes likelihoodot, addig az átformulázott algoritmusban erre nincs szükség. Így az alább ismertetett algoritmus olyan egyszerű, mint a távolság/hasonlóság alapú optimális illesztést kereső algoritmusok.

Két szekvencia kapcsoltsági valószínűsége felírható

$$P_t(A,B) = P_\infty(A)\,P_{2t}(B|A) \tag{4.7.1}$$

alakban. Mivel $P_\infty(A)$ egyszerűen számítható, a dinamikus programozási algoritmusban csak $P_{2t}(B|A)$-t számoljuk ki. A B_j szekvencia utolsó karakterének a lehetséges leszármazásai alapján

$$P(B_j|A_i) = p_0'(t)P(B_j|A_{i-1}) + \\ + \sum_{k=1}^{j} P(B_{j-k}|A_{i-1})\left[p_k f_{a_i,b_{j-k+1}}(t) \prod_{l=j-k+2}^{j} \pi(b_l) + p_k'(t) \prod_{l=j-k+1}^{j} \pi(b_l) \right] \quad (4.7.2)$$

Ez az algoritmus O(l^3) futási idejű. A futási idő azonban lecsökkenthető O(l^2)-re a Gotoh algoritmusában (Gotoh, 1982) alkalmazotthoz hasonló ötlet alapján. Definiáljuk $R_{i,j}$-t a következőképpen

$$R_{i,j} = P(B_j,\ b_j \text{ leszármazottja } a_i \text{ - nek} | A_i) \quad (4.7.3)$$

$R_{i,j}$ a (4.7.2)-ben az összegzés. Ekképpen (4.7.2) átírható a következő alakba

$$P(B_j|A_i) = p_0'(t)L_\theta(B_j|A_{i-1}) + R_{i,j} \quad (4.7.4)$$

$R_{i,j}$-re teljesül továbbá az, hogy

$$R_{i,j} = \left[p_1(t) f_{a_i,b_j}(t) + p_1'(t)\pi(b_j) \right] P(B_{j-1}|A_{i-1}) + \lambda\beta(t)\pi(b_j)R_{i,j-1} \quad (4.7.5)$$

$R_{i,j-1}$-et (4.7.4)-ből kifejezve, majd az így módosított (4.7.5)-öt (4.7.4)-be beírva és rendezve kapjuk, hogy

$$P(B_j|A_i) = p_0'(t)P(B_j|A_{i-1}) + \lambda\beta(t)\pi(b_j)P(B_{j-1}|A_i) + \\ + \left[p_1(t) f_{a_i,b_j}(t) + p_1'(t)\pi(b_j) - \lambda\beta(t)\pi(b_j)p_0'(t) \right] P(B_{j-1}|A_{i-1}) \quad (4.7.6)$$

Látszik, hogy ebben az algoritmusban a dinamikus programozási táblázat minden egyes elemének a kiszámításához csak ugyanazokra a szomszédos elemekre van szükség, mint a távolság/hasonlóság alapú algoritmusok esetében, csupán most a konstans idő alatt módosított értékeik közül nem a minimálisat/maximálisat kell kiválasztani, hanem a kapott értékeket össze kell adni.