Valószínűségi játékok stratégiakeresési módszerei

Bodon Ferenc

2000. május 21

Köszönetnyilvánítás

Ezúton szeretném köszönetet mondani **dr. Rónyai Lajosnak**, a Budapesti Műszaki és Gazdaságtudományi Egyetem Számítástudományi és Információelméleti Tanszék tanárának az egész munka során nyújtott segítségéért, hasznos ötleteiért, útmutatásáért.

Köszönöm **dr. Horváth Gábornak**, az egyetem Méréstechnika és Információs Rendszerek Tanszék docensének a Mesterséges Intelligencia eszközeinek megismerésében nyújtott segítégét.

További köszönet illeti **dr. Györfi László** és **dr. Tóth Bálint** egyetemi tanárokat valószínűségszámítással kapcsolatos tanácsaikért.

Marx Dánielnek és Surányi Gábornak, az egyetem Műszaki Informatika Szakos hallgatóinak Java, illetve C++ programok elkészítése során felmerülő problémák megoldásainak ötleteiért.

Bereczki Tamásnak, Gillemot Lászlónak, dr. Gohér Anikónak és Tapolcai Jánosnak köszönöm az értékes észrevételeit, megjegyzéseit.

Végül külön köszönöm családtagjaim nélkülözhetetlen támogatását és türelmét.

Tartalomjegyzék

0.1. Diplomaterv Bodon Ferenc mérnökjelölt részére 2

1. Bevezetés **3**
1.1. Valószínűségi játékok 3
1.2. A stratégiakeresés szintjei 3
1.3. Tévhit a rulettel kapcsolatban 4

2. A Black Jack szabályai **7**
2.1. Az eredeti Black Jack 7
2.2. Black Jack variánsok 9

3. Szimulációs stratégiakeresés **10**
3.1. Szimulációs stratégiakeresés alkalmazása a Black Jackre 10

4. Stratégiakeresés megerősítő tanulással **13**
4.1. A megerősítő tanulás 13
4.2. A sztochasztikus megerősítő tanulás 16
4.3. Sztochasztikus megerősítő tanulás alkalmazása a Black Jackre 18

5. Stratégiakeresés a matematika eszközeivel **23**
5.1. Állapotátmenetek valószínűségei 23
5.2. A Black Jack matematikája 24
5.2.1. A megállás várható nyereménye 25
5.2.2. A húzás várható nyereménye 26
5.2.3. A duplázás várható nyereménye 26
5.2.4. A bedobás várható nyereménye 27
5.2.5. A szétszedés várható nyereménye 27
5.2.6. Kiegészítések 27
5.3. Gyorsítási lehetőségek 28
5.3.1. Közelítés végtelen kártyacsomaggal 29
5.3.2. Stratégia-beégetés módszere 30
5.3.3. Közelítés neurális hálóval 32

6. Globális stratégiakeresés **34**
6.1. Az optimális stratégia 34
6.2. Szimulációs várhatóérték képzés 35
6.3. Várhatóérték képzés matematikai eszközökkel 35
6.3.1. A pontos várható érték 35

6.3.2. A Black Jack globális várható értékének kiszámítása 36
6.3.3. Közelítés poli-hipergeometrikus eloszlással 37
6.3.4. Ideális állapotér „zömét" bejáró algoritmus 40
6.3.5. Megvalósítható „zöm"bejáró algoritmus 41
6.3.6. Módosított memóriabarát algoritmus 45
6.3.7. A nyeremény globális várhatóértéke a Black Jack esetében . . 51

7. Összefoglalás **53**

8. Függelék **55**

0.1. Diplomaterv Bodon Ferenc mérnökjelölt részére

A szakirodalom megismerése után ismertesse a jelenlegi stratégiakeresési módszereket. Elemezze azok előnyeit és hátrányait.

Ismertesse a Black Jack stratégiakeresés matematikai megközelítését. Tegyen javaslatot olyan algoritmus kidolgozására amely valószínűségszámításon alapul. Az algoritmust elemezze sebesség szempontjából

Implementáljon olyan programot, amely megvalósít különböző algoritmusokat pontosság-számítási igény alapján.

Határozza meg, hogy az egyes kártyalapok adott valószínűsége mellett mennyi a maximális várható nyeremény, továbbá adjon javaslatot a hosszútávú nyerés megvalósításához.

1. fejezet

Bevezetés

Munka mellett az ember mindig is kikapcsolódásra vágyott, aminek egyik formája a játékokban való részvétel. Habár ennek fő célja a szocializáció, a győzelemre való törekvés szintén fontos szerepet játszik. Sok ember nem elégszik meg a vak szerencse által nyújtott lehetőségekkel, és ezért különböző taktikákkal, stratégiák felállításával próbálják meg levenni a lábáról Fortuna istennőt.

Diplomamunkámban stratégiakereső módszerekről lesz szó. Az egyes módszerek elméleti leírása után, a módszerek konkrét alkalmazására is mutatok példát. Az alkalmazási példák játékok lesznek, de a módszerek érvényességi köre ezzel nem ér véget. Számos fontosabb folyamat modellje felfogható ugyanis valamilyen valószínűségi játéknak.

1.1. Valószínűségi játékok

Minden játékban az egyes játékosok bizonyos számú lehetőség közül választhatnak pl.: táblás játékokban: merre lépjen; kártyajátékokban: mit dobjon; kockajátékokban: milyen tétet tegyen stb. Az akció végrehajtása után új helyzet alakul ki. Amennyiben az adott helyzet, továbbá az akció egyértelműen meghatározza a következő helyzetet, akkor determinisztikus játékról beszélünk. Ilyen pl. a sakk, malom, stb.. Azonban ha az új helyzet ezen kívül valamilyen véletlen tényező függvénye is, akkor az valószínűségi játék. Így tekinthetünk például a kártyajátékokra, ahol egy laphúzásnál a csomagban lévő, nem látható lap határozza meg az új helyzetet. Munkámban az utóbbi típusú játékok stratégiakeresési módszereit mutatom be.

1.2. A stratégiakeresés szintjei

Az általam vizsgált valószínűségi játékok megegyeznek a kaszinókban elérhető játékokkal. Ezekre jellemző, hogy rövid elemi játékok sorozatai. Egy elemi játék a tét elhelyezésétől a tét elvesztéséig, vagy megnyeréséig tart. Ilyen elemi játék például a rulettben egy pörgetés, a kockajátékban egy dobás, vagy a Black Jackben egy kiosztás. Optimális stratégiát ezért két szinten kereshetünk:

Elemi szinten Itt az egyes rövid játékokra jellemző legjobb stratégia, illetve annak nyereményének meghatározása a cél. A tét nagyságának ezen a szinten nincs

szerepe.

Globális szinten Az egyes elemi játékok várható nyeréséről szerzett információ alapján megmondhatjuk milyen tétet tegyünk. A teljes játék tehát tétmanipuláció és elemi játékok sorozata. A globális szinten történő stratégiakeresés a teljes játéksorozatra vonatkozik. Itt tudunk választ adni olyan kérdésekre, hogy érdemes-e egyáltalán játszani, milyen tőkére van szükség, stb..

tétel 1.1. *Amennyiben az elemi játékok egymástól teljesen függetlenek, akkor a globális szinten történő stratégiakeresés értelmetlen, a teljes játék egy tétegységre jutó várható nyereménye független a tétmanipulációtól.*

Bizonyítás. Vegyünk egy tetszőleges véges hosszú elemi játéksorozatot. Jelöljük az i-edik elemi játék várható nyereményét E_i^{elemi}-vel. Ekkor a teljes sorozat várható nyereménye megegyezik az egyes elemi játékok várható nyereményeinek súlyozott összegével, ahol a súly az aktuális tét, tehát:

$$\begin{aligned} E^{\text{globális}} &= \text{tét}_1 \cdot E_1^{\text{elemi}} + \text{tét}_2 \cdot E_2^{\text{elemi}} + \ldots + \text{tét}_n \cdot E_n^{elemi} \\ &= E^{\text{elemi}} \left(\text{tét}_1 + \text{tét}_2 + \ldots + \text{tét}_n\right) \\ &= E^{\text{elemi}} \cdot \text{össztét} \end{aligned} \tag{1.1}$$

a második egyenlőségnél azt használtunk ki, hogy

$$E_1^{\text{elemi}} = E_2^{\text{elemi}} = \ldots = E_n^{\text{elemi}} = E^{\text{elemi}}$$

Tehát a játéksorozat egy egységére jutó várható nyereménye E^{elemi}-ami független a téttől. A teljes sorozat várható nyereménye bármilyen $\text{tét}_1, \text{tét}_2, \ldots, \text{tét}_n$ kombinációra ugyanaz, feltéve, hogy az össztét ugyanaz.

□

Következmény: Nem lehet hosszútávon nyerni olyan játékokban, ahol az elemi játékok függetlenek, és várható nyereményük negatív.

1.3. Tévhit a rulettel kapcsolatban

Az emberek többsége hisz abban, hogy működik az alábbi stratégia: „Tegyünk egy egységet a pirosra, vagy a feketére véletlenszerűen. Ha veszítünk, duplázzuk meg a tétünket, és játszunk újra. A duplázást addig folytassuk, amíg nem nyerünk. Ha nyertünk kezdjünk mindent elölről (azaz tegyünk ismét egy egységet). Ha a tőkém nagy, akkor annak az esélye, hogy annyiszor veszítek egymás után, hogy már nem tudok tovább duplázni, annyira kicsi, hogy majdnem biztos, hogy nyerni fogok. Pl.: ha az össztőkém $2^{10} - 1 = 1023$, akkor 9-szer tudok egymás után duplázni. 10 egymást követő játék elvesztésének az esélye olyan kicsi $\left(\approx \frac{1}{2^{10}} < 0.001\right)$ hogy nyugodtan játszhatok, nyerni fogok"
Ha ez igaz, és a rulett elemi játékának (egy pörgetés) várható nyereménye negatív,

akkor az ellent látszik mondani az előző tétel következményének. Vizsgáljuk meg ezért egy pörgetés várható nyereményét.

A rulettben a 0, 1 , ... 35, 36 számok tetszőleges részhalmazára fogadhatunk (például a pirosra való fogadás a 18 elemű 1, 3, ... 34, 36 részhalmaz kiválasztását jelenti). Amennyiben T nagyságú téttel fogadunk egy n elemű részhalmazra, akkor a kaszinó szabályai szerint a nyerés esetén a tiszta nyerés $ny = T\left(\frac{36}{n} - 1\right)$, vesztés esetén pedig elveszítjük tétünket ($v = -T$). Például, ha pirosra fogadok, \$5-tel, akkor, ha nyerek $5\left(\frac{36}{18} - 1\right) = \$5$-t kapok, ha vesztek, akkor ugyanennyit vesztek. Lévén a teljes halmaz $\{0 \ldots 36\}$ 37 elemű, így a nyerés valószínűsége: $p_{ny} = \frac{n}{37}$, a vesztésé pedig $p_v = 1 - \frac{n}{37}$. Tehát az elemi játék várható nyereménye:

$$\begin{aligned} E^{\text{elemi}} &= p_{ny} \cdot ny + p_v \cdot v \\ &= \frac{n}{37} T \left(\frac{36}{n} - 1\right) - \left(1 - \frac{n}{37}\right) T \\ &= T\frac{36}{37} - n\frac{T}{37} - T - n\frac{T}{37} \\ &= T\frac{-1}{37} \end{aligned} \tag{1.2}$$

Tehát az egy tétegységre jutó várható nyeremény, a részhalmaztól függetlenül mindig $\frac{-1}{37}$. Ezek szerint az, hogy én a pirosra teszem a tételemet, vagy a fekete 24-re, a várható nyereményem szempontjából teljesen mindegy. Felmerülhet a kérdés:„ Akkor most a duplázós módszer nem jó, vagy a tétel nem igaz ?" A válasz az, hogy egyikben sincs hiba. Azt kell látni, hogy az, hogy „99.99% az esélye annak, hogy nyerek" nem mond ellent annak, hogy a várható nyereményem negatív. Könnyű belátni azt, hogy a duplázós módszernél, ha a tőkém véges és így maximum n-szer veszthetek egymás után, akkor igaz, hogy

$$\begin{aligned} E^{\text{globális}} &= p_{ny}^{\text{globális}} \cdot E^{\text{globális}}\,(\text{nyerés}) - p_v^{\text{globális}} \cdot E^{\text{globális}}\,(\text{vesztés}) \\ &= \left[1 - \left(\frac{19}{37}\right)^n\right] \left(2^k - \sum_{i=1}^{k-1} 2^i\right) - \left(\frac{19}{37}\right)^n (2^n - 1) \\ &= \left[1 - \left(\frac{19}{37}\right)^n\right] 1 - \left(\frac{19}{37}\right)^n (2^n - 1) \\ &= 1 - \left(\frac{38}{37}\right)^n \\ &= 1 - \left(1 + \frac{1}{37}\right)^n < 0 \end{aligned} \tag{1.3}$$

Mindezekből következik, hogy nem elegendő a nyerés esélyét meghatározni. Egy stratégia „jóságáról" a várható nyereménye árulkodik. Ez alapján tudunk különböző stratégiákat összehasonlítani, továbbá az optimálisat meghatározni.

Mint azt láttuk, a rulett játékban nincs értelme stratégiát keresni. Éppen ezért a következő fejezetekben bemutatásra kerülő stratégiakereső módszereket olyan játékon illusztrálom, ahol mind az elemi, mind a globális szinten ennek értelme van.

Ez a játék a kaszinók egyik legnépszerűbb játéka, a neve: Black Jack. Nem létezik matematikai bizonyítás arra nézve, hogy az optimális stratégia várható nyereménye

negatív. Éppen ezért az 50-es évektől napjainkig, a játékot folyamatosan kutatják, és jelennek meg könyvek állítólagos egyre jobb stratégiákról[1]. A játék szabályainak komplikáltsága miatt eddig matematikai hátteret vizsgáló eredmények nem születtek. Az eddigi eredmények szimulációkon alapulnak, így sokszor pontatlanok, és gyakran egymásnak ellentmondóak.

Diplomamunkámban ezt a hiányosságot is pótolni kívánom, külön fejezetet szánva a matematikai eszközökkel történő optimális stratégia meghatározásának. Ez a játék fog tehát végigkísérni minket a diplomamunka során. Hogy megértsük a stratégiakeresések alkalmazását, meg kell ismernünk magát a játékot. A játék pontos szabályait a következő fejezetben olvashatjuk.

[1]Csak az Egyesült Államokban 1999 novemberéig bezáróan 68 könyv jelent meg a témában.[8][9][10].

2. fejezet

A Black Jack szabályai

Az eredeti játék Franciaországból származik. Az idők során a szabályai megváltoztak, bonyolódtak. A Black Jacknek ma már egyre több típusa létezik. Nemcsak a különböző kaszinókban, de még az egyes kaszinókon belül is több kicsit módosított szabályrendszerű Black Jacket lehet játszani. A következőkben a Magyarországon legelterjedtebb verziót mutatom be.

2.1. Az eredeti Black Jack

A játékot általában 6 csomag francia kártyával, legalább 1 maximum 7 játékos játszhatja, egy speciális játékos, az osztó (más néven bank) ellen. A játékosok egymástól függetlenül játszanak.

Még a lapok kiosztása előtt minden játékos megteszi a tétjét. Mindenki kap 1 lapot, aztán az osztó, és végül újra egy lap jár minden játékosnak. A 2 kezdőlap kiosztása után minden játékosnak lehetősége van újabb lapokat kérni (HIT). Ekkor nyilván nem láthatja, hogy mit fog kapni. Amennyiben úgy gondolja, hogy lapjainak összege elég magas, akkor megállhat (STAND), azaz nem kér több lapot. Ha lapjainak összege 21 fölé emelkedett, akkor a játszmát elvesztette, nem kérhet több lapot, tétje a banké. Ezután a következő játékos kezdheti el a lapkérést, vagy ha nincs több játékos, akkor a bank.

A banknak kötött szabályai vannak: ha lapjainak összege 17 alatt van, akkor mindig lapot kér, 17-re, vagy e fölött megáll. Ha lapjainak összege 21 fölé emelkedik, akkor ő vesztett, kifizeti az egyes játékosoknak a tétjüket. Ha viszont a bank nem fuccsol be (lapjainak összege < 21), akkor azok a játékosok, akiknek lapösszegük nagyobb a bankénál, nyertek, akiknek kisebb, vesztettek, akiknek pedig egyenlő (PUSH), azok visszavehetik tétjüket.

Hogy ki tudjuk számolni a lapösszeget, tudnunk kell az egyes lapok mennyit érnek:

- nemfigurás lapok (2-10) annyit, amennyi rájuk van írva
- jumbó, dáma, király 10-et
- ász pedig érhet 1-et, vagy 11-et (például, ha valakinek az alábbi lapjai vannak: 5-dáma-ász, akkor az 16, nem pedig 26, de a következő lapot célszerű 20-nak

számítani: 9-ász)

Az alap szabályrendszert, mely meglehetősen egyszerű, az alábbi 5 szabály teszi komplikáltabbá:

1. Amikor a játékosnak még csak 2 lapja van, akkor lehetősége van duplázni. Ekkor be kell tennie a tétjének megfelelő összeget, és kap egy lapot, ami után már nem kérhet továbbiakat.

2. A 2 lapos 21, amit Black Jacknek nevezünk, erősebb a több lapos 21-nél. Például, ha a játékosnak ász-10-ese van, a banknak pedig 10-5-6 akkor a játékos nyer, de nem csak a tétjét, hanem annak másfélszeresét! Fordítva, ha a játékosnak van többlapos 21-e, és a banknak Black Jack-e, akkor a bank nyer, de csak a játékos tétjét (NEM a másfélszeresét)

3. Amennyiben a játékosnak Black Jack-je van, és a bank első lapja ász, akkor a játékos kikérheti a nyereményét, ami azt jelenti, hogy megkapja a tétjét (NEM a másfélszeresét), viszont nem kockáztatja meg, hogy a banknak is Black Jack-je legyen és ne kapjon semmit.

4. A játékos 4. lehetősége a húz, nem húz, dupláz mellett, a szétszedés (SPLIT). Ezt csak akkor választhatja, amikor még csak 2 lapja van, amelyek egyforma értékűek (pl.: 2 db 8-as). Ha tehát a szétszedés mellet dönt, akkor tétjének megfelelő tétet be kell tennie, a 2 lapot szétszedik, újabb 1-1 lapot kap a szétszedett lapok mellé, majd mintha 2 különböző játékos lenne, folytatódhat a játék. Természetesen, ha az így kialakult 2 új kétlaposok között van olyan, akinek megint ugyanolyanok a lapjai, az ismét alkalmazhatja a szétszedést. Hogy érthetőbb legyen, egy példát mutatok: a játékosnak miután 2 db 8-asa volt szétszedett, az első 8-asra kapott egy 10-est és *megállt*, majd a második 8-asra 8-ast kapott, amit ismét szétszedett. Az így szétszedett két 8-asból az elsőre egy 3-as kapott amit megduplázott és egy 7-est kapott. Az utolsó 8-asra egy 6-ost kapott, amire *megállt* választott. Ha a banknak 17-e lett és az alaptét \$2 volt, akkor a játékos $\$2 + 2 \cdot \$2 - \$2 = \$4$-et nyert. A *szétszedés* nagy fegyver a játékos kezében. Erre a kaszinók is rájöttek, ezért az alábbi kiegészítést tették: 2 ászt szét lehet szedni, de csak 1 lapot kap rá a játékos (természetesen ha az újabb ász akkor tovább splittelhet) és ha az új lap 10-es, akkor ez összesen 21-nek, nem pedig Black Jack-nek számít.

5. Amennyiben a bank lapja ász, ami nagyon erős lapnak számít, akkor a játékosok biztosítást köthetnek Black Jack ellen. Ez azt jelenti, hogy be kell tenniük tétjük felét és ha a bank 10-est kap, akkor nem vesztenek semmit. Persze ha nem 10-es a bank következő lapja, akkor a biztosítási összeget azonnal elveszti a játékosok (sőt tétjüknek is búcsút inthetnek, ha lapösszegük kisebb lesz, mint a banké).

A játék során a 6 csomag kártyát sosem osztják ki teljesen, hanem csak körülbelül 80%-át. Ezzel próbálják elkerülni, hogy azok az emberek, akik fejben tudnak tartani 6 csomag kártyát, nyerni tudjanak a játék vége felé. A szakzsargonban a 6

csomag 80%-át SHUE-nak nevezik. Minden SHUE után újrakeverik a paklikat, és folytatódhat a játék. Végül egy, a későbbiekben gyakran használt zsargon: soft x ($11 < x \leq 21$) azt jelenti, hogy egy ász mellett olyan lapok vannak amelyek összege x-11, tehát soft 17 lehet:ász-6, vagy mondjuk ász-ász-2-3

2.2. Black Jack variánsok

Az egyes kaszinók a Black Jack szabályaiban apróbb módosításokat hajtanak végre. Ezek egy része a játékosnak kedvez (pl.: bedobási lehetőség), mások pedig a banknak (pl.: soft 17-re húz). Az alábbi módosítások a legelterjedtebbek:

- A játékos, ha úgy ítéli, hogy veszteni fog, bedobhatja a lapját és így csak a tétje felét veszti el. Ezt csak 2 lapnál teheti meg, amikor a bank lapja NEM ász!
- A bank soft 17-re lapot kér. Ez nagy előny a banknak, kerüljük azokat a kaszinókat, ahol így játsszák a Black Jack-et.

3. fejezet

Szimulációs stratégiakeresés

A szimulációkon alapuló stratégiakeresés a legegyszerűbb, és legkevésbé hatékony stratégiakereső módszer. Az eljárás lényege, hogy az ember kitalál egy stratégiát, amit sok-sok elemi játékon keresztül tesztel, és kiszámítja az egy játékra jutó nyereséget. Ezután változtat a stratégián, majd megint szimulál. Amennyiben az egy játékra jutó nyeremény nőtt, akkor a stratégiát jó irányba módosítottuk, ha pedig csökkent, akkor rossz irányba.

Maga a stratégiakeresés nem áll másból, mint stratégiamódosítás és szimuláció párok sorozatából. Nyilván minél kisebb módosításokat hajtunk végre a stratégián, annál biztosabban, de ugyanakkor annál lassabban haladunk a legjobb stratégia felé. Ennek a módszernek 3 hátránya van:

- Pontatlan: Nyilván minél több játékon keresztül szimulálunk, annál pontosabb eredményt kapunk. Mivel azonban szimulációk sorozatát kell végrehajtanunk, ez nem mindig lehetséges.
- Emberi beavatkozást igényel.
- Lassú.

Felmerül az igény tehát egy olyan stratégiakereső módszerre, ami egyrészt automatikusan, másrészt matematikailag bizonyítottan találja meg az optimális stratégiát. Egy ilyen módszert a következő fejezetben részletezek, most azonban lássuk, hogyan kell alkalmazni a szimulációs stratégiakeresést a Black Jackre.

3.1. Szimulációs stratégiakeresés alkalmazása a Black Jackre

Ahhoz, hogy stratégiakereső módszert tudjunk alkalmazni a Black Jackre, meg kell határoznunk mi is egy stratégia a Black Jack esetén. Általánosan igaz, hogy egy stratégia a lehetséges állapot- és akciótér egy altere. A stratégia minden állapothoz meghatároz egy akciót: azt, amelyiket ki kell választani, ha a játék során ebbe az állapotba jutunk.

A Black Jack esetében ezért a stratégiát el lehet képzelni egy táblázatként, ami a játékos és a bank lapjaihoz definiálja, hogy milyen akciót hajtson végre a játékos. Egy ilyen stratégiatáblázat látható a következő oldalon található ábrán. A

szimulációs stratégiakeresésnél tehát ezt a táblát kell módosítgatni, az egy játékra jutó nyereség nyomon követésével. A szimulációs eredmények azt mutatják, hogy 10 milliós nagyságrendű elemi játékon keresztül kell tesztelni egy stratégiát, hogy az eredmény a tizedes jegy utáni 4 jegyében már ne változzon.

A táblázat jelentése a következő: az oszlop határozza meg a bank lapját, míg a sor a játékos lapjait. Az egyes betűk jelentése pedig:

n	megáll
h	újabb lapot kér
b	bedobja lapjait
D	dupláz, amennyiben ez lehetséges, ha nem (mert kettőnél több lapja van), akkor új lapot kér
d	dupláz, amennyiben ez lehetséges, ha nem (mert kettőnél több lapja van), akkor megáll
s	szétszedi a lapjait

3.1. táblázat. Példa stratégiatáblára a Black Jack esetében

	ász	2	3	4	5	6	7	8	9	10
21	n	n	n	n	n	n	n	n	n	n
20	n	n	n	n	n	n	n	n	n	n
19	n	n	n	n	n	n	n	n	n	n
18	n	n	n	n	n	n	n	n	n	n
17	n	n	n	n	n	n	n	n	n	n
16	h	n	n	n	n	n	h	h	b	b
15	h	n	n	n	n	n	h	h	h	b
14	h	n	n	n	n	n	h	h	h	b
13	h	n	n	n	n	n	h	h	h	h
12	h	n	n	n	n	n	h	h	h	h
11	h	D	D	D	D	D	D	D	D	h

3.2. táblázat. Példa stratégiatáblára a Black Jack esetében

	ász	2	3	4	5	6	7	8	9	10
ász-10	n	n	n	n	n	n	n	n	n	n
ász-9	n	n	n	n	n	n	n	n	n	n
ász-8	n	n	n	n	n	n	n	n	n	n
ász-7	h	n	d	d	d	d	n	n	h	h
ász-6	h	h	D	D	D	D	h	h	h	h
ász-5	h	h	h	D	D	D	h	h	h	h
ász-4	h	h	h	h	D	D	h	h	h	h
ász-3	h	h	h	h	D	D	h	h	h	h
ász-2	h	h	h	h	h	D	h	h	h	h
ász-ász	h	h	h	h	h	h	h	h	h	h
10	h	D	D	D	D	D	D	D	D	h
9	h	h	D	D	D	D	h	h	h	h
8	h	h	h	h	h	h	h	h	h	h
7	h	h	h	h	h	h	h	h	h	h
6	h	h	h	h	h	h	h	h	h	h
5	h	h	h	h	h	h	h	h	h	h
10-10	n	n	n	n	n	n	n	n	n	n
9-9	n	s	s	s	s	s	n	s	s	n
8-8	h	s	s	s	s	s	s	s	s	b
7-7	h	s	s	s	s	s	s	h	h	b
6-6	h	s	s	s	s	s	h	h	h	h
5-5	h	D	D	D	D	D	D	D	D	h
4-4	h	h	h	s	s	h	h	h	h	h
3-3	h	s	s	s	s	s	s	h	h	h
2-2	h	s	s	s	s	s	s	h	h	h
ász-ász	h	s	s	s	s	s	s	s	s	s

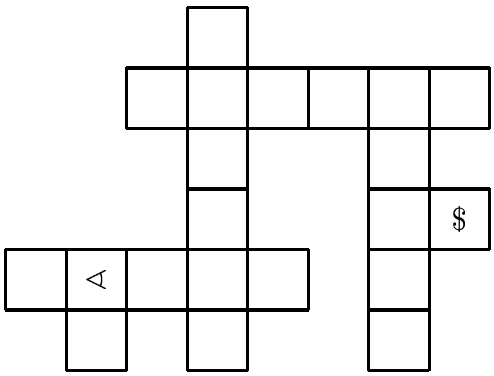
4. fejezet

Stratégiakeresés megerősítő tanulással

A most bemutatásra kerülő stratégiakereső algoritmussal a szimulációs stratégiakeresés hibáit fogjuk kiküszöbölni. Képzeljük el, hogy egy robot leül játszani, órákon keresztül játszik, és tanulva a tapasztalataiból változtatja a stratégiáját. A tanuló algoritmus helyessége miatt, előbb-utóbb eljut az optimális stratégiához. A tanuló algoritmus, amit sok játék esetében hatékonyan lehet alkalmazni, már a 70-es évektől létezik, neve: megerősítő tanulás.[5]

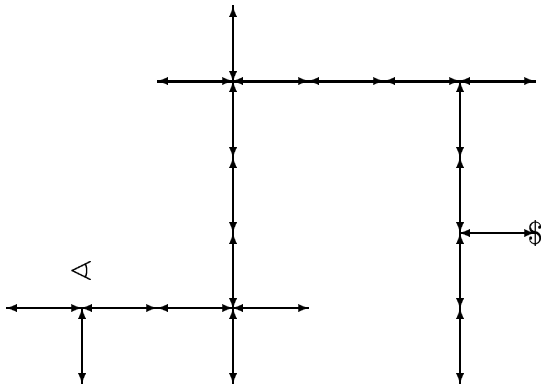
4.1. A megerősítő tanulás

Az érthetőség kedvéért az algoritmus működését egy példán keresztül mutatom be. A példa a következő: adott egy robot egy ismeretlen labirintusban. A robot célja, hogy megtalálja a kijáratot (ezt tovább lehet kombinálni olyanokkal, hogy bizonyos helyekre üzemanyagot teszünk, vagy valami tárgy megtalálásáért plusz pontot kap, stb.). A robot véges számú irányba (pl.: előre-hátra, jobbra-balra) tehet meg egységnyi lépést, majd tovább léphet. A robot napokig próbálkozhat, de ez idő alatt úgy fel kell térképeznie a labirintust, hogy ha bárhova letesszük meg kell találnia a kijáratot (egy ilyen labirintus látható az 4.1 ábrán).



4.1. ábra. Labirintus bejárása

Ezt a matematika nyelvére valahogy így fordíthatnánk le: A labirintus nem más, mint egy irányított gráf, a robot egy-egy helyzetét a gráf egy-egy pontja, az akciókat (pl.: lépés balra) pedig a gráf élei jellemzik. Az él kezdőpontja a robot lépés előtti, míg végpontja a lépés utáni állapota (4.2 ábra).



4.2. ábra. A labirintust reprezentáló irányított gráf

Az alap reinforcement learning algoritmusnál a robot minden lépése után jutalmat kaphat. Ennél a feladatnál nyilván az ésszerű, ha csak a kijárat megtalálásáért (kombinált esetben pl.: üzemanyag felvételért) kap jutalmat. A robot célja, hogy olyan akciópolitikát alakítson ki, amelynél maximális az összegyűjtött jutalom (azaz nincs olyan akciósorozat, amivel több jutalomhoz juthatna). A feladat formális leírására vezessük be az alábbi jelöléseket.

S	állapotok halmaza
A	akciók halmaza
s_t	t-edik időpontban a robot állapota.
r_t	t-edik időpontban a kapott jutalom.
$d(s,a) = s'$	állapotátmenet leíró függvény, jelentése:
	s állapotban az a akció végrehajtása után az új állapot s'
$¶(s_t) = a_t$	a robot politikája (a kiválasztott akció)

Ezek után definiálhatjuk a halmozott jutalmat ¶ stratégiára:

$$V^{¶}(s_t) \equiv r_t + \mu r_{t+1} + \mu^2 r_{t+2} + \dots \tag{4.1}$$

ahol μ egy konstans, amivel azt fejezzük ki milyen arányban áll az azonnali jutalom a késleltetetthez képest. Ha $\mu = 0$ akkor csak az azonnali jutalom számít. Most már formálisan is megfogalmazhatjuk a célunkat: keressük azt a ¶* stratégiát amire igaz:

$$¶^*(s) = \arg\max_{¶} V^{¶}(s), (\forall s) \tag{4.2}$$

Azaz azt a stratégiát keressük, amire a felhalmozott jutalmunk maximális (természetesen minden állapotra, hiszen a robotnak minden pozícióban meg kell tudnia

állapítania, hogy mit tegyen). Definiálhatjuk a maximális jutalmat: $V^* = V^{¶*}$ Alakítsuk át az 4.2 egyenletet:

$$¶^* (s) = \arg\max_{¶} [r (s, a) + \mu V^* (d (s, a))] \tag{4.3}$$

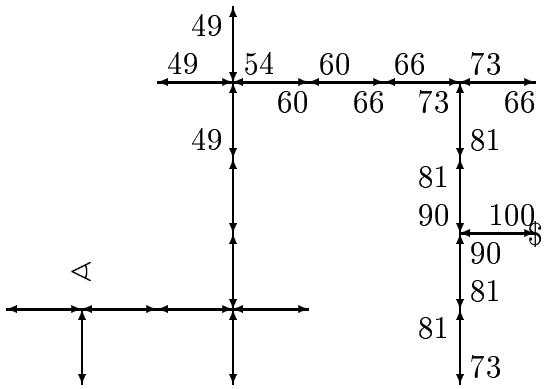
Ez a triviális átalakítás azt jelenti, hogy az optimális stratégiát felbonthatom úgy, hogy először kiválasztom a legjobb lépést, majd a további lépések nem mások, mint a következő állapot optimális stratégiájához tartozó lépések. Definiáljunk most egy „jósági mutatót" az úgynevezett Q-értéket, amit minden állapot-akció párhoz rendelünk az alábbi módon:

$$Q (s, a) \equiv r (s, a) + \mu V^* (d (s, a)) \tag{4.4}$$

Ezt behelyettesítve a 4.3-as egyenletbe:

$$¶^* (s) = \arg\max_{a} Q(s, a) \tag{4.5}$$

Miért fontos ez az átírás? Azért, mert ha a robot megtanulja minden állapot Q-értékét a V* helyett, akkor képes lesz a legjobb akció kiválasztására, még akkor is, ha nincs semmi ismerete az r és d értékekről. Egy állapottér feltérképezése tehát a *Q(s,a)* értékek meghatározásában rejlik. Hogy könnyebben megértsük mindezt, a fenti ábra mutatja a labirintusos példában szereplő néhány Q értéket, azzal a feltétellel, hogy $\mu = 90$, és a robot csak a kijáratnál kap jutalmat, de ott 100 egységet.



4.3. ábra. Néhány q-érték

További kérdés azonban még, hogy hogyan kapjuk meg az egyes Q értékeket, aminek az optimális stratégia összegyűjtött jutalmához szoros köze van, hiszen

$$V^* (s) = \max Q (s, a') \tag{4.6}$$

Ezt felhasználva, újraírhatjuk a 4.4 egyenletet

$$Q (s, a) = r (s, a) + \mu \max Q (d (s, a), a') \tag{4.7}$$

Ez a rekurzív képlet alapjául szolgálhat egy olyan algoritmusnak, amely *Q(s,a)* értékeket iteratívan közelíti. Az algoritmus lényege tehát, hogy kiindulunk egy alaphipotézisből, ami nem más, mint az összes $\hat{Q}(s,a)$ érték valamilyen kezdeti értéke, és az állapottérben való barangolás során az aktuális hipotézisünket folyamatosan javítjuk, a kapott jutalom és az eddigi hipotézis alapján, az alábbi képlet szerint:

$$\hat{Q}^{\text{új}(s,a)} \leftarrow r + \mu \max\left(\hat{Q}^{\text{régi}}(s',a')\right) \tag{4.8}$$

Összefoglalva tehát az algoritmus lépései:

1. Inicializáld az összes állapot-akció pár q-értékét, például q=0-ra.
2. Válassz egy tetszőleges állapotot (s).
3. Hajts végre véletlenszerűen egy akciót (a). Az új állapot: s' = d(s,a)
4. Módosítsd az eredeti Q(s,a) értéket, a fentiek szerint.
5. Amennyiben Q-értékek már csak "kicsit" változtak, vagy a robot politikájának hibája elfogadható tartományba esik, akkor kilépés, ellenkező esetben a ugrás 2-es lépésre.

Az algoritmus bizonyítottan konvergál az optimális megoldáshoz, az alábbi feltételekkel:

- Az állapotok száma véges.
- Létezik egy olyan c konstans, hogy $r(s,a) < c$ minden s,a párra, azaz a jutalmak korlátosak.
- A tanítás alatt minden állapot-akció párt végtelen sokszor választunk (tanítunk).

Az alap q-tanulásnál feltételeztük, hogy minden állapot-akció párnál, mind a jutalom, mind a következő állapot determinisztikus, azaz mindig ugyanakkora jutalmat kapunk és ugyanabba az állapotba jutunk az akció végrehajtása során, adott állapotból. Felmerülhet azonban az igény, hogy a robot akkor is ki tudjon alakítani magának egy jó politikát, ha nem tudjuk sem azt, hogy mekkora jutalmat fogunk kapni, sem azt, hogy melyik állapotba kerül. Így jutunk el a sztochasztikus q-tanuláshoz.

4.2. A sztochasztikus megerősítő tanulás

A sztochasztikus megerősítő tanulás egy eszköz arra, hogy feltérképezzük az állapotteret abban az esetben, amikor egy akció végrehajtása után sem azt nem tudjuk, hogy mekkora jutalmat kapunk, sem azt, hogy melyik új állapotba jutunk, ezek valamilyen nem ismert valószínűségi változók függvényei. Az állapotátmenetek és

a jutalomszerzések valószínűségi értelmezése miatt újra kell definiálnunk egy adott politikához tartozó halmozott jutalmat az alábbiak szerint:

$$V^{\P}(s_t) \equiv E\left[\sum_{i=0}^{\infty} \mu^i r_{t+1}\right] \tag{4.9}$$

Hiszen a stratégiát most már a nyert jutalmak várható értéke jellemzi. Emiatt természetesen változik $Q(s,a)$ definíciója is:

$$\begin{aligned} Q(S,A) &\equiv E\left[r(s,a) + \mu V^*(d(s,a))\right] \\ &= E\left[r(s,a)\right] + \mu E\left[V^*(d(s,a))\right] \\ &= E\left[r(s,a)\right] + \mu \sum_{s'} P(s'|s,a) V^*(s') \end{aligned} \tag{4.10}$$

ahol $P(s'|s,a)$ jelenti azt a valószínűséget, amivel s' állapotba jutunk a akció végrehajtásakor, ha s állapotban vagyunk. Ahogy az eredeti q-tanulásnál, úgy itt is adhatunk Q(s,a)-ra egy rekurzív definíciót:

$$Q(s,a) = E[r(s,a)] + \mu \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a') \tag{4.11}$$

A tanuló algoritmus lépései megegyeznek a determinisztikus eset lépéseivel, pusztán a tanuló képlet változik, az alábbiak szerint:

$$\hat{Q}_n(s,a) \leftarrow (1 - \alpha_n(s,a))\hat{Q}_{n-1}(s,a) + \alpha_n\left[r + \max_{a'}\left(\hat{Q}_{n-1}(s',a')\right)\right] \tag{4.12}$$

ahol $\alpha_n(s,a)$-re teljesülni kell az alábbiaknak:

$$\sum_{i=1}^{\infty} \alpha_i(s,a) = \infty \tag{4.13}$$

$$\sum_{i=1}^{\infty} \left[\alpha_i(s,a)\right]^2 < \infty \tag{4.14}$$

Ezeket kielégítjük, ha a $\alpha_n(s,a)$-re a következő szabályt adjuk:

$$\alpha_n = \frac{1}{1 + \text{visit}_n(s,a)} \tag{4.15}$$

ahol $visit_n(s,a)$ mondja meg, hogy hányszor hajtottuk végre a tanítás során ezt az akciót az n-edik iterációig.

Ahogy a determinisztikus q-tanulás, úgy a sztochasztikus is csak bizonyos feltételek mellett konvergál az optimális megoldáshoz. Ezek a feltételek egyrészt megegyeznek a determinisztikusnál olvashatókkal, másrészt igaznak kell lennie a 4.13 4.14 feltételeknek is.

4.3. Sztochasztikus megerősítő tanulás alkalmazása a Black Jackre

Talán egy kicsit távolinak tűnhet a robot labirintusban való tájékozódásának problémája, egy kártyajáték optimális stratégia keresésétől. Ennek ellenére mindkét feladatnál remekül alkalmazható a reinforcement learning algoritmus.[1]

Hogy megértsük a két példa analógiáját, gondoljuk végig, hogy mi is történik a Black Jack játék során: A játékos kap egy pár lapot, majd végrehajthat valamilyen akciót. Az akció és esetleg a következő lap függvényében a játékos helyzetét egy új lapkombinációval jellemezhetjük, ahol újabb akciók közül választhat. A játék végén ha nyer, pénzt kap, ha veszít, akkor pénzt ad.

Tehát míg a robot állapotát jellemzi a labirintusbeli helyzete, addig a Black Jackben a játékos állapotát jellemzi a bank és a játékos összes lapja. A robot valamilyen akció (pl.: előre, hátra, jobbra, balra mozgás) hatására jut el egy új állapotba, a Black Jacknél pedig az akció lehet: húz, nem húz, dupláz. A robot, ha megtalálja a kijáratot, jutalmat kap, a játékos minden parti után jutalmat kap, ami nem más, mint a pénzösszeg, amit a banknak ad, vagy kap, egységnyi kezdőtét mellet.

Ahhoz, hogy valamilyen problémára alkalmazni tudjuk a megerősítő tanulást, az alábbi lépéseket kell végrehajtanunk:

1. Meg kell határoznunk a probléma állapotterét. Egy probléma leírására több különböző állapotteret is felírhatunk. Mivel a megerősítő tanulás algoritmusának sebessége szoros összefüggésben van az állapottér állapot-akció párjainak számával, ezért azt az állapottér reprezentációt kell választani, ami a legkevesebb ilyen párt tartalmazza. A Black Jack esetében látni lehet majd, hogy a nem minimális állapottér felvételén a módszer gyakorlati alkalmazhatósága múlik.

2. Rögzíteni kell az állapotátmenetek és a jutalomkiosztás szabályrendszerét. Ez determinisztikus esetben azt jelenti, hogy fel kell tudnunk rajzolni az állapotgráfot, sztochasztikus esetben csak a tanulás alatt dől el pontosan, hogy melyik állapotba jut a tanuló (persze szabályt itt is tudnunk kell megfogalmazni).

3. Kezdeti Q értékeket célszerűen fel kell venni.

4. Definiálni kell a leállási feltételeket.

5. Futtatni kell a tanuló algoritmust.

Nézzük ezeket a lépéseket a Black Jack esetében:

Állapottér felvétele Első nekifutásunkban azt mondhatnánk, hogy egy állapotot jellemezhet a bank egy lapja és a játékos összes lapja nagyság szerint sorba állítva (nyilván mindegy, hogy 4-es és egy 5-ös, vagy egy 5-ös és egy 4-es).

[1] A megerősítő tanulás alkalmazhatóságát Black Jackre már más egyetemen is kutatták, az ő modelljükben azonban a játék túlzott leegyszerűsítése miatt az eredmények messze elmaradtak a matematikailag bizonyítható legjobb eredményektől [7]

Ez azonban olyan nagy redundanciát tartalmaz, hogy évekbe telne, míg az eredményt megkapnánk. Hogy megértsük hol van a redundancia, gondoljuk meg hogy az 5-4-3-3 és a 7-6-2 lapkombinációk külön állapotot jellemeznének, holott mindkettőt úgy jellemezhetjük, hogy: "Kettőnél több lapos 15". Felvetődhet ezután, hogy ne a lapok kombinációja jellemezzen egy állapotot, hanem összegük. Ez a megoldáskísérlet viszont 3 hibát rejt magában:

1. az ász érhet 1-et és 11-et is, tehát a 4-ász és a 7-8 lapkombinációk nincsenek megkülönböztetve.
2. két lapot duplázhatunk, vagy bedobhatunk, de többet már nem. Mivel csak az összeget tudjuk, a lapszámról nincs fogalmunk, helytelen útvonalon juthatunk el a célállapotig.
3. két ugyanolyan lapot szétszedhetünk.

Az alapelgondolás, miszerint a lapok összegét tároljuk, mindenképpen jó, tehát induljunk ki ebből, és lássuk, hogyan lehet tovább csiszolni, hogy a 3 hibát kiküszöböljük. Először is vegyünk fel 16 állapotot:

- a játékos lapjainak összege 4
- a játékos lapjainak összege 5
- $\vdots$
- a játékos lapjainak összege 21

Az ász okozta hibát úgy küszöbölhetjük ki, hogy felveszünk újabb 10 állapotot:

- a játékos lapja: ász - ász
- a játékos lapja: ász - a többi lap összege 2
- $\vdots$
- a játékos lapja: ász - a többi lap összege 9
- a játékos lapja: ász - a többi lap összege 10

A duplázásból fakadó problémára egyszerű a megoldás: duplázzuk meg az eddig felvett állapotokat, és az első csoport jelölje azt amikor a játékosnak kettőnél több lapja van, a második csoport pedig azt amikor lapjainak száma kettő. Nyilván a második csoport minden állapotához négy él tartozik, amelyek a húz, nem húz, dupla, bedob akciókat reprezentálják, az első csoport minden állapotához pedig csak kettő él tartozik (húz, nem húz).

Nyilván a szétszedés okozta hibát is hasonlóan (állapotok felvételével) küszöbölhetjük ki. A 10 új állapot a:

- a játékos lapja: ász - ász
- a játékos lapja: 2 - 2
- $\vdots$

- a játékos lapja: 9 - 9
- a játékos lapja: 10 - 10

eseteket jellemzik.

Ezekben az állapotokban a gép csak két akció közül választhat:

1. szétszed
2. nem szed szét

Amennyiben nem szedi szét a lapjait, akkor átugrik a lapösszeg által meghatározott állapotba (tehát például 4-4-es estén a 8-as lapösszeget reprezentáló állapotba). Ellenben ha a szétszedés mellet dönt, akkor a következő két lap határozza meg, hogy melyik két új állapotba kerül. Tanuláskor itt a 4.12 képlet helyett a következő képletet kell alkalmaznunk.

$$\hat{Q}_n\left(s, \text{szétszed}\right) \leftarrow \left(1-\alpha_n\right)\hat{Q}_{n-1}\left(s, \text{szétszed}\right) + \alpha_n\left[r + \max_{a'}\left(\hat{Q}_{n-1}\left(s_1, a'\right)\right) + \max_{a'}\left(\hat{Q}_{n-1}\left(s_2, a'\right)\right)\right] \tag{4.16}$$

ahol s_1,illetve s_2 jelentik az egy-egy új lap után kialakult állapotokat.

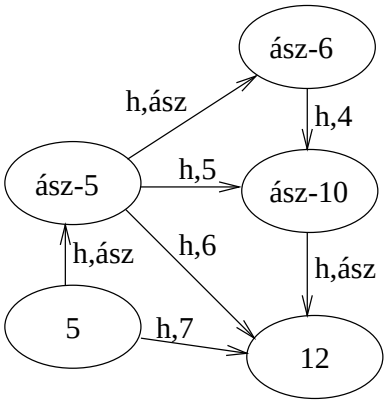
Az állapotok felvételével ezzel még nem vagyunk készen, hiszen a bank lapját nem tároltuk sehol. Tudjuk, hogy a banknak 10 különböző lapja lehet (Ász,2,3, ... 9,10) éppen ezért az eddig felvett állapotok 10-szeresére van szükségünk. Az első csoport jelenti, hogy a banknak ász van, a második, hogy a banknak 2-ese, ... Összesen tehát $1 + 10(2 \cdot 27 + 10) = 641$ állapot szükséges a minimális reprezentációhoz. Az eddig nem tárgyalt extra állapot a végleges elfogadó állapot, ahonnan nem léphet tovább a gép.

Állapotátmenetek és jutalomkiosztás Azt, hogy egy állapotból melyik új állapotba jutunk, a Black Jack szabálya egyértelműen definiálja. Erre mutatok példát a következő oldalon látható ábrán, ahol néhány húzás akcióhoz tartozó él látható (a megáll, dupláz, bedob akciókhoz tartozó élek mindig a végállapotba mutatnak): A jutalomkiosztásra az alábbi szabály igaz: csak azon akciók végrehajtása után adunk jutalmat, amikor a végleges elfogadó állapotba jutunk. Ekkor +1-et, -1-et illetve 0-át, attól függően, hogy nyertünk, vesztettünk, vagy egyenlő lett az eredmény.

Kezdő Q értékek Ha megnézzük a 4.12 képletet, akkor láthatjuk, hogy a módosított Q függ az előző Q értéktől. Tehát a kezdeti értékből kiindulva a Q érték tart az optimális Q értékhez. Ez azt jelenti, hogy minél közelebb van a kezdeti Q érték az optimálishoz, annál gyorsabb a tanulás. A Black Jack esetében a Q értékek +1 és -1 közé fognak esni (mivel az optimális akció várható nyereménye +1 és -1 közé esik), attól függően, hogy mennyire jó az adott akció. Célszerű ezért minden Q értéket kezdetben 0-ra állítani, mert ez egy semleges érték.

Leállási feltétel : általánosan igaz, hogy 3 leállási feltétel típust definiálhatunk:

1. bizonyos tanulólépés után álljon le a tanulás



4.4. ábra. Példa néhány állapotátmenetre

2. ha a Q értékek változása bizonyos korlát alá esik
3. ha a kiadott optimális stratégia bizonyos számú tanulólépés után is változatlan

Ezeket a feltételeket együttesen is használhatjuk

Tanulás Maga a tanulás úgy zajlik, mintha a kaszinóban lennénk: Megkeverjük a képzeletbeli 6 csomag kártyát, majd kiosztunk a játékosnak 2, a banknak 1 lapot. Ez a laphármas egyértelműen meghatároz egy állapotot az állapotgráfunkban. Véletlenszerűen választunk egy akciót (dupla, húz, ...), amennyiben húz, split, dupla akciók közül választottunk, akkor a következő lap függvényében eljutunk egy új állapotba. Ha végül is az elfogadó állapotba kötünk ki, akkor lapot adunk a banknak a saját szabályai szerint, majd kiosztjuk a jutalmat (annak az élnek, amellyel az elfogadóba jutottunk). Természetesen az állapotgráfban való barangolás során a Q-értékeket mindig felülírjuk. Miután elfogadó állapotba jutottunk és kiosztottuk a jutalmat, folytathatjuk a tanítást, oszthatunk új lapokat. Abban az esetben, ha egy általános stratégiára vagyunk kíváncsiak, akkor folytassuk a lapkiosztást a paklik újrakeverése nélkül. Amennyiben viszont adott lapkombináció optimális stratégiáját akarjuk megtudni, akkor a kimenő lapokat tegyük vissza a pakliba és keverjünk majd osszunk újra.

Mint azt az előző részben leírtam, a módszer csak bizonyos feltételek mellett alkalmazható. Mivel az állapottér véges, és a jutalmak korlátosak, továbbá az $\alpha_n(s,a)$-re a 4.15 szabályt alkalmazzuk, ezért a Black Jack esetében minden feltétel teljesül.

Látható, hogy a megerősítő tanulást alkalmazó módszer tényleg kiküszöböli a szimulációs stratégiakeresés két fontos hibáját. Ez a módszer egyrészről nem igényel emberi beavatkozást, másrészről matematikailag megalapozott, hogy a generált stratégia az optimális stratégiához tart.

A módszernek azonban van egy súlyos hibája. Nem mondja meg, hogy ennek az optimális stratégiának mennyi az egy tétegységre jutó nyeresége.

Nyilván megoldás lenne erre egy hibrid módszer, azaz egyszerűen szimuláljuk a megerősítő tanulást alkalmazó módszer által generált stratégiát.

Egy ennél jobb megoldást ad, ha egy kicsit kibővítjük a megerősítő tanulást alkalmazó módszer állapotterét: Vegyünk fel egy kezdő állapotot. Ehhez az állapothoz egyetlen akció fog tartozni, amit hívhatunk úgy, hogy kezdőlap kiosztás. Minden tanulást tehát ebből a kezdeti állapotból indítunk, és a lapkiosztás akciót kell válasszuk, ami a következő három lap függvényében elvezet minket egy új állapotba. Mivel a Q értékek definíciója szerint egy állapot legjobb akciójához tartozó Q érték nem más mint ezen állapotból elérhető maximális jutalom várható értéke, ezért a tanulás végén az új él Q értéke nem lesz más, mint a stratégia várható nyereménye.

A megerősítő tanulást implementáló programot Java nyelven készítettem el. A tanítás milliós nagyságrendű tanítólépés felhasználásával történt. Ezáltal a végeredmény már csak 10%-ban tért el a matematikai megközelítéssel kapott eredményektől. A tanítás alatt megfigyelhető volt, hogy maga az optimális stratégia a legtöbb állapotban már kis számú tanítás után beállt (tízezres nagyságrend), míg bizonyos állapotoknál az optimális akció két akció között oszcillált. A matematikai megközelítésekkel kapott eredmények is igazolták, hogy ezek az akciók azok, amelyek várható nyereményeik nagyon közel esnek egymáshoz, tehát annak eldöntésére, hogy melyik a jobb, sok játékot kell játszani.

A megerősítő tanulás által kapott optimális stratégiát a 11 oldalon lehet látni. Ez az optimális stratégia a játék kezdetéhez tartozó, tehát a $k_1 = 6 * 4 = 24, k_2 = 6*4 = 24 \ldots k_9 = 6*4 = 24, k_{10} = 6*16 = 96$ lapkombinációhoz tartozik, és a tanítás alatt minden játék után újrakevertük a lapokat. Az optimális stratégia várható nyereménye ekkor -0.002157, ami az jelenti, hogy a játék kezdetekor, tehát amikor megkeverték a 6 csomag kártyát és indul az első leosztás, a játékos kedvezőtlen helyzetben van (játszunk tehát minél kisebb téttel).

5. fejezet

Stratégiakeresés a matematika eszközeivel

A megerősítő tanulás felhasználásával tehát bizonyítottan eljutunk az optimális stratégiához, és annak várható nyereményéhez. Az eredmény pontossága azonban összefüggésben van a tanítás lépésszámával. Éppen ezért sosem mondhatjuk azt, hogy az eredmény pontos, legfeljebb azt, hogy 99% annak az esélye, hogy az eredmény a 4. tizedes jegyig pontos. Ha explicit képleteket adnánk, mind az optimális stratégiára, mind annak várható nyereményére, akkor az egy jobb megoldás lenne. Valószínűségszámítást és kombinatorikát felhasználva ilyen képletekhez juthatunk.

5.1. Állapotátmenetek valószínűségei

Ha elmélyedünk egy kicsit a megerősítő tanulás algoritmusában, akkor észrevehetjük, hogy habár a P(s'|s,a) valószínűségeket (tehát annak a valószínűségét, hogy az s állapotból a akció végrehajtása után az új állapot s' lesz) a tanuló robot nem tudja, mégis a Q(s,a) értékeket, elvileg, meg tudja határozni. Ehhez azonban a 4.10 képlet szerint pont az előbb említett P(s'|s,a) ismerete szükséges Ennek oka az, hogy a robot a sok példa (tanulás) alapján „rátanul" ezekre a valószínűségekre. Ezen valószínűségekre való rátanulás bizonyos játékokban (pl.: nyílt kártyás játékok) szükségtelen, hisz meg lehet őket határozni.

A valószínűségszámítás alkalmazásakor tehát feltételezzük, hogy olyan játékokkal foglalkozunk, ahol az állapotátmenetek valószínűségeit ismerjük. Mint azt később látni fogjuk, a felírt képletek bonyolultaknak tűnhetnek, azonban matematikai alapjuk az alábbi két egyszerű tételre épül:

tétel 5.1. *Egy állapotban végrehajtható tetszőleges akció várható nyerése megegyezik azon állapotok optimális stratégiájához tartozó várható nyerések súlyozott összegével, amelyekbe ezen akció végrehajtásával el lehet jutni. A súlyok pedig nem mások, mint az állapotátmenetek valószínűségei, formálisan:*

$$E(s,a) = \sum_{s'} p\left(s'|s,a\right) E\left(s'_{opt}\right) \tag{5.1}$$

ahol s *jelöli a kérdéses állapotot,* a *az akciót,* s' *pedig a szomszédos állapotokat.*

tétel 5.2. *Az előző tétel jelölései mellett az optimális stratégiájához tartozó akciók azok az akciók, amelyekhez tartozó várható nyereség maximális, tehát*

$$a^s_{opt} = \arg\max_a E(s,a) \tag{5.2}$$

amiből következik, hogy

$$E\left(s_{opt}\right) = \max_a E(s,a) \tag{5.3}$$

5.2. A Black Jack matematikája

Célom a Black Jack elemi játékának optimális stratégiája, és annak várható nyereményének meghatározása a valószínűségszámítás felhasználásával.

Mint tudjuk, a játékot 6 csomag francia kártyával játsszák. A játék során mindenki nyílt lapokat kap, így tetszőleges elemi játék (amit leosztásnak hívnak) előtt pontosan tudjuk, hogy hány ász, 2-es, 3-as, ... 9-es, 10-es van a kiosztandó lapok között.

Egy leosztásban először a játékos kap egy lapot, majd az osztó, majd ismét a játékos. Az optimális stratégia meghatározásához vezessünk be két műveletet a Black Jack lapösszegek tere felett. Ennek a térnek az elemei az alábbiak:

$$\{\text{ász}, 2, 3 \ldots 20, 21, \text{fuccs}, \text{soft } 12, \text{soft } 13, \ldots, \text{soft } 21, 10_{\text{egy lap}}, \text{Black Jack}\}$$

Az első művelet az összeadás művelete, amit a továbbiakban $\boxplus$-nal jelölök. Az összeadás műveletét a Black Jack szabályai egyértelműen definiálják, például:

$$\begin{array}{ccc} 3 \boxplus 4 = 7 & \text{soft } 12 \boxplus 4 = \text{soft } 16 & 10 \boxplus \text{ász} = \text{soft } 21 \\ 5 \boxplus \text{ász} = \text{soft } 16 & \text{soft } 14 \boxplus 8 = 12 & 10_{\text{egy lap}} \boxplus \text{ász} = \text{Black Jack} \end{array}$$

A másik művelet a $\uplus$-szal jelölt végösszeg művelet. Két Black Jack lapösszeg végösszege adja meg azt a Black Jack lapösszeget, amit össze kell hasonlítani az ellenfél lapösszegével, tehát

$$\begin{array}{ccc} 3 \uplus 4 = 7 & \text{soft } 12 \uplus 4 = 16 & 10 \uplus \text{ász} = 21 \\ \text{soft } 14 \uplus 8 = 12 & 5 \uplus \text{ász} = 16 & 10_{\text{egy lap}} \uplus \text{ász} = \text{Black Jack} \end{array}$$

Ha meg tudjuk mondani az optimális stratégia e három lap által meghatározott állapotának várható nyereményét, akkor az előző szakasz első tétele alapján ennek az elemi játéknak is meg tudjuk határozni a várható nyereményét, tehát

$$\begin{aligned} E^{opt}_{k_1,\ldots k_{10}} &= \sum_{(j1,b,j2)=(1,1,1)}^{(10,10,10)} p(j1,b,j2) \cdot E^{opt}_{k'_1,\ldots k'_{10}}\left(b,(j1 \boxplus j2)\right) \\ &= \sum_{j1=1}^{10} p(j1) \sum_{b=1}^{10} p(b) \sum_{j2=1}^{10} p(j2) \cdot E^{opt}_{k'_1,\ldots k'_{10}}\left(b,(j1 \boxplus j2)\right) \end{aligned} \tag{5.4}$$

ahol, amennyiben $k_1, k_2, \ldots k_{10}$-el jelöltük a kiosztandó lapokban található ászt, 2-est, ... ,10-est, továbbá $\sum_{i=1}^{10} = K$, akkor

$$p(j1) = \frac{k_{j1}}{K} \text{ , és } p(b) = \begin{cases} \frac{k_b}{K} & \text{ha } b \neq j1 \\ \\ \frac{k_b - 1}{K} & \text{ha } b = j1 \end{cases}$$

$$p(j2) = \begin{cases} \frac{k_{j2}}{K} & \text{ha } j1 \neq j2 \text{ és } b \neq j2 \\ \\ \frac{k_{j2}-1}{K} & \text{ha } j1 = j2 \text{ vagy } b = j2 \text{ de } j1 \neq b \\ \\ \frac{k_{j2}-2}{K} & \text{ha } j1 = b = j2 \end{cases} \tag{5.5}$$

Mivel $k_1, k_2, \ldots k_{10}$ ismert értékek, ezért a feladat $E^{opt}_{k'_1,\ldots k'_{10}}(b, (j1 \boxplus j2))$ meghatározása. A játékos a $(b, (j1, j2))$ lapok által meghatározott állapotban 5 akció (megáll, húz, dupláz, bedob, szétszed) közül választhat. Nyilvánvalóan most az előző szakasz 2. tételének második képletét kell alkalmazni:

$$E^{opt}_{k_1,\ldots k_{10}}(b, (j1 \boxplus j2)) = \max \left\{ \begin{array}{l} E^{\text{megáll}}_{k_1,\ldots k_{10}}(b, J') \,, \\ \\ E^{\text{húz}}_{k_1,\ldots k_{10}}(b, J) \,, \\ \\ E^{\text{dupláz}}_{k_1,\ldots k_{10}}(b, J) \,, \\ \\ E^{\text{bedob}}_{k_1,\ldots k_{10}}(b, J) \,, \\ \\ E^{\text{szétszed}}_{k_1,\ldots k_{10}}(b, J) \end{array} \right\} \tag{5.6}$$

ahol $J' = j1 \uplus j2$ és $J = j1 \boxplus j2$.

Az optimális akció pedig az lesz, amelyhez tartozó várható nyerés maximális (előző szakasz 2 tételének első képlete). Feladat tehát minden egyes akció várható nyerésének meghatározása.

5.2.1. A megállás várható nyereménye

Tudjuk, ha a játékos nem kér több lapot, tehát megáll, akkor az osztó következik, aki kötött szabályok szerint játszik (17-es lapösszeg alatt mindig kér, a fölött soha). Azt is tudjuk, hogy ha az osztó lapösszege nagyobb, mint 21, vagy kisebb, mint a játékos lapösszege, akkor a játékos nyer, ellenkező esetben az osztó.

Mivel elemi stratégiakereső módszereknél az egységtétre jutó várható nyeremény meghatározása a cél, ezért nyerés esetén 1-et, vesztés esetén -1-et kap a játékos. Tehát:

$$E^{\text{megáll}}_{k_1,\ldots k_{10}}(b, J) = \begin{cases} p(j_{\text{köv}}) \cdot E^{\text{megáll}}_{k_1,\ldots k_{10}}(b \uplus j_{\text{köv}}, J) & \text{ha } b < 17 \\ 1 & \text{ha } b < J \text{ vagy } b = \text{fuccs} \\ 0 & \text{ha } b = J \\ -1 & \text{ha } b > J \text{ és } b \neq \text{fuccs} \end{cases} \tag{5.7}$$

ahol $p(j_{\text{köv}}) = \frac{k_{j_{\text{köv}}}}{K}$.

Ha a játékos lapjainak összege nagyobb 21-nél, akkor az osztó már nem kap több lapot, ekkor tehát a fenti képlet helyett az $E^{\text{megáll}}_{k_1,\ldots k_{10}}(b,(J)) = -1$ lesz igaz. Igaz, hogy két lapból a játékos nem mehet 21 fölé, de a későbbiekben is szükség lesz a megállás várható nyereményére, és azokban az esetekben a játékosnak nem csak két lapja lehet. Ezzel meghatároztuk a megállás akcióhoz tartozó várható nyereményt.

5.2.2. A húzás várható nyereménye

Ha a játékos a húzás akciót választja, akkor egy újabb lapot kap és ezáltal egy új állapotba kerül. Alkalmazva az első tételt:

$$\begin{aligned} E^{\text{húz}}_{k_1,\ldots k_{10}}(b,J) =& p(\text{ász}) \cdot E^{opt}_{k_1-1,k_2\ldots k_{10}}(b,(J \boxplus \text{ász})) \\ & p(2) \cdot E^{opt}_{k_1,k_2-1\ldots k_{10}}(b,(J \boxplus 2)) \\ & \vdots \\ & p(10) \cdot E^{opt}_{k_1,k_2\ldots k_{10}-1}(b,(J \boxplus 10)) \\ =& \sum_{i=1}^{10} p(i) \cdot E^{opt}_{k_1,\ldots,k_i-1\ldots k_{10}}(b,(J \boxplus i)) \end{aligned} \tag{5.8}$$

ahol $p(i) = \frac{k_i}{K}$

A játék szabályai szerint, ha egyszer már a játékos kért újabb lapot, akkor onnantól kezdve már nem duplázhat, továbbá be sem dobhatja a lapjait, és a szétszedést sem választhatja. Ebből következik, hogy a $E^{opt}_{k_1,k_{10}}(b,J)$-re adott 5.6 képlet nem igaz, mivel csak két akció közül választhat, tehát

$$E^{opt}_{k_1,\ldots k_{10}}(b,J) = \begin{cases} -1 & \text{ha } J = \text{fuccs} \\ \max\left\{E^{\text{megáll}}_{k_1,\ldots k_{10}}(b,J), E^{\text{húz}}_{k_1,\ldots k_{10}}(b,J)\right\} & \text{ha } J \neq \text{fuccs} \end{cases} \tag{5.9}$$

a helyes.

5.2.3. A duplázás várható nyereménye

Duplázásnál a játékos tétjét megduplázza, de csak pontosan egy lapot kap. Ezután meg kell állnia, az osztó kezdheti meg a laphúzást. Magától értetődik ezért az alábbi képlet:

$$\begin{aligned} E^{\text{dupláz}}_{k_1,\ldots k_{10}}(b,J) =& 2 \cdot p(\text{ász}) \cdot E^{\text{nem húz}}_{k_1-1,k_2\ldots k_{10}}(b,(J \uplus \text{ász})) \\ & 2 \cdot p(2) \cdot E^{\text{nem húz}}_{k_1,k_2-1\ldots k_{10}}(b,(J \uplus 2)) \\ & \vdots \\ & 2 \cdot p(10) \cdot E^{\text{nem húz}}_{k_1,k_2\ldots k_{10}-1}(b,(J \uplus 10)) \\ =& 2 \cdot \sum_{i=1}^{10} p(i) \cdot E^{\text{nem húz}}_{k_1,\ldots,k_i-1\ldots k_{10}}(b,(J \uplus i)) \end{aligned} \tag{5.10}$$

ahol, hasonlóan az előzőkhöz $p(i) = \frac{k_i}{K}$.

5.2.4. A bedobás várható nyereménye

Ha a játékos úgy ítéli meg, hogy nagyon rossz lapja van (pl.: 10-6), de az osztónak jó (pl.:10) akkor bedobhatja a lapját. Ekkor a tétjének felét elveszti. Tehát a lapoktól függetlenül a bedobás akcióhoz tartozó várható nyereménye mindig -0.5, azaz

$$E^{\text{bedob}}_{k_1,\ldots k_{10}}(b,(J)) = -0.5 \tag{5.11}$$

A Black Jack szabályai szerint a játékos csak akkor dobhatja be a lapjait, ha a bank lapja nem ász.

5.2.5. A szétszedés várható nyereménye

A szétszedést csak akkor választhatja a játékos, amikor a kiosztott két lap azonos. Ekkor betesz a tétjének megfelelő összeget, a lapjait kettészedik, mindkettőre kap egy-egy újabb lapot, és mint két külön játékos folytatja a játékot a bank ellen. Ha nagyvonalúak lennénk ezt írhatnánk:

$$E^{\text{szétszed}}_{k_1,\ldots k_{10}}(b,(j1 \boxplus j1)) = 2 * \sum_{i=1}^{10} p(i) \cdot E^{opt}_{k_1,\ldots k_i-1\ldots k_{10}}(b,(j1 \boxplus i)) \tag{5.12}$$

Miért nem igaz a fenti képlet?

Azért, mert egy részről a második játékos várható nyereménye nem lesz azonos az első játékos várható nyereményével (lévén, hogy a lapok valószínűsége megváltozik), továbbá az $E^{opt}_{k_1,\ldots k_{10}}(b,J)$ által felhasznált $E^{\text{nem húz}}_{k_1,\ldots k_{10}}(b,J)$ érték kiszámítására felírt képlet sem lesz igaz, mert ott feltételeztük, hogy a játékos után az osztó következik, ami itt nem teljesül. Szimulációs eredmények igazolták, hogy a fenti képlet jó közelítését adja a valóságnak, ezért a továbbiakban ezt a képletet fogom használni

5.2.6. Kiegészítések

Az eddig felírt képletek nem vettek figyelembe két apró szabályt:

1. A játékos Black Jack esetében két speciális akció közül választhat
2. Két ász szétszedése után, a játékos csak egy-egy lapot kaphat.

Nézzük meg hát, hogyan kell módosítani a képleteinket, hogy most már minden szabálynak megfeleljenek.

Black Jack esetében, ha az osztó lapja nem ász vagy 10-es, akkor a játékos automatikusan megkapja tétjének másfélszeresét. Ha azonban az osztó lapja ász vagy 10-es, tehát van esélye, hogy neki is Black Jackje lesz, akkor két akció közül választhat:

1. Kikéri a nyereségét, ami ebben az esetben csak a téttel, nem annak másfélszeresével egyezik meg.
2. Hagyja, hogy húzzon az osztó, kockáztatva ezzel, hogy neki is Black Jack-e lesz és nem nyer semmit. Ha nem ez következik be, akkor a játékos nyeresége tétjének másfélszerese.

Ezek alapján felírhatjuk, hogy

$$E^{opt}_{k_1,\dots k_{10}}(b,\text{B.J.}) = \begin{cases} \max\left\{E^{\text{kikér}}_{k_1,\dots k_{10}}(b,\text{B.J.}), E^{\text{nem kér ki}}_{k_1,\dots k_{10}}(b,\text{B.J.})\right\} & \text{ha } b \in \{10,\text{ász}\} \\ 1.5 & \text{ellenben} \end{cases} \tag{5.13}$$

ahol

$$E^{\text{kikér}}_{k_1,\dots k_{10}}(b,\text{B.J.}) = 1 \tag{5.14}$$

$$E^{\text{nem kér ki}}_{k_1,\dots k_{10}}(b,\text{B.J.}) = 0 \cdot p(\text{B.J.}|\text{első lap} = b) + 1.5(1 - p(\text{B.J.}|\text{első lap} = b))$$

$$\begin{aligned} &= \begin{cases} 1.5(1-p(\text{ász})) & \text{ha b=10} \\ 1.5(1-p(10)) & \text{ha b=ász} \end{cases} \\ &= \begin{cases} 1.5(1-\frac{k_1}{K}) & \text{ha b=10} \\ 1.5(1-\frac{k_{10}}{K}) & \text{ha b=ász} \end{cases} \end{aligned} \tag{5.15}$$

A két ászra vonatkozó szabálymódosítás miatti új képlet magától értetődő:

$$\begin{aligned} E^{\text{szétszed}}_{k_1,\dots k_{10}}(b,(\text{ász} \boxplus \text{ász})) =& 2 \cdot \Bigg((E^{opt}_{k_1-1,\dots k_{10}}(b,\text{ász} \boxplus \text{ász}) + \\ &+ \sum_{i=2}^{10} p(i) \cdot E^{\text{megáll}}_{k_1,\dots k_i-1\dots k_{10}}(b,(11 \uplus i))) \Bigg) \end{aligned} \tag{5.16}$$

a képletben szereplő $(E^{opt}_{k_1-1,\dots k_{10}}(b,\text{ász} \boxplus \text{ász})$-ra nem alkalmazható a 5.6 képlet, hisz tudjuk, hogy csak a további szétszedés vagy a megállás között választhat a játékos, tehát ebben az esetben:

$$E^{opt}_{k_1,\dots k_{10}}(b,\text{ász} \boxplus \text{ász}) = \max\left\{E^{\text{szétszed}}_{k_1,\dots k_{10}}(b,(\text{ász} \boxplus \text{ász})), E^{\text{megáll}}_{k_1,\dots k_{10}}(b,12)\right\} \tag{5.17}$$

Ezzel minden képletet megadtunk egy tetszőleges $k_1,\dots k_{10}$ lapkombinációhoz tartozó elemi játék optimális stratégiájának és annak várható nyereményének meghatározásához.

5.3. Gyorsítási lehetőségek

Habár a képletek által kapott eredmény pontos, maga a számítás, a rekurzió mélysége miatt, lassú. Az általam implementált programnak egy PII 300Mhz-es processzoron mintegy 32 óráig tartott míg kiadta az optimális stratégiát és annak várható nyereményét.

A globális stratégiakeresésnél majd látni fogjuk, hogy az elemi játék optimális stratégiájához tartozó várható nyeremény kiszámításának gyors műveletnek kell lennie, lévén nagyon sok lapkombinációra kell ezt az értéket meghatározni, hogy a globális kérdésekre választ tudjunk adni.

Feladat lenne ezért, hogy a számítási időt a pontosság rovására nagymértékben lecsökkentsük, törekedve minél jobb közelítés elérésére. A következőkben az alábbi három gyorsítási módszert fogom részletezni:

- közelítés végtelen kártyacsomaggal
- stratégia-beégetés módszere
- közelítés neurális hálóval

5.3.1. Közelítés végtelen kártyacsomaggal

Ennél a közelítésnél feltételezzük, hogy a kiosztandó lapok száma végtelen, de az egyes lapok valószínűségei ismertek. Például az ász valószínűsége $\frac{1}{13}$, a 2-é $\frac{1}{13}$ ... a 10-é $\frac{4}{13}$.

Hogy lássuk mekkora eltérést ad ez a közelítés, gondoljunk például arra, hogy mekkora a valószínűsége annak, hogy két ászt húzok, végtelen sok lap közül, és egy csomag kártya = 52 lap közül:

$$p_{\text{végtelen}}(2 \text{ ász}) = \frac{1}{13} \cdot \frac{1}{13}$$

$$p_{\text{egy csomag}}(2 \text{ ász}) = \frac{4}{52} \cdot \frac{3}{51} = \frac{1}{13} \cdot \frac{3}{51} \approx p_{\text{végtelen}}(2 \text{ ász}) \cdot 0.76$$

Mitől lesz több nagyságrenddel gyorsabb a számítás?

Attól, hogy nem kell nyomon követnünk azt, hogy milyen lapok húzásával jutottunk el a bank és a játékos lapösszege által meghatározott állapotig, hiszen a kezdetben megállapított lapvalószínűségek egyértelműen meghatározzák a várható nyereményt. Az tehát, hogy egy adott állapotot két 3-as és egy 2-es, vagy pedig egy 8-as lap húzásával értem el, teljesen mindegy mivel laphúzások által a lapok valószínűségei nem változnak (nem úgy, mint véges pakli esetében). A gyorsítás tehát ott jelentkezik, hogy elég minden állapot várható nyereményét egyszer kiszámítani, ellentétben a véges paklis esettel, ahol ezt annyiszor kellett megtenni, ahányféleképpen azt az állapotot a kezdeti állapotból el lehet érni.

Felrajzolhatunk tehát egy irányított, körmentessé tehető, úgynevezett szabálygráfot. Ennek a gráfnak a pontjai reprezentálják a játékos és az osztó egy lapja által meghatározott állapotot, éleihez pedig a lapokat rendeljük, úgy, hogy az él kezdőpontjának állapotához az él lapját hozzáadva a végpont által reprezentált állapotba jutunk

Az egyes pontokhoz tartozó várható nyeremény megegyezik a szomszéd pontokhoz tartozó várható nyeremények súlyozott összegével. A súly nem más, mint a szomszéd állapotba mutató él lapjának valószínűsége. Az elemi játék várható nyereményének meghatározása ekkor, a gráf egyszeri topologikus bejárásával megkapható.

A topologikus bejáráshoz a gráfnak körmentesnek kell lennie[4]. A körmentességet csak a szétszedés akciójához tartozó él sérti meg, hiszen ahogy a 5.12 képlet mutatja az $E^{\text{szétszed}}_{k_1,\ldots k_{10}}(b,(j1 \boxplus j1))$ meghatározásához ismernünk kell $E^{opt}_{k_1,\ldots k_i-1\ldots k_{10}}(b,(j1 \boxplus j1))$ értékét, ami viszont függ a szétszedés várható nyereményétől. Végtelen kártyacsomagot feltételezve, azonban igaz, hogy ha a legjobb akció a lapok szétszedése, akkor annak az állapotnak is a szétszedés lesz a legjobb akciója, amit az eredeti állapotból

egy újabb ugyanolyan lap húzásával kapok. Éppen ezért:

$$E^{\text{szétszed}}_{p_1,\ldots p_{10}}(b,(j1 \boxplus j1)) = 2 \cdot \sum_{i=1}^{10} p_i \cdot E^{opt}_{p_1,\ldots p_{10}}(b,(j1 \boxplus i)) =$$

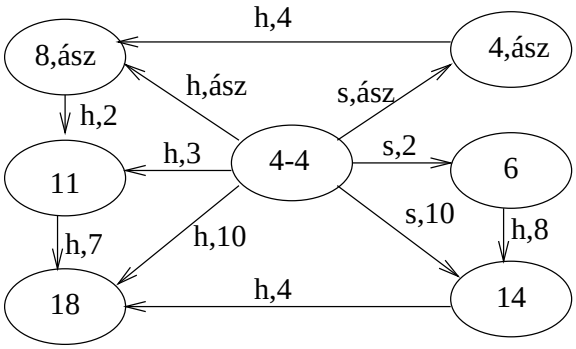
$$2 \cdot p_{j1} \cdot E^{opt}_{p_1,\ldots p_{10}}(b,(j1 \boxplus j1)) + 2 \sum_{i=1,i\neq j1}^{10} p_i \cdot E^{opt}_{p_1,\ldots p_{10}}(b,(j1 \boxplus i))$$

$$2 \cdot p_{j1} \cdot E^{\text{szétszed}}_{p_1,\ldots p_{10}}(b,(j1 \boxplus j1)) + 2 \sum_{i=1,i\neq j1}^{10} p_i \cdot E^{opt}_{p_1,\ldots p_{10}}(b,(j1 \boxplus i))$$

$$(1-2p_{j1}) \cdot E^{\text{szétszed}}_{p_1,\ldots p_{10}}(b,(j1 \boxplus j1)) = 2 \sum_{i=1,i\neq j1}^{10} p_i \cdot E^{opt}_{p_1,\ldots p_{10}}(b,(j1 \boxplus i))$$

$$E^{\text{szétszed}}_{p_1,\ldots p_{10}}(b,(j1 \boxplus j1)) = \frac{2}{1-2p_{j1}} \cdot \sum_{i=1,i\neq j1}^{10} p_i \cdot E^{opt}_{p_1,\ldots p_{10}}(b,(j1 \boxplus i)) \tag{5.18}$$

Látható, hogy a jobb oldalon már nem szerepel olyan állapot várható nyereménye, ami függ a szétszedés várható nyereményétől, tehát a gráfot körmentessé tettük. A következő ábrán a szabálygráf egy apró részlete látható, néhány húzás és szétszedés akcióhoz tartozó éllel.



5.1. ábra. A szabálygráf egy részlete

Az eljárás nem használ rekurziót, az általam készített implementációban egy 2 dimenziós ($40 \times 10 - \text{es}$) tömb sorindex (azon belül oszlopindex) szerinti feltöltésével a végeredmény egy tízezred másodperc nagyságrendű idő alatt megkapható.

Ha végtelen kártyacsomagos közelítéssel az elemi játék várható nyereményét határozzuk meg, akkor a hiba viszonylag nagy (5-20 %, a közelítendő lapkombináció lapszámától függően). Célszerű tehát, csak bizonyos rekurziós szint felett, az egyes állapotok várható nyereményének meghatározásához használni.

5.3.2. Stratégia-beégetés módszere

A módszernél a lehetséges akciók terét szűkítjük. Ennek oka az, hogy vannak olyan akciók, melyeknek várható nyereménye „sosem” lesz maximális, viszont ezen értékek

meghatározása sok időt vesz igénybe. Ha ezeket az akciókat kivágjuk a szabálygráfból, akkor a program futása gyorsabb lesz. Kétféle akcióelvágást különbözetetek meg:

logikus következtetés alapján. Ekkor tudom, hogy a kiiktatandó akción kívül biztosan létezik egy olyan akció, amely várható nyereménye nagyobb. Például, ha a játékos lapjainak összege 11 alatt van, akkor a megállás akció várható nyereménye biztos kisebb, mint a húzás várható nyereménye, hiszen egy újabb lappal csak jobb állapotba kerülhetek.

kis valószínűség alapján. Ilyenkor ha annak a valószínűsége, hogy egy akció várható nyereménye lesz maximális, bizonyos korlát alá esik, akkor az akciót kivágjuk a képzeletbeli szabálygráfunkból. Az általam implementált programban egy akciót akkor iktatok ki, ha bizonyos számú véletlenszerűen generált $k_1, k_2 \ldots k_{10}$ lapkombináció között nincs olyan eset (tehát nem maga az állapot által meghatározott lapkiosztás), amikor a vizsgált akció lenne a maximális. A generált lapkombinációk száma megegyezik egy emberöltő (100 év) alatt, a kaszinóban folyamatosan generált lapkombinációk számával. Magyarán, ha egy olyan ember sem találkozik ilyen lapkombinációval, aki 100 éven keresztül Black Jacket játszik, akkor elhanyagolhatjuk ezt az akciót ebben az állapotban.

A gyorsítások következtében a 32 órás futási időből, a stratégia-beégetés módszerét alkalmazva 14 perc lett, ami a végtelen kártyacsomagos közelítés következtében 1-5 mp-re csökkent, attól függően, hogy az eltérésnek 5%, vagy 2%-os korlátot írtam elő.

Az általam implementált program, amely az optimális stratégia és annak várható nyereménye meghatározásához matematikai eszközöket használt, a következő eredményeket adta. A 6 csomag megkeverése után, az első leosztásban a játékos várható nyereménye negatív, pontosabban, a három közelítéssel kapott eredmény:

5.1. táblázat. Az első leosztás várható nyereménye

$E^{opt}_{24\ldots24,96}$=-0019634	59,1 perces futási idő
$E^{opt}_{24\ldots24,96}$=-002091	2 mp futási idő
$E^{opt}_{24\ldots24,96}$=-0028453	0.00014 mp futási idő

Mindhárom közelítés ugyanazt az optimális stratégiát generálta. Ezt a stratégiát ezután a szimulációs programommal 180 millió játékon (6 óra 45 percen) keresztül teszteltem. Az egy játékra jutó nyeremény -0.0021655[1] lett, ami csak mintegy 7%-os

[1]A bevezetőben láthattuk, hogy a rulett játékban a játékos tétjének $\frac{-1}{37}$-e a bank zsebébe vándorol, minden egyes pörgetésnél. A kaszinók által készített statisztika alapján ez az érték a Black Jack esetében $\frac{-1}{20}$, ami azt jelenti, hogy az emberek a Black Jackben átlagosan másfélszer többet vesztenek. Ez arról tanúskodik, hogy az emberek gyengén játszanak, az általuk kidolgozott stratégia messze elmarad az optimálistól, ahol a várható nyeremény $\frac{-1}{500}$ (Sok ember úgy játszik, hogy sosem kér újabb lapot, ha lapjainak összege 11 fölé emelkedik. Ezt azzal indokolják, hogy így sosem fognak befuccsolni. Hogy lássuk mennyire rossz ez a stratégia elég ha megnézzük a 11. oldalon található optimális stratégiát, ahonnan azonnal kiderül, hogy 11-es lapösszeg fölött meglehetősen gyakran kell újabb lapot kérni).

eltérés a matematikai módszerrel kapott eredménytől.

Az, hogy a várható nyeremény negatív, nem jó jel. Ez azonban nem azt jelenti, hogy hosszútávon nem lehet nyerni, pusztán azt, hogy a játék kezdetekor a helyzet az osztónak kedvez. Bizakodásra adhat okot, hogy akár már a második leosztástól a játékos lehet előnyben. ha például az első játék során a játékos 7,8-ra 2-est kapott, a bank lapjai pedig, a kiosztás sorrendjében: 8,3,5,2, akkor a második leosztás várható nyeremény 0.001998, ami már pozitív érték.

Ebből látni lehet, hogy az egyes játékok során a várható nyeremény előjele gyakran változhat. Kérdés azonban, hogy milyen gyakran lesz pozitív a várható nyereményem, továbbá, hogy az ekkor nyert pénzem több lesz-e, mint az az összeg, amit negatív várható nyeremény esetén elvesztek. A kérdések megválaszolására a globális stratégiakereső/elemző módszereket a 6. fejezetben mutatom be.

5.3.3. Közelítés neurális hálóval

Ha a valószínűségszámítás felhasználásával kapott képletekben a rekurziót kifejtenénk, akkor explicit képletet adhatnánk az optimális stratégia várható értékére, és a $k_1, \ldots k_{10}$ értékek között. A rekurzió tényleges kifejtése és ezáltal az $E^{opt}_{k_1,\ldots k_{10}} = f(k_1, \ldots k_{10})$ függvény meghatározása azonban nem vezetne gyorsabb eredményre, hiszen e függvény bonyolultsága miatt tetszőleges $k_1, \ldots k_{10}$-hez tartozó helyettesítési érték meghatározása sokáig tartana.

Cél lenne tehát e függvény egy olyan egyszerűsítését vagy más jellegű modelljét elkészíteni, ami habár csak közelítést ad, sokkal gyorsabban adja a helyettesítési értékeket.

Ilyen jellegű feladatra (függvény interpoláció) kitűnően alkalmasak a neurális hálók. A feladat tehát egy olyan 10 bemenetelű, 1 kimenetelű neurális háló építése, ami jó közelítésül szolgál az $f(k_1, \ldots k_{10})$ függvénynek. A megfelelő neurális háló paramétereit akkor tudjuk minél pontosabban meghatározni, ha a közelítendő függvény változóinak fokszámáról minél több információval rendelkezünk. Ezek az információk a Black Jack esetében kiszámíthatóak. Az egyes lapokhoz tartozó változók maximális fokszáma megegyezik az egy elemi játék során előkerülő ilyen lap maximális számával. Például 6-osból a játékos és a bank is maximálisan 4-et kaphat, így k_6 legnagyobb fokszáma $2 \cdot 4 = 8$. Hasonlóan határozhatjuk meg a többi változó maximális fokszámát.

Egy neurális hálót tanítani kell. Ehhez kombinációkból és azokhoz tartozó várható nyereményekből álló tanítóhalmazra van szükség. Az egyes kombinációkhoz tartozó várható nyereményeket az előző két gyorsítási eszközt felhasználva (például 1% hibaarányú 5mp idejű) matematikai eszközökkel nyerhetjük.

Az lenne az ideális, ha a gyakran előforduló kombinációkra a neurális háló által adott közelítés pontos lenne, míg a nagyon ritkán előfordulókra ez nem feltétlenül szükséges. Tehát a tanítóhalmazban a gyakran előforduló kombinációk legyenek túlsúlyban. Ezt kétféleképpen érthetjük el.

Az első megoldás roppant egyszerű. Tegyünk úgy, mintha kaszinóban lennénk, játszunk. A játék folyamán előállított kombinációk legyenek mind a tanító, mind a teszthalmaz kombinációi. Így tényleg a gyakrabban előforduló kombinációk lesznek túlsúlyban a tanítóhalmazban.

A másik megoldás alapja az, hogy a gyakran előforduló, tehát amelyek előfordulásának valószínűsége nagy, meghatározhatók. Erről bővebben a 6.3.3 részben, a poli-hipergeometrikus eloszlás eseményterének bejárásánál lehet bővebben olvasni.

6. fejezet

Globális stratégiakeresés

6.1. Az optimális stratégia

Ahogy az elemi stratégiakeresésnél megadtuk az optimális stratégiát, és annak várható nyereményét, úgy globális szinten is ezt kell tenni. Globális szintű optimális stratégia keresésében az alábbi, meglehetősen egyszerű tétel nyújt támpontot.

tétel 6.1 (Optimális globális stratégia). *Tetszőleges játéksorozatnál az egy elemi játékra jutó várható nyeremény maximális ha a tétmanipulációra igaz az alábbi: az i-edik játék előtt maximális tétet teszünk, ha az i-edik játék várható nyeremény pozitív, és minimálisat, ha negatív.*

Bizonyítás. Könnyű belátni, hogy tetszőleges más stratégia esetén bármely játéksorozat egy egységre jutó várható nyereménye kisebb lesz, mint az optimálisé, hiszen

$$
\begin{aligned}
E^{\text{globális}} &= \frac{1}{n}\left(\text{tét}_1 \cdot E_1^{\text{elemi}} + \text{tét}_2 \cdot E_2^{\text{elemi}} + \ldots + \text{tét}_n \cdot E_n^{elemi}\right) \\
E_{opt}^{\text{globális}} &= \frac{1}{n}\left(\text{tét}^{\max} \sum_{i,\text{ ahol } E_i^{\text{elemi}}>0} E_i^{\text{elemi}} + \text{tét}^{\min} \sum_{i,\text{ ahol } E_i^{\text{elemi}}<0} E_i^{\text{elemi}}\right)
\end{aligned}
\tag{6.1}
$$

és bármely más stratégiára, ahol $E_i^{\text{elemi}} > 0$ esetében $\text{tét}_i \neq \text{tét}^{\max}$ ott

$$\text{tét}_i \cdot E_i^{\text{elemi}} < \text{tét}^{\max} \cdot E_i^{\text{elemi}}$$

továbbá, ahol ahol $E_i^{\text{elemi}} < 0$ esetében $\text{tét}_i \neq \text{tét}^{\min}$ ott

$$\text{tét}_i \cdot E_i^{\text{elemi}} < \text{tét}^{\min} \cdot E_i^{\text{elemi}}$$

miatt az összeg és annak n-ed része is kisebb lesz, ami azt jelenti, hogy a stratégia egy elemi játékára jutó várható nyeremény elmarad az optimálisétól. □

Feladat lenne tehát $E_{opt}^{\text{globális}}$ értéket meghatározni, hiszen ez alapján lehet választ adni olyan kérdésekre, hogy:
„Érdemes-e egyáltalán játszani?" vagy „Mennyit fogok így nyerni 2 óra alatt?"

Ha figyelmesen végigolvastuk a Megerősítő tanulás fejezetet, akkor feltűnhetett, hogy ott már volt szó egyfajta globális stratégia kereséséről. Mind a szimulációnál

mind a megerősítő tanulásnál, ha egy tanulólépés (illetve egy elemi játékszimuláció) után nem az alapállapotba állítjuk vissza a körülményeket (Black Jacknél ez a lapok újrakeverését jelenti), hanem folytatjuk a tanulást a megváltozott körülményekkel, akkor egy általános stratégiához jutunk. Ez a globális stratégia azt mutatta meg, ha a játéksorozatok alatt nem változtathatnánk stratégiát, akkor mi lenne az optimális. Az így meghatározott stratégia azonban mesze elmarad a tételben kimondott optimális stratégiától (olyannyira, hogy például a Black Jack esetében az előbbi negatív, míg az utóbbi pozitív várható nyereményt ad). Ez várható, hiszen az optimális globális stratégiánál pont azt tudjuk kihasználni, hogy vannak olyan kedvező helyzetek, amikor a várható nyeremény pozitív, és ilyenkor nagy téttel tudunk játszani.

Az alábbiakban két módszert mutatok be az optimális globális stratégiák egy elemi játékára jutó várható nyereményének meghatározására.

6.2. Szimulációs várhatóérték képzés

A szimulációkon alapuló módszert már ismerjük.

Lényege, hogy minden elemi játék előtt meghatározzuk az optimális stratégiát és annak várható nyereményét. Optimális stratégia alapján fogunk játszani és a nyereségünket (ami természetesen lehet negatív is) két változó egyikében fogjuk tárolni attól függően, hogy a várható nyeremény pozitív vagy negatív. Az optimális globális várható értékre igaz, hogy

$$E^{opt}_{\text{globális}} \leftarrow \frac{\text{tét}^{\max} \cdot \text{nyeremény}^{\text{pozitív}} + \text{tét}^{\min} \cdot \text{nyeremény}^{\text{negatív}}}{\text{játékszám}} \tag{6.2}$$

Ha arra vagyunk kíváncsiak, hogy milyen tétarány mellett érdemes leülni játszani, akkor azt a $-\frac{\text{nyeremény}^{\text{negatív}}}{\text{nyeremény}^{\text{pozitív}}}$ tört adja meg.

Mivel az elemi játék optimális stratégiáját, és annak várható nyereményét is sokszor kell meghatározni, ezért csak az elemi matematikai stratégiakeresés alkalmazható.

6.3. Várhatóérték képzés matematikai eszközökkel

Mint azt az elemi stratégiakereső eljárásoknál láttuk, a matematikai eszközöket alkalmazó módszer messze felülmúlta a többi módszert. Kérdés tehát, hogy lehet-e alkalmazni a valószínűségszámítást globális szinten is, gyorsabb és pontosabb eredmények elérése érdekében.

A következőkben bemutatom, hogyan lehetne kibővíteni az elemi stratégiakeresésnél adott képleteket, hogy teljesen pontosan meg tudjam mondani a globális szintű várható nyereményt. Ezt követően bemutatok egy gyakorlatban is alkalmazható közelítő algoritmust.

6.3.1. A pontos várható érték

A bevezetőben leírtam, hogy globális stratégiakeresésnek nincs értelme, ha az elemi játékok függetlenek egymástól. Tételezzük fel, hogy az elemi játékok függenek valamitől, minden elemi játéknak van egy helyzete. Például a Black Jack esetében a

helyzetet a $k_1, k_2, \ldots k_{10}$ lapkombináció (tehát k_1 darab ász, k_2 darab 2-es ... k_{10} darab 10-es) jellemzi egyértelműen.

Egy elemi játék várható nyereményét a helyzet által jellemzett start állapot várható nyereményével határoztuk meg, ami egyenlő volt az ezen állapotból elérhető állapotok várható nyereményének súlyozott összegével. Ezeket a szomszéd állapotokat már más helyzet jellemez (amit a kezdőhelyzetből, és az átmenetekből egyértelműen meg lehet határozni), és várható nyereményüket hasonlóan lehet kiszámítani, mint a start állapotét. A rekurziónak akkor van vége, ha végállapotba jutunk, aminek a várható nyereménye ismert. Ha ezen állapotokra a

$$E^s_{\text{helyzet}} = \text{ismert érték}$$

helyett a

$$E^s_{\text{helyzet}} = \text{ismert érték} + p_{\text{folytatás}}(\text{helyzet}) \cdot E^{\text{start állapot}}_{\text{helyzet}}$$

képletet adjuk, ahol $p_{\text{folytatás}}(\text{helyzet})$ adja meg azt, hogy mennyi a valószínűsége annak, hogy a *helyzet* által jellemzett helyzetben újabb elemi játék kezdődik.

A optimális globális stratégia várható nyereményét, ekkor a $E^{\text{start állapot}}_{\text{kezdeti helyzet}}$ adja meg.

A szemléletesség kedvéért bemutatom, hogyan kellene ezt alkalmazni a Black Jack esetében.

6.3.2. A Black Jack globális várható értékének kiszámítása

Black Jackben egy elemi játék helyzetét a kiosztandó $k_1, k_2, \ldots k_{10}$ lapkombináció írja le. A kiindulási helyzet ismert: $k_1 = 6 \cdot 4 = 24 = k_2 = \ldots = k_9$ és $k_{10} = 6 \cdot 16 = 96$, lévén 6 csomag kártyában 24 ász darab, 2-es ... 9-es. és 96 darab 10-es van.

Minden elemi játék kezdetekor ismerjük a lapkombinációt és tudjuk, hogy a teljes játéknak akkor van vége amikor kiosztják a lapok kb. 80%-át. Pontosabban az utolsó kiosztásra kerülő lapot a 4. és az 5. csomag között véletlenszerűen sorsolják. Közelítsük az utolsó kiosztásra kerülő lap eloszlásfüggvényét ($v =$)4.5pakli $= 4.5 \cdot 52 = 234$ várható értékű normális eloszlással. Az eloszlás szórását határozzuk meg úgy, hogy $P[4 \cdot 52, 5 \cdot 52] = 0.99$ legyen. Ha $\Phi(x, v, \sigma)$ -val jelöljük az így kapott eloszlás eloszlásfüggvényét (nem sűrűségfüggvényét), akkor a keresett $p_{\text{folytatás}}(\text{helyzet})$ -t kifejezhetjük az alábbi módon:

$$p_{\text{folytatás}}(\text{helyzet}) = p_{k_1,\ldots k_{10}}(\text{folytatás}) = 1 - \Phi(K, v, \sigma) \tag{6.3}$$

ahol, mint eddig is $K = \sum_{i=1}^{10} k_i$

Ez alapján átírhatjuk azokat a képleteket, amelyekben a végállapot várható értéke szerepel.

$$E^{\text{megáll}}_{k_1,\ldots k_{10}}(b, J) = \begin{cases} p(j_{\text{köv}}) \cdot E^{\text{megáll}}_{k_1,\ldots k_{10}}(b \uplus j_{\text{köv}}, J) & \text{ha } b < 17 \\ 1 & \text{ha } b < J \text{ vagy } b = \text{fuccs} \\ 0 & \text{ha } b = J \\ -1 & \text{ha } b > J \text{ és } b \neq \text{fuccs} \end{cases} \tag{6.4}$$

helyett

$$E^{\text{megáll}}_{k_1,\ldots k_{10}}(b,J) = \begin{cases} p(j_{\text{köv}}) \cdot E^{\text{megáll}}_{k_1,\ldots k_{10}}(b \uplus j_{\text{köv}}, J) & \text{ha } b < 17 \\ \text{tét} + (1-\Phi(K,v,\sigma))E^{opt}_{k_1,\ldots k_{10}} & \text{ha } b < J \text{ vagy } b = \text{fuccs} \\ 0 + (1-\Phi(K,v,\sigma))E^{opt}_{k_1,\ldots k_{10}} & \text{ha } b = J \\ -\text{tét} + (1-\Phi(K,v,\sigma))E^{opt}_{k_1,\ldots k_{10}} & \text{ha } b > J \text{ és } b \neq \text{fuccs} \end{cases} \tag{6.5}$$

ahol tét = tét$^{\max}$ vagy tét = tét$^{\min}$ attól függően, hogy $E^{opt}_{k_1,\ldots k_{10}} > 0$ vagy $E^{opt}_{k_1,\ldots k_{10}} < 0$. Hasonló módon kell módosítani az 5.11 az 5.13 az 5.14 és az 5.15 képleteket.

Az optimális globális stratégia várható nyereményét tehát az $E^{opt}_{24,24\ldots 96}$ kifejezésével kaphatjuk meg.

A Black Jack esetében ez az eljárás csak elvi szinten alkalmazható, hiszen a rekurzió olyan mély, hogy még szuperszámítógépeknek is évmilliókig kéne dolgozni, hogy az eredményt megkapjuk.

A következőkben ezért, egy nem csak Black Jackre alkalmazható, jól használható közelítő eljárást mutatok be.

6.3.3. Közelítés poli-hipergeometrikus eloszlással

Ez a közelítő módszer csak abban az esetben használható, amikor az elemi játékot leíró helyzetre igaz, hogy jellemezhető egy véges elemszámú halmazzal. A halmazban egy elem többször is előfordulhat. A játék kezdetekor mindig ugyanabból az ismert halmazból indulunk ki, és az elemei játékok során eltávolítunk mindig néhány elemet. Hogy két példát említsek, gondoljunk itt először különböző színű golyókkal teli urnára, amelyből minden játék során kiveszünk egy pár golyót, vagy gondolhatunk az olyan nyílt lapú kártyajátékokra, amelyeknél kezdetben néhány csomag kártyát megkevernek, és minden játék során valamennyi lapot kiosztanak.

A következőkben a halmaz elemeit lapoknak hívom, magát a halmazt pedig talonnak. Amennyiben l különböző lap van a talonban, akkor a halmazt egyértelműen leírja a $k_1, \ldots k_l$ lapkombináció, ahol k_1 jelenti az első típusú lap darabszámát, k_2 a második típusúét ... .

Láthatjuk, hogy ennél a modellnél, minden $k_1, \ldots k_l$ esetén kiszámíthatjuk a $k_1, \ldots k_l$ helyzet által leírt elemi játék várható nyereményét. Ha meg tudjuk mondani, hogy mennyi a valószínűsége a $k_1, \ldots k_l$ lapkombináció előfordulásának, akkor az optimális globális stratégia várható nyereménye közelíthető az egyes lapkombinációkhoz tartozó elemi játékok súlyozott összegével, ahol a súly a lapkombináció valószínűsége, azaz

$$E^{opt}_{\text{globális}} \leftarrow \sum_{k_1=n_1}^{0} \sum_{k_2=n_2}^{0} \ldots \sum_{k_l=n_l}^{0} p(k_1, k_2 \ldots k_l) \cdot E^{opt}_{k_1,k_2\ldots k_l} \tag{6.6}$$

ahol n_1 a kezdőtalonban lévő 1-es típusú lapok száma, n_2 a 2-es típusú lapok száma ... továbbá $p(k_1, k_2 \ldots k_l)$ adja meg, hogy a $K = \sum_{i=1}^{l} k_i$ véletlenszerűen kiválasztott lap közül mennyi a valószínűsége, hogy k_1 darab 1-es típusú, k_2 darab 2-es típusú ... lesz a lapok között.

Ezt a valószínűséget kis kombinatorikai alapismeretek mellett könnyű meghatározni. $N = \sum_{i=1}^{l} n_i$ lap közük K darab lapot $\binom{N}{K}$ féleképpen lehet kiválasztani, amiből a kedvező esetek száma $\binom{n_1}{k_1}\binom{n_1}{k_1}\dots\binom{n_l}{k_l}$. Ezek szerint

$$p(k_1, k_2 \dots k_l) = \frac{\binom{n_1}{k_1}\binom{n_1}{k_1}\dots\binom{n_l}{k_l}}{\binom{N}{K}} \tag{6.7}$$

Ezt a valószínűségű eloszlást hívja a szakirodalom poli-hipergeometrikus eloszlásnak [3].

Amennyiben a játék olyan, hogy sosem osztják ki az összes lapot, és $p_{\text{folyt}}(K)$ adja meg annak valószínűségét, hogy $N-K$ elemszámú talonnál megkezdődik újabb elemi játék, akkor a helyes képlet:

$$p(k_1, k_2 \dots k_l) = \frac{\binom{n_1}{k_1}\binom{n_1}{k_1}\dots\binom{n_l}{k_l}}{\binom{N}{K}} \cdot p_{\text{folyt}}(K) \tag{6.8}$$

A feladat az lenne, hogy az összes $k_1, \dots k_l$ lapkombinációra, amelyre fennáll, hogy $p_{\text{folyt}}(K) > 0$, meghatározni $p(k_1, k_2 \dots k_l)$-t és a hozzá tartozó elemi játék várható nyereményét.

Mielőtt nekiállnánk megprogramozni a feladatot, nem árt ha egy kis információt nyerünk arról, hogy milyen számítás- és időigénye van ennek a közelítésnek. Tudjuk, hogy az $\binom{n}{k}$ típusú kifejezések kiszámítása lassú művelet, és mivel a $p(k_1, k_2 \dots k_l)$ kifejezésben l+1 darab ilyen van, ezért túl nagyszámú lapkombináció esetre kiszámíthatatlan lenne a végeredmény.

Kérdés tehát rögzített K mellett a poli-hipergeometrikus eloszlás állapotterének elemszáma[1]. A következő tételben erre adok képletet, attól függően, hogy K milyen intervallumba esik. Mivel a kártyajátékokra jellemző, hogy kezdetben minden lapból ugyanannyi lap van a talonban, ezért azt a speciális esetet vizsgálom, amikor $n_1 = n_2 = \dots n_l = n$.

tétel 6.2. *Ha az n,l,K paraméterű (tehát* $p(k_1, k_2 \dots k_l) = \frac{\binom{n}{k_1}\binom{n}{k_1}\dots\binom{n}{k_l}}{\binom{n \cdot l}{K}}$*) poli-hipergeometrikus eloszlás állapotterének elemszámát* $S(n, l, K)$*-val, jelölöm, akkor igaz, hogy*

$$S(n,l,K) = \begin{cases} \binom{l+K-1}{K} & ha\ K \le n \\ \binom{l+K-1}{K} - l \cdot \binom{l+K-(n+1)-1}{K-(n+1)} & ha\ n < K \le 2n \\ S(n,l,K-1) + S(n,l-1,K) - \\ S(n,l-1,K-(n+1)) & ha\ n < K \end{cases} \tag{6.9}$$

továbbá

$$S(n,1,K) = \begin{cases} 1 & ha\ K \le n \\ 0 & ha\ n < K \end{cases} \tag{6.10}$$

[1] A poli-hipergeometrikus eloszlás állapotterének elemszámára megadott rekurzív képletek generátorfüggvénye magasszintű kombinatorikai könyvekben megtalálható [2]

Bizonyítás. $S(n,l,K)$ értekét úgy is kiszámítható, ha megmondjuk hányféleképpen lehet l dobozba K golyót elhelyezni úgy, hogy semelyik dobozban ne legyen n-nél több golyó. Ez azért igaz, mert minden egyes golyóelrendezéshez kölcsönösen egyértelműen megfeleltethetünk egy lapkombinációt úgy, hogy az 1-es dobozban lévő golyók száma egyezzék meg a lapkombinációban található 1-es típusú lapok számával, a 2-es dobozban található golyók száma, a 2-es típusúéval ...

Azt, hogy $S(n,1,K) = \begin{cases} 1 & \text{ha } K \leq n \\ 0 & \text{ha } n < K \end{cases}$ könnyű belátni, lévén 1 dobozba K golyót egyféleképpen tehetek, kivéve ha ez megsérti a kritériumot, miszerint n-nél több golyót nem kerülhet egy dobozba sem.

Vizsgáljuk meg a tétel első felét, amikor $K \leq n$. Ilyenkor a kritériumot nem kell figyelembe venni, hiszen ha összesen max. n golyót helyezek el a dobozokban akkor nyilván egy dobozba sem kerülhet n-nél több golyó. A kérdés ilyenkor, hogy hányféleképpen helyezhetek el l darab dobozba K darab golyót.

Ennek megválaszolásához fogjuk fel az l darab dobozt, mintha csak 1 darab nagy doboz lenne, de annak belseje $l-1$ fallal el van választva, ami végül is l részre osztja. Ha a nagy dobozban $K+l-1$ pozíció van golyók és falak számára, akkor az összes fal elhelyezése egyértelműen meghatározza a golyók elrendezését. Tudjuk, hogy $l-1$ falat pedig $\binom{l+K-1}{l-1} = \binom{l+K-1}{K}$ féleképpen tudok letenni. Ezzel az állítás első felét beláttuk.

Most vizsgáljuk az $n < K < 2n$ esetet. A fenti képlet ekkor nem alkalmazható, mert megszámolnánk azokat az eseteket is amelyek megsértik a kritériumot. A kritériumnak megfelelő esetek számát megkaphatjuk ha az összes esetből levonjuk a rossz esetek számát. Az összes eset számát az imént adtuk meg, az pedig hogy mennyi szegi meg a kritériumot könnyű kiszámítani. Mivel $K < 2n$, ezért pontosan 1 dobozban lehet n golyónál több. Tehát egy dobozba tegyünk *n+1* golyót, a maradék $K-(n+1)$ golyót pedig helyezzük el tetszőlegesen. A kritériumszegő dobozt l doboz közül választhatjuk ki, az $K-(n+1)$ golyót pedig $\binom{l+K-(n+1)-1}{K-(n+1)}$ féleképpen tehetjük l dobozba, tehát

$$\begin{aligned} \#\text{jó esetek} &= \#\text{összes eset} - \#\text{rossz esetek} \\ S(n,l,K) &= \binom{l+K-1}{K} - l \cdot \binom{l+K-(n+1)-1}{K-(n+1)} \end{aligned} \tag{6.11}$$

A tétel harmadik része tetszőleges $n < K$-ra egy rekurzív képletet ad, hogy a $2n < K$ estekre is meg tudjuk határozni $S(n,l,K)$ értékét. Az egyenlőség igazolásához az alábbi egyenlőséget fogom felhasználni:

$$S(n,l,K) = \sum_{i=0}^{\min(K,n)} S(n,l-1,K-i) \tag{6.12}$$

Ez az egyenlőség azért igaz, mert az első dobozba 0, vagy 1, ... vagy $\min(K,n)$ golyót tehetek, míg a többi $l-1$ dobozba $S(n,l-1,K-i)$ féleképpen tehetem a maradék golyót. Ha $K > n$, akkor a képletben $\min(K,n)$ n-re módosul. Ekkor:

$$S(n,l,K) = S(n,l-1,K) + S(n,l-1,K-1) + \ldots + S(n,l-1,K-n) \tag{6.13}$$

mivel

$$\begin{aligned} S(n,l,K-1) =& S(n,l-1,K-1)+S(n,l-1,K-2)+\ldots+ \\ & S(n,l-1,K-n)+S(n,l-1,K-(n+1)) \end{aligned} \tag{6.14}$$

ami behelyettesítésével a 6.13 egyenletbe következik:

$$S(n,l,K) = S(n,l,K-1)+S(n,l-1,K)-S(n,l-1,K-(n+1)) \tag{6.15}$$

□

Látható, hogy a [1,2n] intervallumban K növelésével az elemszám exponenciálisan nő. 2n fölött az elemszám tovább nő és $K = \frac{n \cdot l}{2}$-nél éri el a maximumot. Ezekből következik, hogy az összes $k_1, \ldots k_l$ kombináció és ezek által leírt helyzetek elemi játékainak várható értéke, nagy $N = n \cdot l$-re, nem határozható meg.

A következő részben egy olyan algoritmust mutatok be, ami nem határozza meg az összes $k_1, \ldots k_l$ kombinációt, viszont a globális várható értéknek mégis egy jó közelítését adja. Ez az algoritmus már gyakorlatban teljesíthető számításigénnyel rendelkezik.

6.3.4. Ideális állapotér „zömét" bejáró algoritmus

Az eljárás lényege, hogy ne határozzuk meg adott K-ra, az összes $k_1, \ldots k_l$ kombinációt, hanem csak annyit, amennyi lefedi az állapottér nagy (pl.: 95) %-át. Lévén, hogy a hipergeometrikus eloszlás (poli-hipergeometrikus eloszlás 2 dimenziós esete) közelíthető bizonyos paraméterű normális eloszlással, így sejthetjük, hogy a nagy valószínűségek az állapotterének kis helyére koncentrálódnak [1]. Cél megtalálni a legnagyobb valószínűségű kombinációkat.

Definiáljuk az ideális távolság fogalmát a kombinációk terében, az alábbiak szerint:

definíció 6.3 (Ideális távolság). Tetszőleges $\tilde{k} = (k_1, k_2 \ldots k_l)$ és $\tilde{l} = (l_1, l_2 \ldots l_l)$ kombinációk ideális távolsága előfordulásaik valószínűségének különbsége, azaz

$$d^{\text{ideális}}(\tilde{k}, \tilde{l}) = |p(k_1, \ldots k_l) - p(l_1, \ldots l_l)| \tag{6.16}$$

Ez alapján az ideális algoritmus az lenne, ha kiindulva a legnagyobb valószínűségű kombinációból vennénk a hozzá közeli, majd egyre távolabbi kombinációkat és határoznánk meg azok előfordulási valószínűségét és hozzá tartozó elemi játékok várható nyereményét addig, míg a megvizsgált kombinációk összvalószínűsége bizonyos korlát (pl.:0.95) fölé nem emelkedik.

Kérdés még, hogy honnan induljak, azaz hogy melyik kombináció előfordulásának legnagyobb a valószínűsége.

tétel 6.4. *Adott K-ra annak a $k_1, \ldots k_l$ kombináció előfordulásának maximális a valószínűsége, amelynél $\forall i$-re igaz, hogy $\lfloor \frac{K}{l} \rfloor \leq k_i \leq \lceil \frac{K}{l} \rceil$. Tehát az egyenletes kombináció a maximális valószínűségű kombináció.*

Bizonyítás. Bebizonyítom, hogy ha egy tetszőleges kombinációt „torzítok", azaz egy olyan eleméhez, amely $\lfloor \frac{K}{l} \rfloor$-nél nagyobb 1-et hozzáadok, és egy olyan elemhez, amely $\lceil \frac{K}{l} \rceil$-nél kisebb 1-et levonok, akkor a valószínűsége csökken:

$$\begin{aligned} p(k_1 \dots k_i \dots k_j \dots k_l) > & p(k_1 \dots k_i + 1 \dots k_j - 1 \dots k_l) \\ \frac{\binom{n}{k_1} \dots \binom{n}{k_i} \dots \binom{n}{k_j} \dots \binom{n}{k_l}}{\binom{N}{K}} > & \frac{\binom{n}{k_1} \dots \binom{n}{k_i+1} \dots \binom{n}{k_j-1} \dots \binom{n}{k_l}}{\binom{N}{K}} = \\ & \frac{\binom{n}{k_1} \dots \binom{n}{k_i} \cdot \frac{n-k_i}{k_i+1} \dots \binom{n}{k_j} \cdot \frac{k_j}{n-k_j+1} \dots \binom{n}{k_l}}{\binom{N}{K}} \\ 1 > & \frac{n-k_i}{k_i+1} \cdot \frac{k_j}{n-k_j+1} \end{aligned} \tag{6.17}$$

a $k_i \geq \lceil \frac{K}{l} \rceil$, miatt igaz, hogy $\frac{n-k_i}{k_i+1} \leq \frac{n-\lceil \frac{K}{l} \rceil}{\frac{K}{l}+1}$ és hasonlóan a $k_j \leq \lfloor \frac{K}{l} \rfloor$ miatt $\frac{k_j}{n-k_j+1} \leq \frac{\lfloor \frac{K}{l} \rfloor}{n-\lfloor \frac{K}{l} \rfloor+1}$, ezekből pedig következik maga az állítás, hiszen:

$$\begin{aligned} \frac{n-k_i}{k_i+1} \cdot \frac{k_j}{n-k_j+1} \leq & \frac{n-\lceil \frac{K}{l} \rceil}{\lceil \frac{K}{l} \rceil + 1} \cdot \frac{\lfloor \frac{K}{l} \rfloor}{n-\lfloor \frac{K}{l} \rfloor + 1} = \\ & \frac{n-\lceil \frac{K}{l} \rceil}{n-\lfloor \frac{K}{l} \rfloor + 1} \cdot \frac{\lfloor \frac{K}{l} \rfloor}{\lceil \frac{K}{l} \rceil + 1} < 1 \end{aligned} \tag{6.18}$$

lévén $\lfloor \frac{K}{l} \rfloor + 1 = \lceil \frac{K}{l} \rceil$

Ezzel bebizonyítottam, hogy az olyan kombinációnak lesz maximális a valószínűsége, amelyekre nincs olyan i,j, hogy $k_i > \lceil \frac{K}{l} \rceil$ és $k_j < \lfloor \frac{K}{l} \rfloor$. □

Az ideális algoritmus megvalósításához tudnunk kéne, hogy adott kombinációhoz ki van legközelebb, tehát az eddig nem vizsgált kombinációk közül kinek maximális a valószínűsége, hiszen ezt kéne következő lépésben megvizsgálni. Sajnos nem ismerek olyan eljárást, amely bármely más valószínűség kiértékelése nélkül, megadná a következő kombinációt.

Éppen ezért most egy olyan algoritmust mutatok be, ami habár nem ideális, a valószínűségek bejárására igaz lesz egy tétel, ami bizonyíték az algoritmus létjogosultságára.

6.3.5. Megvalósítható „zöm"bejáró algoritmus

Ennek az algoritmusnak a lényege ugyanaz, mint az ideálisnak. Induljunk ki a maximális valószínűségű kombinációból, és próbáljuk megkeresni a többi nagy valószínűségű kombinációt. Már tudjuk, hogy a maximális valószínűségű kombináció az egyenletes kombináció. Adott K-ra ha $l \nmid K$ akkor több egyenletes kombináció is létezik (pl.:K=9 és n=4-ra a (2,3,2,2) és a (2,2,2,3) is egyenletes kombináció). Az algoritmus során az egyenletes kombinációt véletlenszerűen választjuk, de úgy hogy az eggyel kisebb K-hoz tartozó egyenletes kombinációtól csak egy értékben térjen el (hiszen ekkor az új egyenletes kombináció valószínűsége, a régiből, egyetlen szorzással és osztással számítható).

Ahogy az ideálisnál, úgy ennél az algoritmusnál is definiálok egy távolság fogalmat. A bejárásra igaz lesz, hogy egy kombináció vizsgálata után mindig a hozzá legközelebbi kombinációnál folytatódik a bejárás.

Az algoritmusnak két szépsége van:

- Az egyes kombinációk valószínűségei az előző kombináció valószínűségének ismeretében nagyon gyors (egy szorzás és egy osztás) művelettel előállíthatók.
- A következő kombináció meghatározásához nincs szükség más valószínűségek kiértékelésére.

definíció 6.5. Tetszőleges $\tilde{k} = (k_1, k_2 \dots k_l)$ és $\tilde{l} = (l_1, l_2 \dots l_l)$ kombinációk távolságát így értelmezzük:

$$\begin{aligned} d(\tilde{k}, \tilde{l}) &= \frac{1}{2} \left(|k_1 - l_1| + |k_2 - l_2| + \dots |k_l - l_l| \right) \\ &= \frac{1}{2} \left(\sum_{i=1}^{l} |k_i - l_i| \right) \end{aligned} \tag{6.19}$$

Az algoritmus lényege, hogy vizsgáljuk meg először az egyenletes kombinációt, majd az egyenletes kombinációtól 1 távolságra lévőket, majd a 2 távolságra lévőket, egészen addig, amíg a megvizsgált kombinációk előfordulásának összvalószínűsége bizonyos korlát fölé nem emelkedik. Habár erre az algoritmusra nem igaz, hogy az i-edik lépésben vizsgált kombináció előfordulásának valószínűsége nagyobb, mint bármely később vizsgált kombinációé, viszont igaz lesz egy tétel, ami majd összefüggést ad e távolság és a valószínűségek között. Ezt a tételt csak e rész végén mondom ki, és bizonyítom be. Ezt azért teszem, mert az algoritmus részleteinek ismeretéből könnyebb a bizonyítást előállítani.

Kombinációk generálása.

Ahogy az ideális esetnél, úgy itt is felmerül az a kérdés, hogy miként generálom a következő vizsgálandó kombinációt. Hogy ezt megértsük, vezessünk be egy újabb fogalmat a kombináció módosítójának fogalmát

definíció 6.6 (kombináció módosítója). Tetszőleges $\tilde{k} = (k_1, k_2 \dots k_l)$ kombináció módosítója az a kombináció aminek elemeit a következőképpen származtatom:

$$M(\tilde{k}) = (k_1 - e_1, k_2 - e_2, \dots k_i - e_i) \tag{6.20}$$

ahol $(e_1, e_2, \dots e_l)$ egy rögzített egyenletes kombinációt jelöl.

Tehát egy kombináció módosítóján a kombináció és az egyenletes kombináció különbségét értjük. Látható, hogy tetszőleges kombináció távolsága az egyenletestől megegyezik módosítója elemei abszolútértékei felének összegével. Két módosító távolságán ezentúl a hozzájuk tartozó kombinációk távolságát értem.

Az egyenletes kombináció és a tetszőleges kombináció módosítója egyértelműen definiálja magát a kombinációt. Éppen ezért az algoritmus során nem a kombinációkat, hanem azok módosítóit fogjuk előállítani. Egy lépésben az összes T távolságban

lévő módosítóból (a továbbiakban alapmódosítók) állítjuk elő a *T+1* távolságra lévő módosítókat (a továbbiakban generált módosítók). Ezt az egységmódosítók segítségével teszük. Az egységmódosítók az itt felsorolt módosítók

$$
\begin{array}{c}
(+1,\text{-}1,\ 0,\ 0,\ \ldots\ 0,\ 0) \\
(\text{-}1,+1,\ 0,\ 0,\ \ldots\ 0,\ 0) \\
(+1,\ 0,\text{-}1,\ 0,\ \ldots\ 0,\ 0) \\
\ddots \\
(\ 0,\ 0,\ 0\ ,\ 0,\ \ldots\ +1,\text{-}1) \\
(\ 0,\ 0,\ 0\ ,\ 0,\ \ldots\ \text{-}1,+1)
\end{array}
$$

tehát az i,j paraméterű egységmódosítók azok a módosítók amelyek i-edik pozíciójában $+1$-es j-edikben pedig -1-es szerepel.

Az alapmódosítókból (jelöljük $(a_1, a_2 \ldots a_l)$-el) úgy állítjuk elő a generált módosítókat, hogy minden alapmódosítóhoz azokat az i,j paraméterű egységmódosítókat adjuk, hozzá amelyekre igazak az alábbi feltételek:

1. i (illetve j) ne legyen nagyobb, mint az alapmódosító első pozitiv (illetve negatív) elemének pozíciója.

2. $a_i \geq 0$ és $a_j \leq 0$.

3. $a_j + e_j > 0$, ahol e_j-vel az egyenletes kombináció j-edik elemét jelölöm.

Hogy könnyebben megértsük 3 példát mutatok (e=(3,3,4) egyenletes kombinációt feltételezve):

művelet	helyes-e	eredmény/indok
$(2,1-3)+(1,0,-1)$	$\surd$	(3,1,-4)
$(2,1,-3)+(1,-1,0)$	$\ominus$	mivel $a_2 > 0$.
$(2,1,-3)+(0,+1,-1)$	$\ominus$	mivel az első pozitív szám pozíciója nagyobb, mint a $+1$ pozíciója.

Az első feltétellel biztosíthatjuk, hogy ne állítsuk többször elő ugyanazt a kombinációt, a másodikkal azt, hogy ne állítsunk elő kisebb távolságú módosítót, a harmadik feltétel célja pedig, hogy elkerüljük a negatív elemű kombináció módosítójának generálását.

A következő két tétel további információt ad az algoritmus jóságáról. Az elsőt nem bizonyítom, de ezt a teljes indukció felhasználásával bárki könnyedén megteheti.

tétel 6.7. *Tetszőleges K esetén a fenti algoritmus a $(0,0,\ldots 0)$ kombinációból indulva az összes kombináció módosítóját előállítja, úgy hogy egyetlen módosítót sem generál többször.*

tétel 6.8. *Ha a poli-hipergeometrikus eloszlás elemeire úgy definiálom a távolságot, hogy $d(\tilde{k},\tilde{l}) = \frac{1}{2}\left(\sum_{i=1}^{l} |k_i - l_i|\right)$, akkor igaz, hogy az egyenletes kombinációtól T+1 távolságban lévő kombinációk halmazát fel lehet osztani részhalmazokra úgy, hogy minden részhalmazhoz kölcsönösen egyértelműen megfeleltethetek egy elemet a*

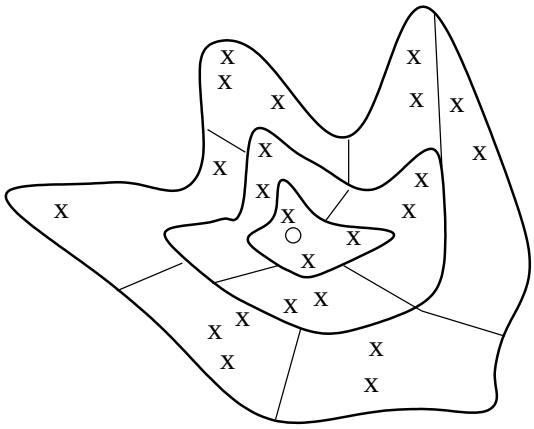
T távolságban lévő kombinációk halmazából, úgy hogy egyrészről ezek az elemek is lefedjék a teljes halmazt, másrészről a részhalmaz minden elemének valószínűsége nem kisebb, mint a megfeleltetett elem valószínűsége.

Bizonyítás. Minden T távolságban lévő kombinációhoz azokat a T+1 távolságban lévő kombinációkat fogom hozzárendelni, amelyek módosítóját a fent vázolt algoritmus segítségével e T távolságban lévő módosítóból állítottam elő. Az előző tétel értelmében, ha a T távolságban lévő elemek halmaza teljes, akkor a T+1 távolságban lévő is az lesz (tétel első fele). Nézzük most azt a generált módosítót, amit egy alapmódosítóból olyan egység-módosító segítségével állítottunk elő, amely i-edik pozíciójában szerepel a +1-es j-edikben pedig a -1-es. A második feltétel kimondja, hogy ekkor a módosító i-edik pozíciójában nemnegatív, j-edik pozícióban pedig nempozitív számnak kell állnia. Tehát a kombináció i-edik pozíciójában az érték nagyobb vagy egyenlő az egyenletes kombináció ezen pozíciójában álló értéknél, a j-edikben pedig kisebb vagy egyenlő.

A 6.4 tétel kimondja, hogy ha egy kombináció i-edik pozíciójára igaz, hogy $k_i \geq \lceil \frac{K}{l} \rceil$, továbbá a j-edikre pedig, hogy $k_j \leq \lfloor \frac{K}{l} \rfloor$, akkor ezen értékekhez $+1$ illetve -1-et hozzáadva az előállított kombináció valószínűségük kisebb lesz. Tehát ezen esetekre az állítást beláttuk.

Egy eset marad csak, amikor is $k_i = \lfloor \frac{K}{l} \rfloor$ és $k_j = \lceil \frac{K}{l} \rceil$. Ekkor azonban $k_i + 1 = \lceil \frac{K}{l} \rceil = k_j$ és $k_j - 1 = \lfloor \frac{K}{l} \rfloor = k_i$ tehát a módosítás során a valószínűség nem változik. □

Hogy szemléltessük a tételt vegyük a 2 dimenziós esetet. Feleltessünk meg minden kombinációnak egy pontot a koordinátarendszerben, és 2 pont távolsága, valószínűségeik különbségével legyen arányos. Ekkor az általam definiált távolság szerinti egyre távolabb lévő kombinációk, mint körgyűrűk veszik körül a kisebb távolságban lévő kombinációkat. Ezen gyűrűk alakja azonban nem szabályos kör (mint ahogy az lenne ideális esetben), hanem amőba alakúak. Az algoritmusunk során tehát elindu-



6.1. ábra. A távolságok amőbaszerű sávjai

lunk a középpontból, és mindig egy amőbaszerű gyűrűt vizsgálunk meg. Habár ezen gyűrűknek lehetnek nagy kinyúlásai (tehát olyan kombinációk aminek a többiekhez

képest sokkal kisebb a valószínűsége) azonban az eggyel távolabbi gyűrű még ezt is körülveszi.

A kombinációk generálása által igazolható, a rész elején kijelentett állítás, miszerint az egyes kombináció előfordulásának valószínűségéhez nem kell egyetlen $\binom{n}{k}$ típusú kifejezést sem meghatározni. Az, i,j paraméterű egység-módosító által generált új kombináció valószínűsége, az alapkombináció valószínűségének $\frac{n-k_i}{k_i+1} \cdot \frac{k_j}{n-k_j+1}$-vel való szorzásával számítható.

A fenti algoritmus gyors, viszont nagy a memóriaigénye. Ennek oka az, hogy a $T+1$ távolságban található kombinációk módosítóinak meghatározásához ismernünk kell a T távolságban lévőket, tehát azokat tárolnunk kell. Mivel már közepes K-ra is olyan távolságban kell vizsgálnunk a kombinációkat, hogy ahhoz nagyon sok módosító tartozik, ezért felmerül az igény az algoritmus olyan változtatására ami után nem kell ennyi módosítót tárolni. A következő részben egy ilyen megoldást mutatok be.

6.3.6. Módosított memóriabarát algoritmus

Ha a fenti algoritmusnak a megvizsgált kombinációk összvalószínűségére 95%-ot írunk elő, akkor a módosítók által lefoglalt terület már K=14-ra 40 Mbyte körül lesz. A következőkben egy olyan módosítást mutatok be, aminek következtében a memóriaigény több nagyságrenddel kisebb lesz, a fenti esetre például 308 byte, ezenfelül a sebesség is tovább nő.

Az elgondolás lényege, hogy ne tároljuk az olyan módosítókat többször, amelyeknek az elemei egymás permutációi. Ha csak az 1 távolságban lévő módosítókat nézzük, akkor abból $l(l-1)$ darab olyan van, amire igaz, hogy 1 darab +1-es és egy darab -1-es szerepel. Tároljunk csak 1 ilyen módosítót és vegyük ennek az összes permutációját.

Ezt az egy kiemelt módosítót a továbbiakban bázismódosítónak nevezem és $(b_1, b_2 \ldots b_l)$-el jelölöm. Minden bázismódosítóra legyen igaz, hogy $b_1 \leq b_2 \leq \ldots \leq b_l$, tehát elemei nemcsökkenően kövessék egymást. Például az 1 távolságban lévő bázismódosítók a

$$(-1, 0 \ldots 0, 1)$$

a 2 távolságban lévők pedig

$$(-2, 0 \ldots 0, 2)$$
$$(-2, 0 \ldots 1, 1)$$
$$(-1, -1 \ldots 0, 2)$$
$$(-1, -1 \ldots 1, 1)$$

Feladat ezután a bázismódosítok előállítása, továbbá egy hatékony permutáló algoritmus megvalósítása.

Bázismódosítok előállítása

Az előző fejezetben bemutatott módosítógeneráló algoritmus bázismósosítóknál nem működik, mivel:

- Nem generálja az összes bázismódosítót.
- Olyan módosítót is előállít, ami nem bázismódosító.

Ezek igazolásához gondoljunk arra, hogy a 2. pozícióban sosem lehet negatív szám az első feltétel miatt, és a $(-2, 0 \ldots 1, 0, 1)$ kombinációt az algoritmus elő fogja állítani annak ellenére, hogy ez nem bázismódosító

A bázismódosítókat generáló új algoritmus az alábbi:

Vegyük a T távolságú bázismódosítókat (alapbázis) és csak olyan i, j paraméterű egységmódosítóhoz adjuk hozzá, amire igaz, hogy:

- $b_i \neq b_{i+1}$ és $b_j \neq b_{j-1}$
- $b_i \geq 0$ és $b_j \leq 0$
- $-b_j < 1 + \frac{K+1-j}{l}$
- Ellenőrizzük, hogy a generált bázist állítottuk-e elő korábban.

Az első feltétel biztosítja, hogy csak bázisokat állítsunk elő. A második feltételt már ismerjük, ez szavatolja, hogy a generált bázismódosító távolsága nagyobb legyen az alapmódosítók távolságánál. A harmadik feltétellel az olyan módosítókat zárjuk ki, amely tartalmazna olyan elemet, amit az egyenletes kombináció bármely eleméhez hozzáadva negatív számot kapnánk (gondoljunk például arra ha $K < l$, akkor az egyenletes kombináció csak 0 és 1-esekből áll, tehát nem szabad -2 elemet tartalmazó módosítót előállítani).

A negyedik feltétel azt sugallja, hogy, ellentétben az előzővel, erre az algoritmusra már nem igaz, hogy nem állítja elő többször ugyanazt a kombinációt. Ha az előző algoritmusban ez nem teljesült volna, akkor a dupla kombinációk kiszűrésére készített módosítás komolyan lassította volna az algoritmust. Látni fogjuk, hogy itt a bázismódosítók kis száma miatt ez az ellenőrzés még kis mértékben sem lassítja az algoritmust implementáló programot.

Teljes indukcióval ismét könnyen belátható, hogy az algoritmus teljes, azaz minden bázismódosítót előállít.

A fenti táblázatban a bázismódosítok előállítását illusztrálom. Azok a bázismódisítók, amelyek mellett a $\ominus$ jel látható, előállításukkor megszegték a 4. szabályt, azaz már egyszer generáltuk őt valamely más bázismódosítóból.

6.1. táblázat. Példa bázismódosítók generálására

T=0	$(0,0,0,0,0,0)$			
T=1	$(-1,0,0,0,0,1)$			
T=2	$(-1,-1,0,0,1,1)$	$(-1,-1,0,0,0,2)$	$(-2,0,0,0,1,1)$	$(-2,0,0,0,0,2)$
T=3	$(-1,-1,-1,1,1,1)$	$(-1,-1,-1,0,1,2)\ominus$	$(-2,-1,0,1,1,1)\ominus$	$(-2,-1,0,0,1,2)\ominus$
	$(-1,-1,-1,0,1,2)$	$(-1,-1,-1,0,0,3)$	$(-2,-1,0,0,1,2)\ominus$	$(-2,-1,0,0,0,3)\ominus$
	$(-2,-1,0,1,1,1)$	$(-2,-1,0,0,1,2)\ominus$	$(-3,0,0,1,1,1)$	$(-3,0,0,0,1,2)\ominus$
	$(-2,-1,0,0,1,2)$	$(-2,-1,0,0,0,3)$	$(-3,0,0,0,1,2)$	$(-3,0,0,0,0,3)$

Bázismódosítók permutálása

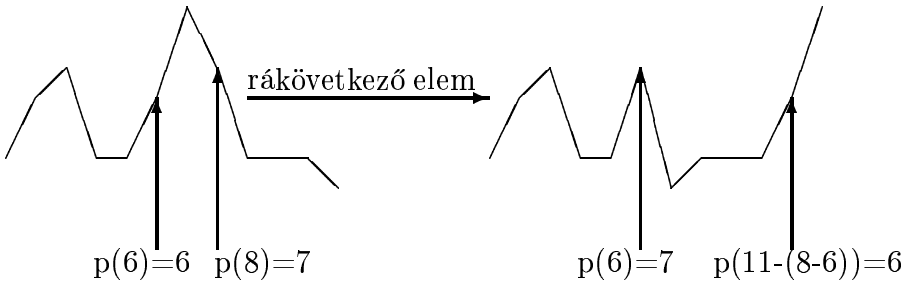
Egy kombináció permutációinak előállításához egy, már ismert permutáló algoritmus módosított változatát fogom felhasználni. Az algoritmus neve: lexikografikus bejárás[6][2].

A lexikografikus bejárás lényege, hogy egy halmaz elemeit akarom bejárni, úgy hogy semelyik elemet ne érintsem kétszer. A halmazra igaz, hogy egy elemet többször is tartalmaz, továbbá a halmaz elemeire lehet definiálni egy tranzitív kisebb relációt.

A kisebb relációból származtathatunk egy új fogalmat, egy elem rákövetkező elemének fogalmát. Akkor mondjuk, hogy b elem a-nak rákövetkező eleme, ha $a < b$, de $\nexists c$ hogy $a < c$ és $c < b$ egy időben teljesüljenek

Az összes permutáció megkeresésénél a feladat tehát a kisebb reláció definiálása, továbbá egy elem rákövetkező elemének meghatározása. Azt mondjuk, hogy $p = (p_1, p_2, \ldots p_l)$ elem akkor kisebb $q = (q_1, q_2, \ldots q_l)$ ha $\exists i$ hogy $p_i < q_i$, és minden $1 \leq j < i$-re $p_j = q_j$. Például $(-3, -2, 0, 5) < (-3, -2, 5, 0)$ vagy $(-1, 1, -1, 1) < (1, -1, -1, 1)$ Ez alapján egy permutációhalmazban az lesz a legkisebb elem, amelyikre $p_1 \leq p_2 \leq \ldots \leq p_l$, a legnagyobb pedig amelyikre $p_1 \geq p_2 \geq \ldots \geq p_l$.

Ahhoz, hogy be tudjuk járni a permutációhalmazt, meg kellene határozni egy permutáció rákövetkező permutációját (nyilván a legnagyobbnak nincs rákövetkező eleme). A következő ábra illusztrálja az utána leírt algoritmust.



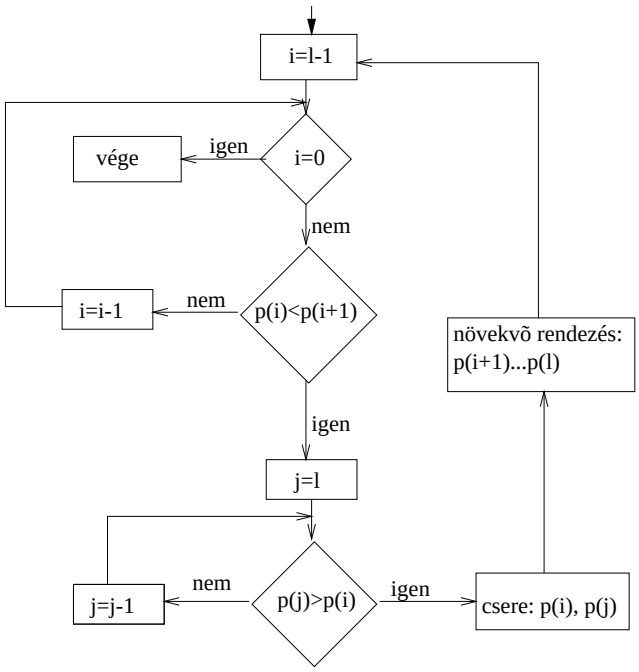
6.2. ábra. Példa a rákövetkező elem meghatározására

Nézzünk egy tetszőleges $p_1, \ldots p_l$ kombinációt. Induljunk el az utolsó elemétől egészen addig amíg igaz, hogy az elem nem nagyobb az előtte álló elemnél. Ha a vizsgált permutáció nem a legnagyobb permutáció, akkor lesz olyan p_i amire már igaz, hogy $p_i < p_{i+1}$. Keressük meg ekkor a $\{p_{i+1}, \ldots p_l\}$ elemek közül azt a legkisebbet (p_j-t), ami nagyobb p_i-nél. Cseréljük meg p_i-t p_j-vel, majd rendezzük a $\{p_{i+1}, \ldots p_l\}$ halmazt nagyság szerint növekvő sorrendbe. Mivel könnyű utánagondolni, hogy a $\{p_{i+1}, \ldots p_l\}$ halmaz nagyság szerint csökkenően volt rendezve, ezért az átrendezéshez csak $\lfloor \frac{l-i}{2} \rfloor$ csere szükséges. Az így kapott permutáció az eredeti rákövetkező eleme. A következő oldalon az algoritmus blokkdiagramja látható.

Amennyiben a legkisebb permutációból indulunk el, akkor az összes különböző permutációt érinteni fogjuk.

Hogy könnyebben megértsük a $(4, 6, 7, 4, 4, 6, 9, 7, 4, 4, 3)$ kombinációnak fogom meghatározni a rákövetkező elemét. Mivel hátulról indulva a 6. pozícióban (elölről

[2]Az irodalomjegyzékben található könyvben az algoritmus NEXPER néven található meg.



6.3. ábra. A lexikografikus bejárás blokkdiagramja

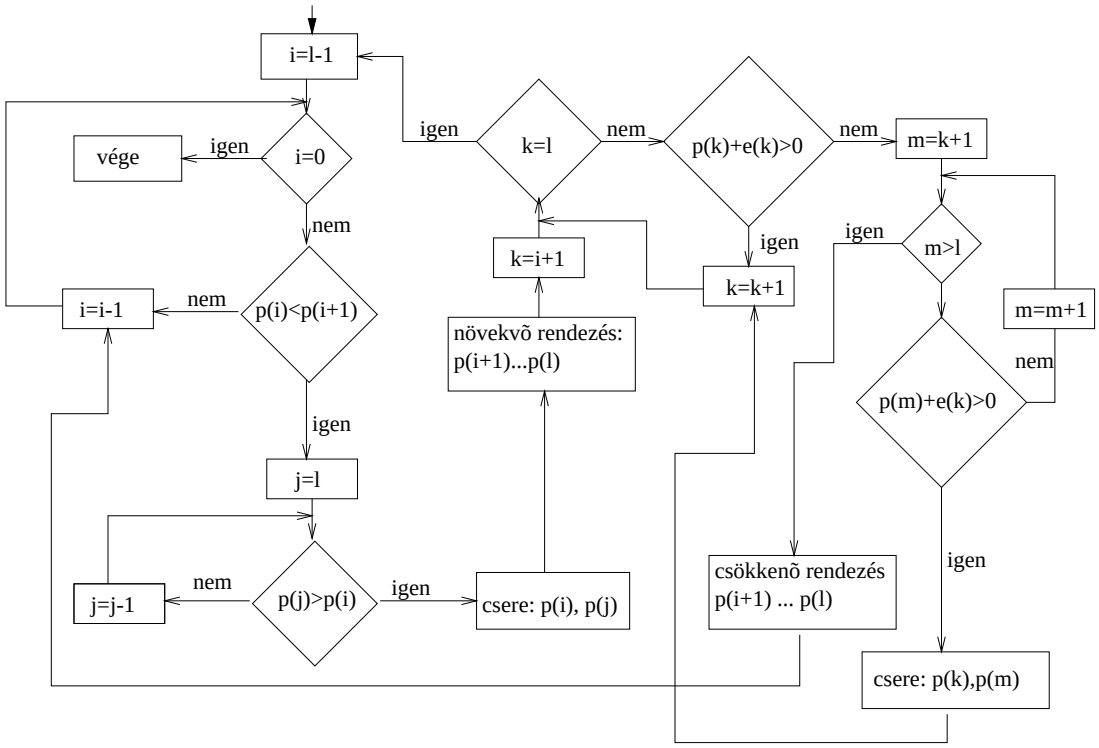
is a 6) álló elemre (p(i=6)=6) lesz először igaz, hogy kisebb az utána álló elemnél (9) ezért azt a legkisebb elemet kell megkeresni, ami nagyobb ennél az értéknél. Ez az 8. pozícióban álló 7-re lesz igaz (p(j=8)=7). Ezt a két elemet meg kell cserélni és, a 7. elemtől az összeset nagyság szerint növekvő sorrendbe kell állítani. Tehát a rákövetkező elem a $(4, 6, 7, 4, 4, 7, 3, 4, 4, 6, 9)$ kombináció lesz.

A fent bemutatott ismert bejáró algoritmust a mi esetünkben módosítani kell, hiszen tudjuk, hogy nem szabad olyan permutációt előállítanunk, aminek van olyan eleme, amit az egyenletes kombináció ugyanezen pozícióban lévő elemével összeadva negatív számot kapunk (hisz ekkor a generált módosítóhoz tartozó kombináció tartalmazna negatív elemet). Például ha az egyenletes kombináció $(2, 1, 2, 2)$ és a bázismódosító $(-2, -1, 0, 3)$, akkor ennek nem állíthatjuk elő azon permutációit, amelyekben a -2-es elem a második helyen áll.

Mivel a végső cél a kombinációhoz tartozó valószínűség, továbbá a kombináció által leírt helyzethez tartozó elemi játék várható nyereményének meghatározása, könnyelműen azt mondhatnánk, hogy ha a kombináció szabályt sért, akkor e két értéket ne számítsuk ki. E módszer igen rossz hatékonyságú. Gondoljunk arra az esetre, amikor az egyenletes kombinációnak csak az első pozíciójában áll kisebb elem a többinél (pl.:(4,5,5,5,5,5)). Ha ehhez a bázismódosítónk olyan lenne, hogy az első pozíciójában álló szabályt szeg (pl.:(-5,-2,0,1,3,4)) akkor az összes permutáció megszegné a szabályt, amíg ezen első pozícióban álló elemet át nem helyeznénk a második pozícióba. Amennyiben a 2.3. ... l-edik elemek mind különbözőek, akkor ez $(l-1)!$ felesleges permutáció előállítását jelentené. Ez az érték már közepes l-nél (kártyajátékokban l=13) is túl nagy szám ($\approx 5 \cdot 10^9$)

Látható, hogy a lexikografikus bejárásnál új permutációt mindig akkor állítunk elő, amikor találunk olyan elemet (i), ami kisebb az utána következőnél. Ekkor az

új permutáció az elemek cserélgetéseivel áll elő. Ha az aktuális permutáció nem szegett szabályt, akkor ez továbbra is igaz lesz a $p_1 \dots p_{i-1}$ elemekre, hiszen ezek értéke nem változnak. Probléma csak a $p_i \leq p_{i+1} \leq \dots \leq p_{i-1}$ elemekkel lehet. Ha a generált permutáció megszegi a szabályt, akkor egy olyan permutációt kellene találni, ami a legkisebb azok közül, amelyek nem szegik meg a szabályt, de nagyobbak a most generált szabályszegő permutációnál. Ilyen elemet a következő módon lehet előállítani: Nézzük meg, hogy p_i megszegi-e a szabályt. Ha nem, akkor vizsgáljuk a következő ($i+1$-edik) elemet, ha igen akkor keressük meg az a legkisebb, i-nél nagyobb indexű elemet amelyet ha az i-edik pozícióba mozgatnánk, akkor az nem szegné meg a szabályt. Ha van ilyen elem, akkor tegyük ezt, és vizsgáljuk az $i+1$-edik elemet. Ha nincs ilyen elem, akkor tegyük $p_i \dots p_l$ elemeket nagyság szerint csökkenő sorrendbe, és generáljuk a következő permutációt. Ezt azért kell megtennünk, mert ha $p_i \dots p_l$ semmilyen keverésével nem tudunk szabályos permutációt előállítani, akkor új permutáció előállításával ezen elemek közé legalább egy új elem kerül, ami segítségével már lehet, hogy orvosolni tudjuk a problémát. A következő ábrán látható az algoritmus blokkdiagramja.



6.4. ábra. A módosított lexikografikus bejárás blokkdiagramja

A következőkben szabályszegés kiküszöbölésére két példát mutatok. Az elsőben egyszerű átrendezéssel feloldható a szabályszegés, míg a másodikban új kombinációt kell generálni. Az egyenletes kombináció minkét esetben a $(2, 2, 2, 1, 2)$ kombináció lesz.

Az $(1, -1, -1, 3, -2)$ rákövetkező eleme a $(1, -1, 3, -2, -1)$ lenne, ahol a -2 szabályt sért, de van olyan elem, amivel megcserélve ez megoldható. A helyes rákövetkező permutáció tehát a $(1, -1, 3, -1, -2)$ lesz. A második példában az alappermu-

táció a $(2,-1,-2,3,-2)$, aminek rákövetkező eleme a $(2,-1,-3,-2,-2)$ lenne, de ez szabályt sért a 4. pozícióban. Mivel az 5. pozícióban ugyanilyen elem áll, így új permutációt kell generálnunk, ami a $(2,-3,-2,-2,-1)$ lenne, ami habár még mindig szabályt sért, ezt már orvosolni lehet egy cserével. A helyes rákövetkező elem tehát a $(2,-3,-2,-1,-2)$ lesz

Az egyes bázismódosítók helyes permutációinak előállításán felül, meg kell határozni még a permutációk előfordulásainak valószínűségét.

Az $\binom{n}{k+1} = \binom{n}{k} \cdot \frac{n-k}{k+1}$ és az $\binom{n}{k-1} = \binom{n}{k} \cdot \frac{k}{n-k+1}$ felhasználásával könnyű bebizonyítani, hogy

$$p(k_1 \dots k_i + 1 \dots k_j - 1 \dots k_l) = p(k_1 \dots k_i \dots k_j \dots k_l) \cdot \frac{n-k_i}{k_i+1} \cdot \frac{k_j}{n-k_j+1}. \quad (6.21)$$

Ennél általánosabban az is igaz, hogy amennyiben $\sum_{i=1}^{l} k_i = 0$ akkor:

$$p(e_1+k_1, e_2+k_2 \dots e_l+k_l) = p(e_1, e_2 \dots e_l) \cdot m(e_1,k_1) \cdot m(e_1,k_1) \dots m(e_l,k_l) \quad (6.22)$$

ahol

$$\begin{aligned} m(e_i,k_i) &= \begin{cases} \frac{n-e_i}{e_i+1}\frac{n-(e_i+1)}{e_i+2}\dots\frac{n-(e_i+k_i-1)}{e_i+k_i} & \text{ha } k_i > 0 \\ \frac{e_i}{n-e_i+1}\frac{e_i+1}{n-e_i+2}\dots\frac{e_i+k_i+1}{n-e_i-k_i} & \text{ha } k_i < 0 \\ 1 & \text{ha } k_i = 0 \end{cases} \\ &= \begin{cases} \sum_{i=0}^{k_i-1} \frac{n-(e_i+i)}{(e_i+i)+1} & \text{ha } k_i > 0 \\ \sum_{i=0}^{k_i+1} \frac{e_i+i}{n-(e_i+i)+1} & \text{ha } k_i < 0 \\ 1 & \text{ha } k_i = 0 \end{cases} \end{aligned} \quad (6.23)$$

Ha tehát az egyenletes kombináció és annak valószínűsége ismert, akkor tetszőleges módosítóhoz tartozó kombináció előfordulásának valószínűségét a fenti képlet alapján meg lehet határozni. Minden helyes kombináció előállítása után a valószínűség explicit kifejezése helyett egy trükkösebb módszert mutatok, amivel a szorzások és osztások számát nagymértékben csökkenteni lehet.

A trükk lényege, hogy egy új kombináció előállításakor a $k_1, k_2, \dots k_{i-1}$ értékek nem változnak, tehát felesleges az ezekhez tartozó módosítótörteket (m_i) újra kiszámolni. Vegyünk fel ezért egy $l+1$ elemű tömböt. 0. eleme tartalmazza az egyenletes kombináció valószínűségét, i-edik eleme pedig az $i-1$-edik elem és a k_i-hez tartozó módosítótört szorzatát. Ekkor tetszőleges kombináció előfordulásának valószínűségét az utolsó elem fogja tartalmazni. A permutáció során egy új elem előállításánál a régi elemhez tartozó tömbből állítsuk elő az új elem tömbjét. Ehhez csak a $i, i+1, \dots l$-edik elemet kell újra meghatározni, hiszen csak ezek változtak.

A számítást még tovább lehet gyorsítani, ha az egyes $m(e_i,k_i)$ elemeket is tároljuk. Ezek száma maximum $2 \cdot l$, ami nem igényel nagy memóriát.

Ezzel a memóriabarát algoritmus minden részletét bemutattam. Az algoritmus félelmetes sebességéről és szinte nulla memóriaigényéről az alábbi szimulációs eredmények tanúskodnak. A szimuláció során $l = 13$ (hiszen általában 13 különböző kártya van egy csomagban) $n = 16$ (ez négy csomagnak felel meg), és az előírt összvalószínűség, tehát a vizsgált kombinációk valószínűsége 95% illetve 90%.

K	#bázis	állapottér	$\#\tilde{k}$, 95%-nál	arány	$\#\tilde{k}$, 90%-nál	arány
1	1	13	13	100%	12	92.3%
2	2	91	83	91.2%	77	84.6%
3	3	455	410	90.1%	371	81.5%
4	5	1,820	1,619	88.9%	1,495	82.1%
5	7	6,188	5,557	89.8%	5,001	80.8%
6	7	18,564	15,684	84.4%	14,412	77.6%
7	11	50,388	43,514	86.3%	36,425	72.2%
8	11	125,970	100,400	79.7%	92,610	73.5%
9	11	293,930	217,114	73.8%	191,867	65.2%
10	11	646,646	439,294	67.7%	380,679	58.8%
11	11	1,352,078	843,686	62.3%	696,584	51.5%
12	11	2,704,156	1,471,946	54.4%	1,186,921	43.8%
13	11	5,200,300	2,405,626	46.2%	1,843,104	35.4%
14	29	9,657,700	4,539,891	47.0%	3,287,546	34.0%
15	44	17,383,860	7,728,670	44.4%	6,104,855	35.1%
16	59	30,421,755	13,815,785	45.4%	9,889,785	32.5%
17	103	51,895,922	22,048,758	42.4%	15,461,451	29.7%
18	103	86,493,056	33,597,169	38.8%	26,236,651	30.3%
19	103	141,119,342	51,200,572	36.2%	38,205,023	27.0%
20	124	225,786,853	81,499,098	36.0%	52,372,547	23.1%

Felhívnám a figyelmet arra, hogy a mintegy 222 millió $p(k_1, k_2 \dots k_l) = \frac{\binom{n}{k_1}\binom{n}{k_1}\dots\binom{n}{k_l}}{\binom{N}{K}}$ típusú kifejezést az algoritmussal még egy átlagos gép is két és fél perc alatt számolta ki. Ahogy azt már említettem az alapalgoritmusnak K=13-14-nél már olyan memóriaigénye volt, hogy azt átlagos gépen futatni nem lehetett. A táblázat is mutatja, hogy a fejlesztett algoritmus K=20-nál is csak 124 darab bázismódosítónak foglal helyet, ami $124 \cdot 2 \cdot 14 < 3.5$ kbyte. Tehát K növelésével csak a futási idő nő, a memóriaigény alig.

6.3.7. A nyeremény globális várhatóértéke a Black Jack esetében

A 6.1 tétel kimondja, az optimális globális stratégia nagyon egyszerű: tegyünk kis tétet, ha az aktuális kombináció által leírt leosztás várható nyereménye negatív, és nagy tétet, ha pozitív. A minimális és a maximális tét kaszinóról kaszinóra változik, így a:

$$E^{\text{globális}}_{\text{pozitív}} = \sum_{k_1=1}^{n}\sum_{k_2=1}^{n}\dots\sum_{k_{13}=1}^{n} E^{opt}_{k_1,k_2,\dots k_{13}} \text{ ha } E^{opt}_{k_1,k_2,\dots k_{13}} > 0$$

$$E^{\text{globális}}_{\text{negatív}} = \sum_{k_1=1}^{n}\sum_{k_2=1}^{n}\dots\sum_{k_{13}=1}^{n} E^{opt}_{k_1,k_2,\dots k_{13}} \text{ ha } E^{opt}_{k_1,k_2,\dots k_{13}} < 0$$

értékeket próbálom közelíteni. Ezekből adott tétszabályok mellett a teljes várható nyeremény az $E^{\text{globális}} = \text{tét}^{\max} E^{\text{globális}}_{\text{pozitív}} + \text{tét}^{\min} E^{\text{globális}}_{\text{negatív}}$ azonnal adódik.

Ebben a fejezetben két módszert ismertettem, az egyes várható ételek közelítésére, ezért ezek eredményeit hasonlítom össze. A szimulációs várhatóérték képzésnél 1 millió SHUE (a SHUE 6 csomag 80%-ának megfelelő kártya, azaz körülbelül 35-40 kiosztás (elemi játék)). Ezek alapján:

$$E_{\text{pozitív}}^{\text{globális}} = 0.004735$$

$$E_{\text{negatív}}^{\text{globális}} = -0.006557$$

A poli-hipergeometrikus eloszlásnál K=1,2 ... 40-re határoztam meg kombinációkat, és az azokhoz tartozó valószínűségeket és várható nyereményeket. Ezzel a közelítéssel az alábbi eredményeket kaptam:

$$E_{\text{pozitív}}^{\text{globális}} = 0.004468$$

$$E_{\text{negatív}}^{\text{globális}} = -0.006973$$

Látható, hogy az eltérés kicsi (7% alatt van). Nyilván minél több leosztást szimulálok, illetve minél több kombinációt vizsgálok annál pontosabb eredményt kapok.

A Black Jack játékkal az eredmények szerint hosszútávon is lehet nyerni. Ehhez az kell, hogy változtatni tudjam a tétemet és kedvező helyzetekben a tétem legalább 1.38-szeresét tudjam betenni. Mivel a legtöbb kaszinóban a minimum tét \$1 és \$5 között van a maximális pedig \$100 és \$200 között, ezért ez nem jelent igazi megszorítást. Az $E_{\text{pozitív}}^{\text{globális}}$ és $E_{\text{negatív}}^{\text{globális}}$ értékek megadásával a Black Jack játékban az elméleti maximális nyereményt kaptuk meg. A bevezetőben említett 68 könyv bármelyikében, a pusztán számítógépes szimulációkon alapuló stratégiák várható nyereménye messze elmarad ettől az optimális értéktől.

Az optimális stratégia és az várható nyereményének meghatározása rengeteg számítással jár így számítógépet igényel. A kaszinókba azonban bármilyen elektronikus eszközt bevinni tilos, így az optimális nyerés csak elméleti nyerés. Gyakorlatban alkalmazható módszert, az $E_{k_1,k_2\ldots k_{10}}^{opt} = f(k_1, k_2 \ldots k_{10})$ függvény egydimenziós vetületeinek különböző tartományba eső meredekségének meghatározásából nyerhetünk. Diplomadolgozatomban erről bővebben nem írok, hiszen ez a terület már nagyon távol esik az általánosan alkalmazható stratégiakeresés/elemzés területétől. Látni kell azonban, hogy ezzel a módszerrel sokkal hatékonyabb eredményre juthatok, mint a hasraütésszerűen kitalált, és azután szimulációs eredményekkel igazolt stratégiákkal.

7. fejezet

Összefoglalás

Diplomadolgozatomban stratégiakereső módszereket és alkalmazási példákat mutattam be. Látható volt, hogy az egyes módszerek különböző előnyökkel és hátrányokkal rendelkeztek. Éppen ezért nem rangsorolhatjuk őket és nem emelhetünk ki egyet sem mint a minden probléma megoldóját.

A legjobb megoldás, az egyes módszerek együttes, hibrid alkalmazása. Hogy lássuk mennyire igaz ez, nézzük a Black Jack példáját, ahol az elemi optimális stratégia várható nyereményének meghatározásához egy neurális hálózat építése az ideális, amit a gyorsított matematikai eszközökkel generált tanítópontokkal taníthatunk be. A következő szinten pedig szimulációs vagy a poli-hipergeometrikus eseménytér-bejáró algoritmus segítségével határozhatnánk meg az optimális globális stratégia jóságát (várható nyereményét). Ennél a játéknál tehát mindhárom alap (szimuláció, Mesterséges Intelligencia, valószínűségszámítás) stratégiakereső-elemző módszert együttesen kell használni.

A stratégiakereső-elemző módszerek nem kizárólag valószínűségi játékokra alkalmazhatóak, hanem bármilyen olyan elemi folyamatra, vagy irányítási problémára ami megfelel a modellünknek. A modellre igaz volt, hogy egy véges állapotteret kellett feltérképezni, ahol az állapotátmenetek sztochasztikusak. A diplomadolgozatomban ezt tovább általánosítottam úgy, hogy azt egyes állapotátmeneteket leíró valószínűségi változók is változnak, valamilyen ismert halmaz elemeinek függvényében. Így jutottam el a poli-hipergeometrikus eloszláshoz, és ahhoz az igényhez, hogy ennek az eloszlásnak az eseményterét gyakorlatban is megvalósítható algoritmussal bejárjam.

Az általam felsorolt módszerek listája nem teljes, továbbá a már bemutatottak is további kutatási lehetőségeket rejtenek magukban. Ez leginkább a poli-hipergeometrikus tér bejárására szolgáló algoritmusra igaz. Itt egy olyan módszert dolgoztam ki, amely helyességéről nem csak szimulációs eredmények, hanem matematikai tételek is tanúskodnak. Az is kiderült, hogy az algoritmus továbbfejlesztésére és javítására vannak még lehetőségek, így további kutatások valószínűleg meg jobb eredményekre vezetnének.

Úgy gondolom, eredményeim igazolják, hogy a hasonló, véges állapotterű (sztochasztikus) játékra vezethető folyamatok tanulmányozására, elemzésére érdemben használhatóak az itt bemutatott módszerek és eljárások. Az alapvető összetevők, a matematikai modell felállítása, majd ehhez kapcsolódó stratégiakeresés/értékelés

a mesterséges intelligencia eszközeivel (reinforcement learning, neural networks), illetve a valószínűségszámítási módszerek pontos, valamint közelítő alkalmazásával, más hasonló helyzetekben is eredményesnek bizonyulhatnak. Emellett fontos szerep jut a gyors algoritmusoknak is, amelyek segítségével a felmerülő bejárási/generálási problémák hatékonyan megoldhatóak.

Vizsgálataim fontos tanulsága, hogy a nagyobb, gyakorlatias méretű feladatoknál több terület módszereinek az ötvözésével kaphatunk használható megoldásokat.

8. fejezet

Függelék

A következőkben a poli-hipergeometrikus állapotteret bejáró gyorsított, memóriabarát algoritmus kódja olvasható. A első segédeljárás generálja a bázismódosítók permutációit, a második az egyes bázismódosítókat. A harmadik eljárás a főeljárás, ami egyrészről előállítja az egyes K-khoz tartozó egyenletes kombinációkat, azok valószínűségeit, továbbá meghívja a segédeljárásokat.

8.1. táblázat. A permutal függvény paraméterei

pkszama	Adott K-hoz tartozó megvizsgált kombinációk darabszáma
pegyenletes	Az egyenletes kombináció valószínűsége
pszumma	A vizsgált kombinációk valószínűségeik összege
kegyenletes	Az egyenletes kombináció elemei
kmodosito	A bázismódosító elemei
lapeloszlas	A talonban található lapok kombinációja
eredmenytabla	Ebben tárolom $E^{\text{globális}}_{\text{pozitív}}$ illetve $E^{\text{globális}}_{\text{negatív}}$ értékeit
kisn	A kezdőtalonban található egyes lapok száma (n)
osszval	Vizsgált kombinációk valószínűségeik összegének minimuma
splitmelyseg	Az elemi várható érték számításának pontossága

```
void permutal(unsigned long& pkszama,double pegyenletes,double& pszumma,
              short int kegyenletes[14],short int kmodosito[14],
              unsigned int lapeloszlas[11],double eredmenytabla[2],
              short int kisn,double osszval,short int splitmelyseg)
{
   short int i,j,k,l,ok=1,ujkmodosito[14],tempmod[14],eltolbazis=-kmodosito[1],
             ki,kisegyen,temp;
   double puj[14],temperedmeny,*mtorttomb[2];

   kisegyen=kegyenletes[1];
   for (i=2;i<14;i++) if (kisegyen>kegyenletes[i])
   {
      kisegyen=kegyenletes[i];
```

```
   break;
}
for (i=0;i<2;i++)
{
   mtorttomb[i]=(double *) calloc(kmodosito[13]+eltolbazis+1,sizeof(double));
   for (j=1;j<14;j++)
   {
      mtorttomb[i][kmodosito[j]+eltolbazis]=1.0;
      if (kmodosito[j]>0) for (ki=i+kisegyen;ki<i+kisegyen+kmodosito[j];ki++)
                  mtorttomb[i][kmodosito[j]+eltolbazis]*=(kisn-ki)/(ki+1.0);
      else for (ki=i+kisegyen;ki>i+kisegyen+kmodosito[j];ki--)
                  mtorttomb[i][kmodosito[j]+eltolbazis]*=ki/(kisn-ki+1.0);
   }
}
for (i=1;i<14;i++) ujkmodosito[i]=kmodosito[i];
i=1;
k=2;
while (ujkmodosito[i]<0)
{
   if (ujkmodosito[i]+kegyenletes[i]<0)
   {
      j=i+1;
      while (j<14) if (kegyenletes[i]+ujkmodosito[j]>-1) break;
      else j++;
      temp=ujkmodosito[j];
      for (l=i;l<j;l++) ujkmodosito[l+1]=ujkmodosito[l];
      ujkmodosito[i]=temp;
   }
   i++;
}
puj[0]=pegyenletes;
do
{
   if (ok)
   {
      for (l=k-1;l<14;l++)
      {
        puj[l]=puj[l-1]
                *mtorttomb[kegyenletes[l]-kisegyen][ujkmodosito[l]+eltolbazis]
      }
      pszumma+=puj[13];
      pkszama++;
      for (j=1;j<14;j++) lapeloszlas[vesszo[j]]+=kmodosito[j];
      varhatonyeres(lapeloszlas,temperedmeny,splitmelyseg);
      if (temperedmeny>0) eredmenytabla[0]+=puj[13]*temperedmeny;
      else eredmenytabla[1]+=puj[13]*temperedmeny;
```

```
   for (j=1;j<14;j++) lapeloszlas[vesszo[j]]-=kmodosito[j];
   k=12;
}
if (ujkmodosito[k]<ujkmodosito[k+1])
{
   for (j=13;j>k;j--) if (ujkmodosito[j]>ujkmodosito[k]) break;
   temp=ujkmodosito[k];
   ujkmodosito[k]=ujkmodosito[j];
   ujkmodosito[j]=temp;
   k++;
   for (l=k;l<14;l++) tempmod[l]=ujkmodosito[l];
   for (l=k;l<(14+k)/2;l++)
   {
      temp=ujkmodosito[l];
      ujkmodosito[l]=ujkmodosito[13+k-l];
      ujkmodosito[13+k-l]=temp;
   }
   i=k;
   ok=1;
   while (i<14 && ujkmodosito[i]<0 && ok)
   {
      if (ujkmodosito[i]+kegyenletes[i]<0)
      {
         j=i+1;
         while (j<14) if (kegyenletes[i]+ujkmodosito[j]>-1) break;
         else j++;
         if (j==14)
         {
            ok=0;
            for (l=k;l<14;l++) ujkmodosito[l]=tempmod[l];
            k--;
         }
         else
         {
            temp=ujkmodosito[j];
            for (l=i;l<j;l++) ujkmodosito[l+1]=ujkmodosito[l];
            ujkmodosito[i]=temp;
         }
      }
      i++;
   }
}
else
{
   k--;
   ok=0;
```

```
      }
   }
   while (k>0 && pszumma<osszval);
   for (i=0;i<2;i++) free(mtorttomb[i]);
}
```

8.2. táblázat. A segedtotal függvény paraméterei

kmodosito	A bázismódosítók tömbje
hanyp	A bázismódosítók száma
nagyk	A kezdőtalonból kiosztott lapok száma (K)

```
void segedtotal(unsigned long& pkszama,double pegyenletes,double& pszumma,
        short int kegyenletes[14],short int (*kmodosito)[14],
        unsigned long& hanyp,unsigned int lapeloszlas[11], double eredmenytabla[2
        short int kisn,short int nagyk,double osszval,short int splitmelyseg)
{
   unsigned long k;

   for(k=0;k<hanyp;k++) if (pszumma<osszval)
   permutal(pkszama,pegyenletes,pszumma,kegyenletes,kmodosito[k],lapeloszlas,
            eredmenytabla,kisn,osszval,splitmelyseg);
   if (pszumma<osszval)
   {
      short int i,j,m,(*ujkmodosito)[14];
      unsigned long l,ujhanyp=0,korlat=2*hanyp;
      ujkmodosito=(short int (*)[14]) calloc(korlat,14*sizeof(short int));
      for(k=0;k<hanyp;k++)
      {
         i=1;
         while (kmodosito[k][i]<0) i++;
         if (kmodosito[k][i]>0) i--;
         do
         {
            if ((i==1 || kmodosito[k][i]>kmodosito[k][i-1]) &&
                 -kmodosito[k][i]<(nagyk-i+13)/13)
            {
               kmodosito[k][i]--;
               j=13;
               while (kmodosito[k][j]>0) j--;
               if (kmodosito[k][j]<0) j++;
               do
               {
                  if (j==13 || kmodosito[k][j]<kmodosito[k][j+1]
```

```
                    {
                       kmodosito[k][j]++;
                       l=0;
                       while (l<ujhanyp)
                       {
                          m=1;
                          while (m<14) if (ujkmodosito[l][m]!=kmodosito[k][m])
                                          break;
                          else m++;
                          if (m==14) break;
                          else l++;
                       }
                       if (l==ujhanyp)
                       {
                          if (ujhanyp==korlat)
                          {
                             korlat+=hanyp;
                             ujkmodosito=(short int(*)[14]) realloc(ujkmodosito,
                                                 korlat*14*sizeof(short int));
                          }
                          for (l=1;l<14;l++) ujkmodosito[ujhanyp][l]=
                                                          kmodosito[k][l];
                          ujhanyp++;
                        }
                        kmodosito[k][j]--;
                    }
                    j++;
                 }
                 while (j<14);
                 kmodosito[k][i]++;
              }
              i--;
           }
           while (i>0);
        }
        free(kmodosito);
        ujkmodosito=(short int(*)[14]) realloc(ujkmodosito,
                                          ujhanyp*14*sizeof(short int));
        hanyp=ujhanyp;
        segedtotal(pkszama,pegyenletes,pszumma,kegyenletes,ujkmodosito,hanyp,
                   lapeloszlas,eredmenytabla,kisn,nagyk,osszval,splitmelyseg);
     }
     else free(kmodosito);
  }
```

8.3. táblázat. A totalvegeredmeny függvény paraméterei

pakliszam	A kezdőtalonban található paklik száma
maxnagyk	Vizsgálandó kombinációkhoz tartozó K-k maximuma

```
void totalvegeredmeny(short int pakliszam,short int maxnagyk,
              double eredmenytabla[2],double osszval,short int splitmelyseg)
{
   short int kisn,nagyn,nagyk=1,segedtomb[14],i,hanyelem=0,
             (*kmodosito)[14],kegyenletes[14];
   unsigned int lapelo[11];
   unsigned long hanyp,pkszama,osszespszam=0;
   double pegyenletes=1.0,pszumma,temperedmeny;

   for (i=1;i<14;i++) kegyenletes[i]=0;
   for (i=1;i<10;i++) lapelo[i]=pakliszam*4;
   lapelo[10]=pakliszam*4*4;
   lapelo[0]=pakliszam*4*13;
   kisn=pakliszam*4;
   nagyn=pakliszam*4*13;
   eredmenytabla[0]=0.0;
   eredmenytabla[1]=0.0;

   varhatonyeres(lapelo,temperedmeny,splitmelyseg);
   if (temperedmeny>0) eredmenytabla[0]+=pegyenletes*temperedmeny;
   else eredmenytabla[1]+=pegyenletes*temperedmeny;
   do
   {
      kmodosito=(short int(*)[14]) calloc(1,14*sizeof(short int));
      if (!hanyelem)
      {
         cout<<"\nFigyelem a segedtomb kiurult!!!";
         hanyelem=13;
         for (i=1;i<14;i++) segedtomb[i]=i;
      }
      i=1+(short int) (hanyelem*(rand()/(RAND_MAX+1.0)));
      cout<<"\ni: "<<i<<" Az uj lap"<<segedtomb[i];
      pegyenletes*=nagyk*(kisn-kegyenletes[segedtomb[i]])/(nagyn-nagyk+1.0)/
                                          (kegyenletes[segedtomb[i]]+1.0);
      kegyenletes[segedtomb[i]]++;
      lapelo[vesszo[segedtomb[i]]]--;
      lapelo[0]--;
      segedtomb[i]=segedtomb[hanyelem];
      hanyelem--;
```

```
      for (i=1;i<14;i++) kmodosito[0][i]=0;
      hanyp=1;
      varhatonyeres(lapelo,temperedmeny,splitmelyseg);
      pszumma=0.0;
      pkszama=0;
      segedtotal(pkszama,pegyenletes,pszumma,kegyenletes,kmodosito,hanyp,
                 lapelo,eredmenytabla,kisn,nagyk,osszval,splitmelyseg);
      osszespszam+=pkszama;
      nagyk++;
      cout<<"\n\nK="<<nagyk-1<<"-hoz "<<pkszama<<" darab p-t kell kiszamolni,
             \n amik lefedtek az allapotter "<<pszumma*100.0<<"%-at,
             \n amihez "<<hanyp<<" darab bazist kellett a memben tarolni";
      cout<<"\nAz aktialis totalis vegeredmeny:
             \nA pozitiv varhato nyeremenye: "<<eredmenytabla[0]<<
            "\ta negative pedig: "<<eredmenytabla[1];
      cout.flush();
   }
   while (nagyk<maxnagyk);
   cout<<"\nA totalis vegeredmenyhez "<<osszespszam<<" db p-t kell kiszamolni";
}
```

Irodalomjegyzék

[1] Rényi Alfréd. *Valószínűségszámítás.* Tankönyvkönyvkiadó, 1968.

[2] L. Comtet. *Advanced Combinatorics.* D. Reidel Publishing Co., 1974.

[3] William Feller. *Bevezetés a Valószínűségszámításba és Alkalmazásaiba.* Műszaki könyvkiadó, 1978.

[4] Rónyai Lajos, Ivanyos Gábor, and Szabó Réka. *Algoritmusok.* Typotex Kiadó, 1998.

[5] Tom M. Mitchell. *Machine Learning.* Boston: McGraw-Hill, 1977.

[6] Albert Nijenhuis and Herbert S. Wilf. *Combinatorial Algorithms.* Academic Press, 1975.

[7] Andrés Perez-Uribe. Learning to play black jack with artificial neural networks. Technical report, Logic System Laboratory, Swiss federal Institute of Technology-Lausanne, 1998.

[8] Don Schlesinger. *Blackjack Attack: Playing the Pro's Way.* RGE Publishing, 1997.

[9] Arnold Snyder. *Blackjack Wisdom.* RGE Publishing, 1997.

[10] Standford Wong. *Professional Blackjack.* Pi Yee Press, 1994.