Ising spinüveg genetikai algoritmussal

Dokumentáció a számítógépes szimulációk a statisztikus fizikában című tantárgy házi feladatához

A programot megíró hallgató:

- neve: Koics Dániel
- neptun-kódja: D1V90V
- hallgatói státusza: elsőéves, aktív, alkalmazott fizikkus mesterképzéses hallgató
- e-mail címe: koics.dani@gmail.com

A feladat kiírása

Tekintsünk egy *NxN*-es négyzetes mátrixot periodikus határfeltétellel. Legyen a Hamilton-függvény:

$$H = -\sum_{<i,j>} J_{ij} s_i s_j$$

Itt J_{ij} a csatolási együtthatók, s_i és s_j pedig a ±1 értékű spinek. Az összegzés csak az első szomszéd párokra történjen. A csatolási együtthatókat véletlen számgenerálással döntsük el az algoritmus elején, értékük legyen ±1, mégpedig ½ valószínűséggel. Határozzuk meg az egy rácspontra jutó alapállapoti energiát genetikai algoritmussal!

Használjunk P különböző realizációt, válasszuk közülük a *P/2* legkisebb energiájút a következő generációhoz. Generáljunk *P/2* véletlen gyereket véletlen génkeveréssel. Minden egyed két véletlen génje mutálódjék p= ¼ valószínűséggel. Teszteljük az algoritmust $N = 8$, *16* és $P = 100, 200, 400$ értékekre!

A genetikai algoritmusról és a kitűzött problémáról

A genetikai algoritmus lényege az, hogy az evolúciót próbáljuk utánozni egy probléma megoldása során. A megoldás több realizációját is létrehozzuk, s ezek úgy foghatók fel, mint egy faj különböző egyedei, melyek egy generációt alkotnak. Valamilyen mennyiség segítségével mérjük az egyes megoldások alkalmasságát, s ez alapján választjuk ki a következő generációban is életben maradó egyedeket. A kiszelektált egyedek helyét olyan új egyedekkel töltjük be, melyeket a megmaradó egyedekből génkeveréssel kapunk. Az új generáció létrehozásakor mindig két szülő génjeinek keverésével állítunk elő egy gyerek egyedet, miközben a mutáció mintájára néhány gént véletlenszerűen módosítunk.

A mi esetünkben a spinmátrix lehetséges kitöltései a lehetséges egyedek, egy adott realizáció genetikai kódja a spinjeinek állása, az alkalmasságot mérő mennyiség az (egy spinre jutó) energia. Az energia minimalizálása a cél, ezért minden generációban a legkisebb energiájú realizációkat tartjuk meg. Mutációként két spin ¼ valószínűséggel történő újra beállítását használjuk.

Mivel spinmátrixról van szó, s_i és s_j helyett valójában kétindexes s_{ij} és s_{kl} mennyiségekről, a kétindexes J_{ij} csatolási mátrix helyett pedig négyindexes J_{ijkl} mátrixról van szó:

$$H = -\sum_{<i,j,k,l>} J_{ijkl} s_{ij} s_{kl}$$

Mivel az összegzés csak a első szomszédokra történik, tekinthetjük úgy, hogy a J_{ijkl} mátrix egy ritka mátrix, melynek csak a $J_{i,j,i+1,j}$ és a $J_{i,j,i,j+1}$ elemei térnek el 0-tól. Ekkor a mátrix szétesik egy horizontális és egy vertikális részre, s a Hamilton-függvény az alábbi formában írható:

$$H = -\sum_{i,j=1}^{N} V_{i,j} s_{i,j} s_{i+1,j} - \sum_{i,j=1}^{N} H_{i,j} s_{i,j} s_{i,j+1}$$

Megjegyzendő, hogy ekvivalens az *N+1* és az *1* index.

A program szerkezeti felépítése

A program felépítése objektumorientált szemléletű, kihasználva ezáltal a c++ nyelv nyújtotta lehetőségeket. A három számparaméteren túl fontos lehet a mutáció valószínűségének állítása, ezért ez az érték (pontosabban a reciproka) a forráskód elején egy makró segítségével (*#define INVPMUT 4*) állítható.

A program felépítése:

- Csatolási mátrixot tartalmazó objektumok osztályának definíciója (*coupler*)
  - Mezők:
    - A két, egészértékű mátrixot összetartó másodrenű pointerek (*Horizontal, Vertical*)
    - A mátrixok méretét tartalmazó egész típusú változó (*Size*)
    - Az objektum használhatóságát jelző logikai változó (*Valid*)
  - Tagfüggvények:
    - Paramétert nem váró, nem inicializáló konstruktor
    - Csak a méret paramétert váró konstruktor, mely dinamikus memóriafoglalással és véletlen sorsolással felépíti a két csatolási mátrixot
    - Másoló konstruktor a dinamikusan foglalt memóriaterületekkel együttesen történő másoláshoz
    - Értékadó egyenlőség operátor a dinamikusan foglalt tartalmak egyik objektumból másikba történő másolására
    - Mátrixkiíró függvény parancssorba és fájlba íráshoz egyaránt (*WriteM()*). Argumentumként egy *ofstream* objektumot vár, de ha nem kap semmit, a parancssorba ír.
    - Destruktor a lefoglalt memóriaterület felszabadításához
- A spinmátrix egyedi kitöltését tároló objektumok osztályának definíciója (*entity*)
  - Mezők:
    - Másodrendű egész-pointer a spinmátrix összetartására (*Matrix*)
    - A méretet tároló egész típusú változó (*Size*)
    - Az objektum használhatóságát indikáló logikai változó (*Valid*)
    - Dupla pontosságú lebegőpontos érték az egy spinre jutó energia tárolására (*Eps*)
    - 2 db, szintén entity objektumra mutató pointer az objektumok energia szerinti láncba fűzéséhez (*Up, Down*)

- Tagfüggvények:
  - Alapértelmezett konstruktor
  - Az objektumot a paraméterként megadott csatolási mátrix méretéhez igazító, s véletlen értékekkel feltöltő konstruktor, mely a csatolási mátrixok alapján kiszámíttatja az energiát
  - Az objektumot két szülőobjektumból megalkotó, s az energiát csatolási mátrix alapján kiszámíttató konstruktor
  - Másoló konstruktor
  - Értékadó egyenlőség operátor
  - Csatolási mátrix alapján energiát számító függvény dupla pontosságú lebegőpontos visszatérési értékkel, mely *NaN*, ha az energia nem számítható ki (*Hamilton()*)
  - Buborék függvény egész visszatérési értékkel (*fight()*). Ez a függvény akkor használható, ha több ilyen objektum van láncolva az *Up* és *Down* pointerek segítségével. Lényege, hogy addig helyezi egyre kisebb energiájú helyre az objektumot a láncban, amíg nagyobb energiájú objektumot lát maga előtt, vagy amíg el nem éri a lánc végét (vagy egy hibás elemet). Visszatérési értékként a hátrautasított objektumok számát, hiba esetén *-1*-et ad.
  - Mátrixkiíró függvény parancssorba és fájlba való íráshoz egyaránt (*WriteM()*). Ez is *ofstream* objektumot vár, s a parancssor az alapértelmezés.
  - Destruktor memória-felszabadítással.
- Az egyed objektumokat összefogva tároló objektumok osztályának definíciója (*population*)
  - Mezők:
    - egy pointer az mátrixegyed objektumokat (*entity*) tartalmazó dinamikus tömbhöz (*OurEntities*)
    - A csatolási mátrixok objektuma (*OurCoupler*)
    - további 2 db, szintén entity objektumra mutató pointer, melyek közül az egyik (*Highest*) a legmagasabb, másik (*Lowest*) a legalacsonyabb energiájú mátrix objektumot hivatott mutatni.
    - 2 db, más *population* objektumra mutató pointer (*Last* és *Next*), melyek a különböző generációkat tartalmazó objektumok láncba fűzésére szolgálnak
    - Egy logikai változó az objektum érvényességének, egy másik pedig az egyedobjektumok energia szerinti rendezettségének indikálására.
  - Tagfüggvények:
    - A mátrixok méretét és az egy generáción belüli realizációk darabszámát egész típusú értékben váró konstruktor, mely megalkotja a nulladik generációt. A függvény az egyedobjektumokat először keletkezésük szerint fűzi sorba, majd az energia szerinti rendezéshez meghívja a *Selection()* függvényt.

- Másoló konstruktor az aktuális generáció új generáció létrehozása előtti listára mentéséhez.
- Az új generáció létrehozásához, és a régi tárolásához meghívandó függvény (*generate()*), mely paraméterként egy egészet vár, visszatérése pedig logikai. A bemenő paraméterrel azt lehet megadni, hányszor jöjjön létre új generáció. A visszatérési érték magas, ha a folyamat során probléma adódik. Ha nem adunk meg paramétert, *1* az alapértelmezett érték.
  A függvénynek 2 feladata van. Először is az aktuális generáció másik objektumba mentése. Az új objektumhoz dinamikusan foglal memóriát, s a címet a *Last* pointerben tárolja. Amennyiben a pointer már eddig is tárolt címet, azt az új objektum *Last* pointerében helyezi el. Értelemszerűen értéket ad a megfelelő *Next* pointereknek is. Így végül egy olyan láncot kapunk, melyben a *Last* pointerek segítségével a régebbi, a *Next* pointerekkel pedig az újabb generációk irányába tudunk mozogni, s a kezdeti generáció a lánc végén található.
  Miután az átmenekítés megtörtént, következik a másik feladat. A függvény a paraméterként megadott számszor meghívja a génkeverő (*Mating()*) és a rendező (*Selection()*) függvényt.
- Rendező függvény logikai visszatérési értékkel (*Selection()*). Ciklikusan ismételve meghívja egymás után minden egyedobjektum buborék függvényét, s a ciklust addig ismétli, amíg azt nem tapasztalja, hogy már egyik egyed sem tud helyet cserélni a másikkal, vagy amíg hibát nem tapasztal. Utóbbi esetben magas, előbbi esetben alacsony értékkel tér vissza.
- Génkeverő függvény logikai visszatéréssel (*Mating()*). Végighalad az egyedobjektumok láncán, s az egyedek memóriacímeit egy tömbbe kigyűjti, hogy azok könnyen hivatkozhatóak legyenek. Ezután a populáció nagyobb energiájú felét új egyedekkel cseréli le. Az új egyedek létrehozásához a két szülőegyedet váró konstruktort használja, s ehhez a populáció kisebb energiájú feléből választ egyedeket véletlen sorsolással. Magas visszatérési értékkel jelzi, ha hiba történik az eljárás során (nem sikerül memóriát foglalni a pointereknek).
- Mátixkiíró függvény (*WriteM()*). A paraméterek s az egyedek parancssorba írása. Paraméterek és visszatérési érték nélkül. A függvény végighalad a generációk teljes láncán, s minden egyed mátrixát és egy spinre eső energiáját kiírja.
- Mentő függvény (*Save()*), mely a fájl elérési útját és nevét sztringként várja, s magas logikai értékkel jelzi a mentés sikertelenségét. Hasonlóképp működik, mint az egyszerű mátrixkiíró.

- A parancssori argumentumokat megkapó, s a felhasználóval kommunikáló főfüggvény (*_tmain()*). Miután megszerezte a szükséges paramétereket, létrehoz egy populáció objektumot, s a megadott generációszámszor meghívja az objektum generáló

függvényét, mégpedig paraméter nélkül. Végül meghívja az objektum mentőfüggvényét a paraméterként kapott sztringre.

A program használata

A program egy parancssori alkalmazás, mely kétféleképp hívható meg: a szükséges 4 paraméterrel, vagy azok nélkül.

Paraméterek nélkül meghívva kommunikálni kezd a felhasználóval, hogy kiderítse a paramétereket. Először a spinmátrixok méretét (*N*) kéri, majd az egy generációban lévő egyedek számát (*P*), azután a szimulálni kívánt új generációk számát, s végül a kimeneti fájl elérési útját és nevét kéri. (Utóbbit egyetlen sztringként.) A program minden egyes kérdésről csak akkor lép a következőre, ha számára megfelelő választ kap, ellenkező esetben újra felteszi a kérdést (végtelen ciklus). A párbeszéd és a program futása bármikor megszakítható egy magányos *„0”* karakter beírásával.

Ha mindent megadtunk neki, amire szüksége van, nyugtázza a kapott paramétereket, és elindítja az algoritmust. Az algoritmus sikeres befejeztét vagy kudarcát, valamint a mentés sikerét vagy kudarcát is nyugtázza. Minden osztály fel van készítve a legkülönfélébb lehetőségekre, így mindegyiknek megvan a maga hibaüzenete.

Ha paraméterekkel együtt hívjuk meg a programot, akkor a program neve (*Ising*) után 3 számot és egy sztringet kell írni a parancssorba, szóközzel elválasztva. A három szám rendre a spinmátrix mérete, a generációnkénti egyedszám, s az összes új generáció száma. A sztring a fájl elérési útja és neve. A program ekkor rögtön a paraméterek nyugtázására ugrik, és elkezdi az algoritmus futtatását.

Minden más hívásra a program kiírja, hogy nem a megfelelő paraméterekkel lett meghívva, és anélkül kilép, hogy elvégezné a feladatot.

A kimeneti fájlban az eligazító megjegyzések – a html kódokhoz hasonlóan – „*<*” és „*>*” jelek közé írva láthatók. Ilyen a paraméterek felsorolása előtt és után a „*<Parameters>*” illetve „*</Parameters>*”, a csatolási mátrixok megadása előtt és után a „*<Couplers>*” ill. „*</Couplers>*”, generációk felsorolása előtt és után pedig a „*<Population>*” ill. „*</Population>*” felirat. Minden generáció előtt látunk egy „*<Generation #szám>*”, a generáció után pedig „*</Generation #szám>*” feliratot. A kezdeti, véletlenszerűen létrehozott generáció sorszáma *0*, a maximális sorszám pedig a paraméterként megadott határérték. A generációkon belül az egyes realizációkat a „*<Entity #szám>*” és „*</Entity #szám>*” üzenetek különítik el egymástól. Ezek a számok *1*-től a paraméterként megadott értékig terjednek.

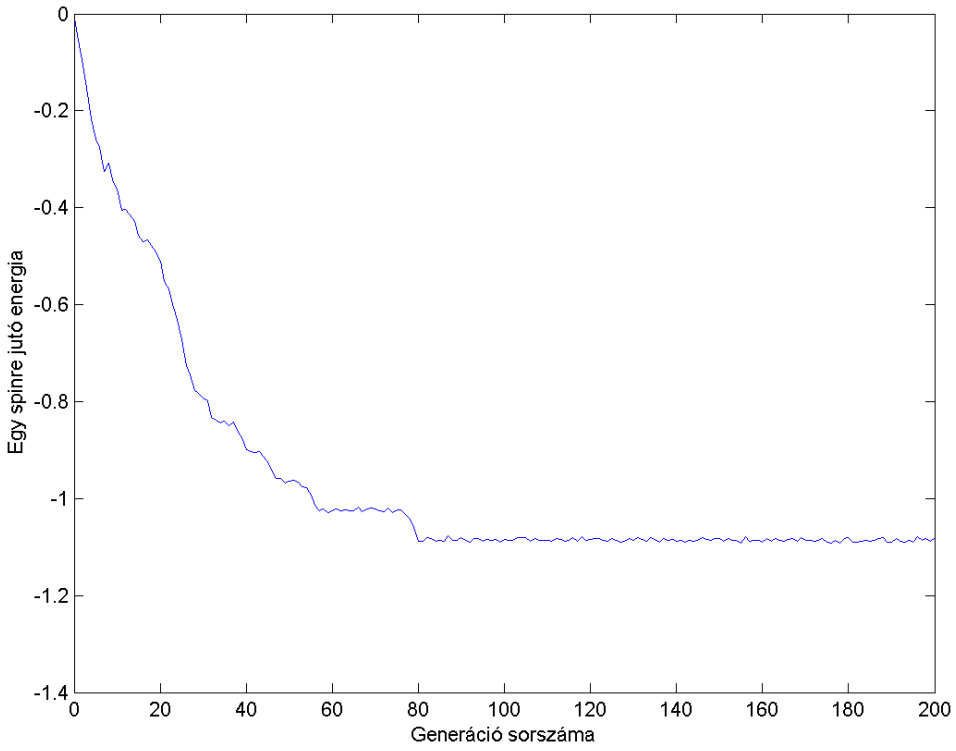
A paramétereket felsoroló részben a „*Size of entities = szám*”, a „*Number of entities per generations = szám*” és a „*Total number of mating-created generations = szám*” sorok olvashatók. A csatolási mátrixokat leíró részben a „*Vertical:*” felirat alatt a függőleges, míg a „*Horizontal:*” felirat alatt a vízszintes irányban első szomszéd spinekre vonatkozó csatolási mátrix elemei olvashatók táblázatos alakban. Az sorban ill. oszlopban első szám a sorban ill. oszlopban első és második spin közti, míg az utolsó mátrixelemek az utolsó és első spinek közti (tehát a periodicitást biztosító) kölcsönhatást kifejező együtthatók. Az egyes realizációkra vonatkozó részekben a „*Matrix =*” sort követi a spinmátrix, s azután pedig az „*Eps = szám*” sor következik, mely az egy spinre jutó energiát adja meg.

A program tesztelése

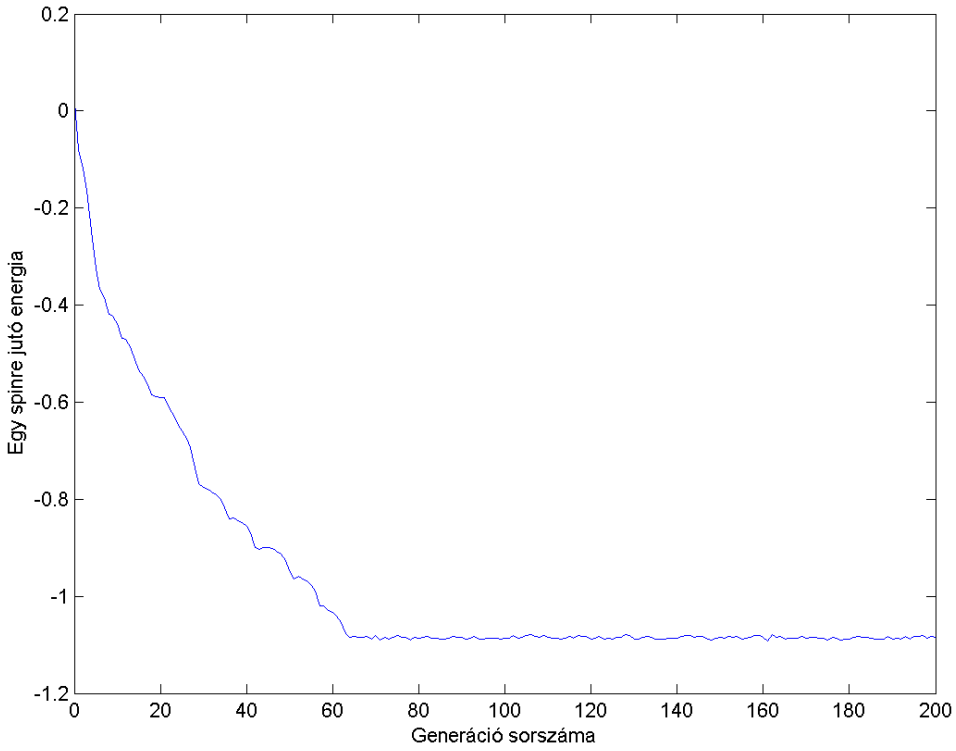
A teszteléshez írtam egy *MATLAB*-szkriptet, melybe beépítettem ugyanazt a párbeszédet, mint a konzolalkalmazásba. A különbség az, hogy ez a szkript nem maga hajtja végre az algoritmust, hanem az operációs rendszernek küldött paranccsal meghívja a konzolalkalmazást. Mikor visszakapja a vezérlést, megnyitja az eredményfájlt, s egy mátrixba kigyűjti minden generáció minden egyedének spinenkénti energiáját. Minden generációra átlagot számít, s a kapott átlagértéket ábrázolja a generáció sorszámának függvényében. Végül az ábrát menti. Így végeredményben grafikonon láthatóvá válik az energia alakulása, s következtetések vonhatók le az algoritmus hatékonyságáról. A feladat kiírásával ellentétben informatívabbnak láttam $N = 8$ esetén $P = 400$ helyett $P = 60$, $N = 16$ esetben pedig $P = 100$ helyett $P = 800$ ábrázolását. (Akkor hasonló a generációnkénti egyedszám és a genetikai kódhossz aránya.)

Az *8x8*-as mátrixokra kapott grafikonok:

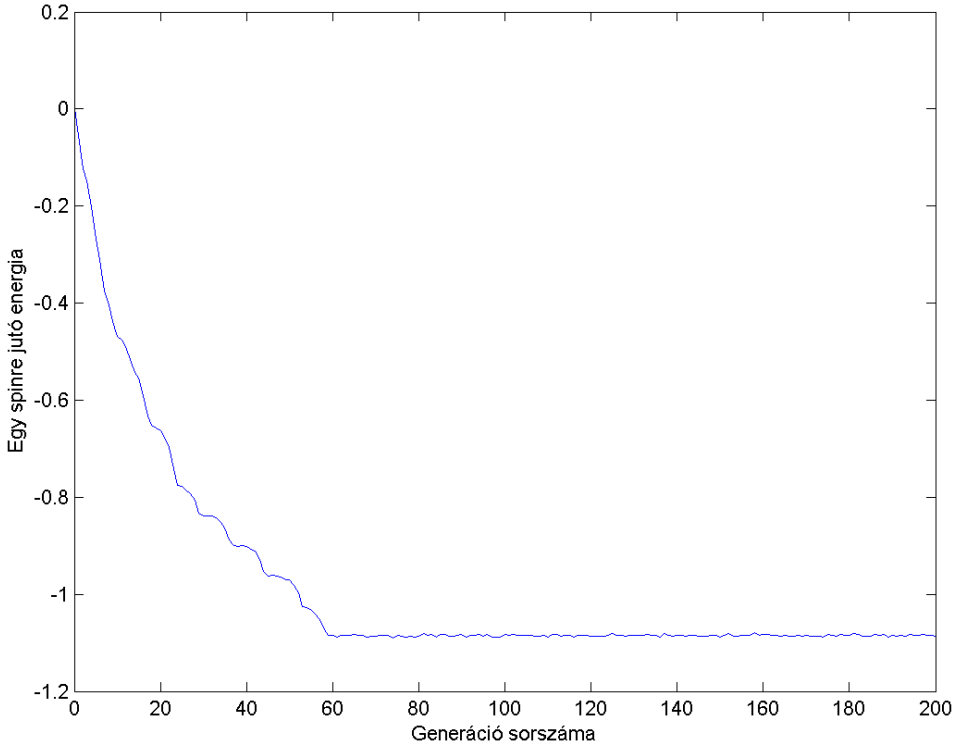
$P = 60$:



P = 100:

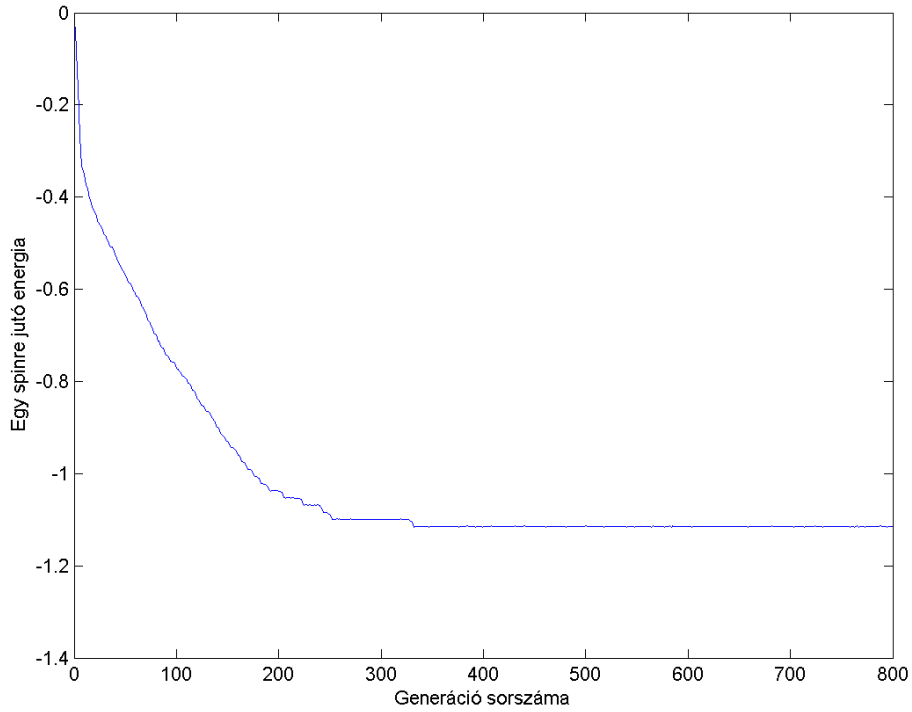

0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
Egy spinre jutó energia
0
20
40
60
80
100
120
140
160
180
200
Generáció sorszáma

P=200:

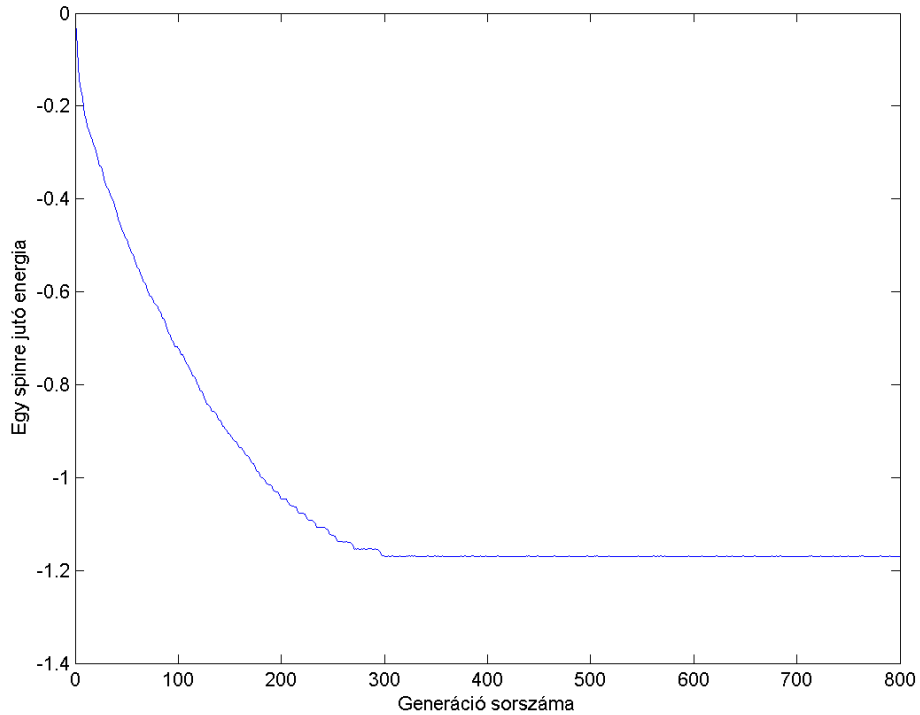

0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
Egy spinre jutó energia
0
20
40
60
80
100
120
140
160
180
200
Generáció sorszáma

A *16x16*-os mátrixokra:

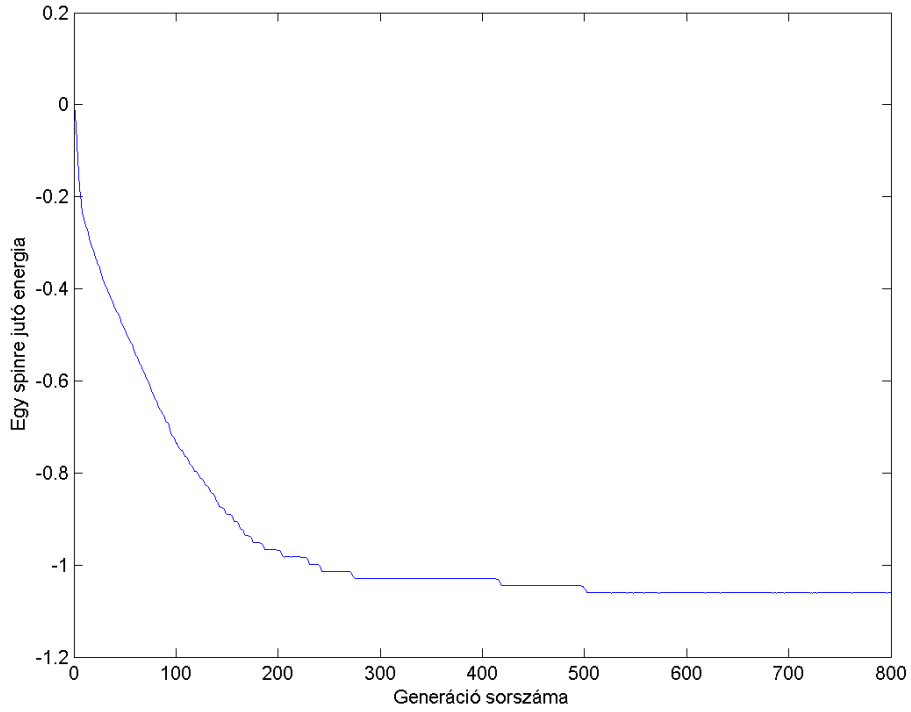
P=200:



P=400:

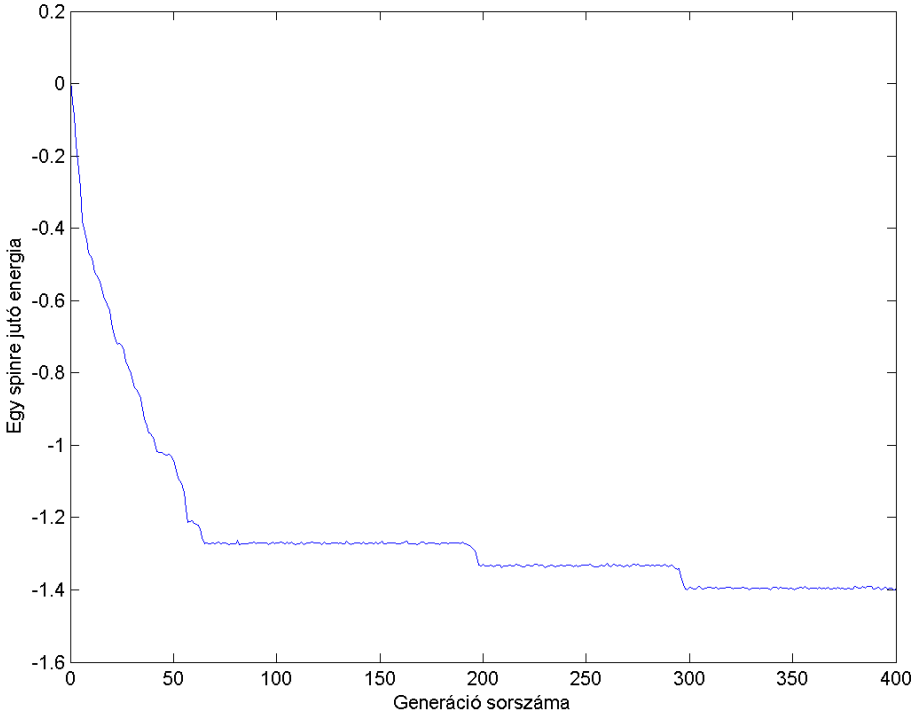


$P = 800$:

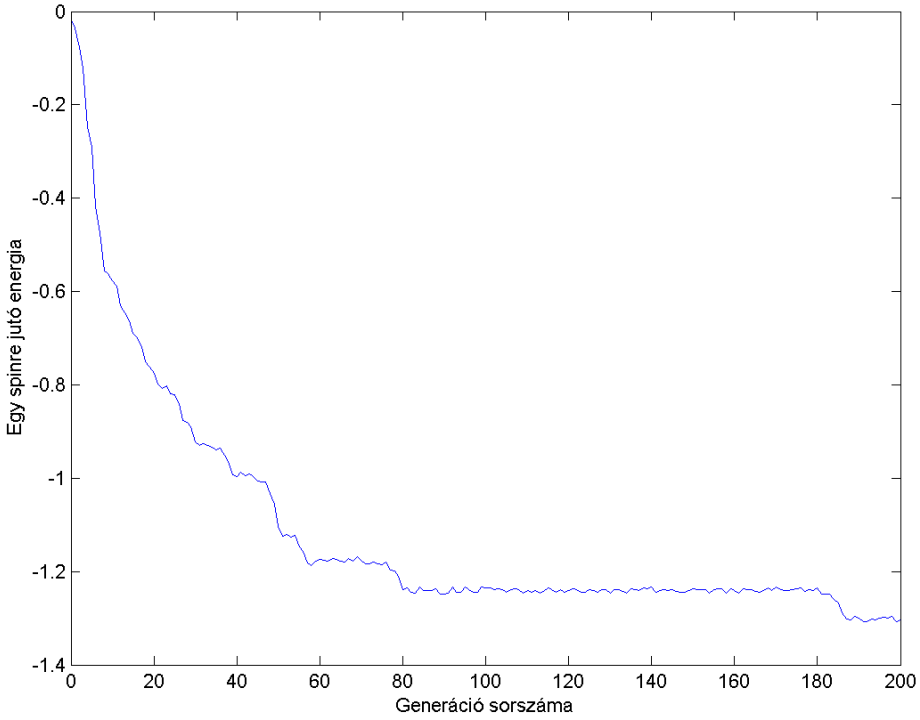


Megállapítható, hogy a függvényben töréspontot találunk, mely annál erősebb, minél kisebb a genetikai kód hossza. Továbbá a töréspont előtti szakasz annál szabályosabb, és a töréspont annál jobban látható, minél nagyobb generációnkénti egyedek száma. A töréspont helye épp a genetikai kód bitszámának környékén van. Vagyis egy véletlen egyedekből álló populációnak nagyjából annyi generációra van szüksége egy adott helyzethez való alkalmazkodáshoz, amennyi a genetikai kód bitszáma. Az alkalmazkodás olyan erős, hogy a fenti esetekben a végső, közel konstans szakaszokon csaknem minden egyed ugyanolyan genetikai állománnyal bír. Persze nem az összes egyed, s nem is konstans tökéletesen ez a szakasz, ennek oka pedig a mutáció. A nagyobb méretű genetikai kóddal rendelkező $N=16$ esetben kevésbé látható a mutáció hatása, mint $N=8$-ra. Ez nem meglepő, hiszen legfeljebb 2 bitre hat a mutáció, ami a kevesebb bitből álló kódra nagyobb hatással van.

Persze sokszor a töréspont után sem éri el azonnal az energiaminimumot a rendszer (főleg rövid genetikai kódok esetén). Például itt van az alábbi $N = 8$-as, $P = 200$-as eset:



Vagy egy $N = 8$-as, $P = 60$-as eset:



De még a fentebb ismertetett $N = 16$-os, $P = 800$-as ábrán is láthatók a töréspont utáni tartományt meg-megszakító lépcsők. Tanulmányozva a kimeneti fájlokat azt tapasztaljuk, hogy ezeken a tartományokon néhány azonos energiájú genetikai kód küzd egymás ellen.

Ami az esetleges továbbfejlesztési lehetőséget illeti: Érdemes lehet úgy módosítani az algoritmust, hogy előre megadott csatolási mátrixszal dolgozzon. Ekkor érdekes lenne minden egyes generációban meghatározni minden spin átlagértékét, az így kapott mátrixot imagescreen algoritmussal ábrázolni, s az egyes generációkhoz tartozó képeket filmmé fűzni. Ezzel a módszerrel azt tanulmányozhatnánk, hogy a csatolási mátrixokba bevitt mintázatok mennyi idő alatt jelennek meg a spinmátrixokban.

Konklúzió

Létrehoztam egy genetikai algoritmust, mely egy véletlen sorsolással eldöntött csatolási mátrixú spinüvegre genetikai algoritmussal becsüli a legkisebb energiát. Láttam, hogy valóban jó becslés adható ezzel a módszerrel, és a populáció fejlődésének egyes sajátságait is megfigyeltem. A feladatot összességében érdekesnek találtam.