VESZPRÉMI EGYETEM

Matematikai és Számítástechnikai Tanszék

DIPLOMADOLGOZAT

GENETIKUS ALGORITMUSOK és ALKALMAZÁSAIK

Témavezető: **Írta**:

Starkné Dr. Werner Ágnes Keveházy Balázs

Veszprém

2005

ELŐSZÓ

Diplomadolgozatom a genetikus algoritmusokról szól. Dolgozatomban ismertetem a genetikus algoritmusok működését különös tekintettel a kanonikus genetikus algoritmusokra, valamint azok alkalmazásaira.

Ezúton fejezem ki köszönetemet Dr. Strakné Werner Ágnes-nek a Veszprémi Egyetem Matematika és Számítástechnikai Tanszék docensének, aki előadásain felkeltette érdeklődésem a téma iránt, valamint hasznos tanácsaival segített, hogy diplomadolgozatom elkészülhessen.

TARTALOM JEGYZÉK

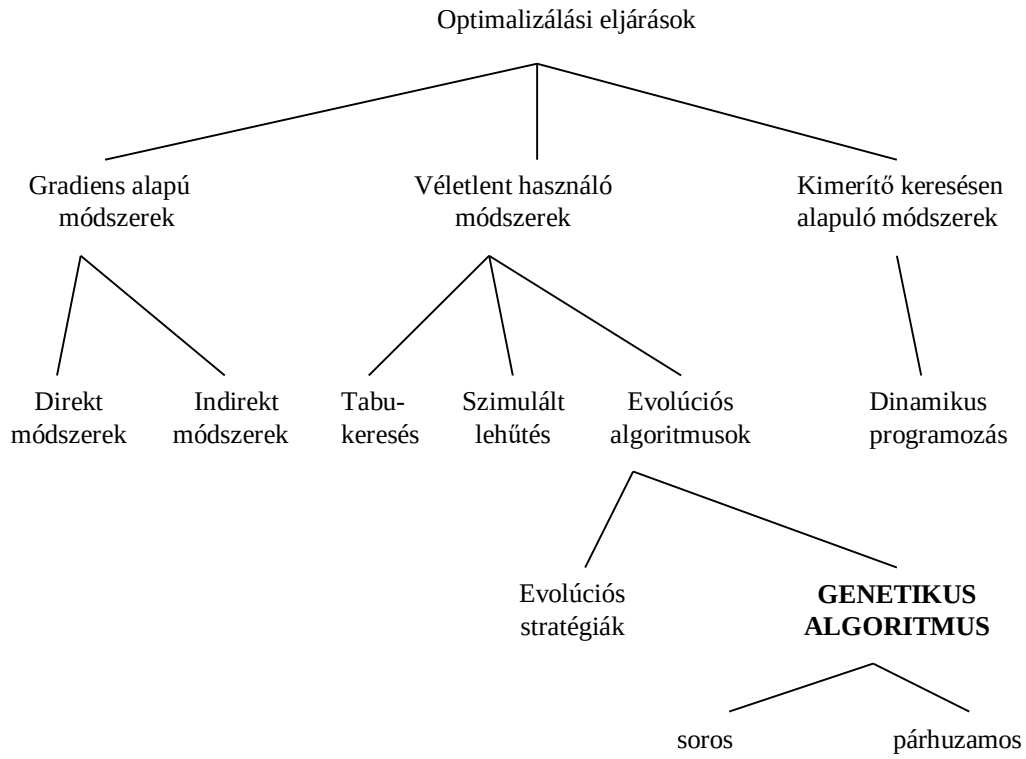
BEVEZETÉS

John Holland 1975-ben kidolgozott egy keresési és optimalizálási módszert, amely a darwini természetes kiválasztódás elveire épült. Ezt a módszert, amely az élőlények öröklődési folyamatait másolja, genetikus algoritmusoknak nevezzük.

A genetikus algoritmusok az utóbbi években komoly fejlődésen mentek keresztül, sokféle változatuk alakult ki, és matematikai megalapozásuk is teljesebbé vált.

Alkalmazása során a problémának nem egy, hanem több különböző optimális megoldását is nyújthatja, amelyből a felhasználó kiválaszthatja a neki tetsző példányt.

Az 1. ábrán látható a genetikus algoritmus helye az optimalizáló eljárások között. [ÁGyHV1]



1. ábra. Az optimalizálási eljárások csoportosítása

A genetikus algoritmusok bizonytalan és pontatlan információs környezetben is eredményesen alkalmazhatók, és hatékonyságukban sokszor felülmúlják a hagyományos optimalizálási eljárásokat.

Dolgozatomban kitérek a genetikus algoritmusok általános ismertetésére, működési elvére, valamint a gyakorlati alkalmazásaikra.

Dolgozatom fő témái:

- A genetikus algoritmusok általános ismertetése, szerkezete
- A genetikus algoritmusok biológiai háttere
- A kanonikus genetikus algoritmus
- Szelekciók
- Genetikus műveletek
- Reprodukciók
- Két feladat gyakorlat megvalósítása Pascal nyelven
- A genetikus algoritmusok gyakorlati alkalmazásai

Dolgozatomban részletesebben a genetikus műveletekkel valamint a kanonikus algoritmusokkal foglalkozom. Itt két példát mutatok be a kanonikus algoritmus működésére.

1. A GENETIKUS ALGORITMUSOKRÓL

A múlt század hatvanas éveiben merült fel az a gondolat, hogy az evolúcióban megfigyelhető szelekciós folyamatok mintájára olyan számítógépes modelleket lehetne létrehozni, amelyek képesek mérnöki (elsősorban optimalizálási) feladatok megoldására.

Egymástól függetlenül több próbálkozás is született. Németországban **Rechenberg** (1973) vezette be az **evolúciós stratégiának** nevezett módszert, melyet repülőgép-szárnyak valós paramétereinek optimalizálására használt. Később **Schwefel** (1991) továbbfejlesztette ezt az elgondolást. Az evolúciós stratégiák mai is a szelekcióalapú heurasztikának egy önállóan fejlődő ága. A többi próbálkozás mind Amerikában történt. **Fogel, Owens, és Walsh** egyszerűbb problémák megoldására szolgáló véges automaták automatikus kifejlesztésével kísérletezett. A kiindulási automaták állapot-átmeneti mátrixát véletlenszerűen megváltoztatták (azaz mutációt alkalmaztak). Ha az új automata rátermettebb volt az kiválasztásra került. Az új területnek az **evolúciós programozás** nevet adták, amely ma is művelt terület. Egy nagyon hasonló, de frissebb terület a **genetikus programozás**. Ez a genetikus algoritmusok egy speciális alkalmazási területe, amikor a cél meghatározott feladatokat végrehajtó számítógép programok automatikus fejlesztése. A programokat rendszerint LISP nyelven fejlesztik. A terület vezető alakja **John R. Koza** (1994).

A genetikus algoritmusok kifejlesztése **John H. Holland** nevéhez fűződik. Ő és diákjai alapozták meg a University of Michigan egyetemen a területet, amelynek kutatási eredményeit Holland foglalta össze 1975-ben. Az ő célja kezdetben nem az optimalizálási módszer kifejlesztése, hanem a szelekció és az adaptáció számítógépes és matematikai modellezése volt. [FI1]

A genetikus algoritmusok iránti érdeklődésnek talán a fő oka a darwini evolúció elmélet.

Az algoritmusok elméletében és gyakorlatában használt fogalmak majdnem mind a biológiából származnak. Valójában a genetikus algoritmus nem a darwini gondolatokból, hanem azokkal közös tőről fakad. A szakirodalom a genetikus algoritmust a természetes szelekción alapuló, a darwini elméletet utánzó módszerként kezeli. A genetikus algoritmus nem tekinthető az evolúciós folyamatok modelljének, sokkal közelebb áll a mesterséges szelekció modelljéhez, azaz úgy fogható fel, mint egy adott probléma számára a megoldások kinemesítésére szolgáló módszer. A fajok

evolúciója és a genetikus algoritmus közötti fő különbség az, hogy az előbbinél a legfontosabb szempont az adaptációra való képesség, azaz a valamilyen szempontból állandóan változó, bonyolult környezethez való alkalmazkodás, addig az utóbbinál a lényeg az optimalizáció, azaz egy rögzített szempontból a lehető legjobb megoldás keresése.

A genetikus algoritmus általános keresési terekben végez kevés tudást igénylő optimalizálást. Maga a módszer nagyon hasonlít az állattenyésztők módszerére. A megoldás populációk sorozatát állítja elő különböző operátorok segítségével, melyek közül legfontosabbak a rátermett szülők kiválasztását végző szelekció, és a szülőkből új megoldásokat előállító mutáció és rekombináció. A genetikus algoritmus alkalmazásához szükség van a keresési teret adó megoldások kódolására, ahol maga a kód analóg az örökítőanyaggal. A kódolás és az operátorok kiválasztása lehetőséget ad a területspecifikus ismeretek indirekt használatára.

A módszer fő előnye, hogy a számítástechnikában előforduló problémák egy nagyon széles területére alkalmazható, ugyanakkor nem használ területfüggő tudást, így akkor is működik, ha a feladat struktúrája kevéssé ismert.

A genetikus algoritmusok mindenhol alkalmazhatók, ahol a feladat sok lehetséges megoldásai közül a lehető legjobbat kell megkeresni. Valójában egy **globális optimalizáló.** A lehető legjobb megoldás keresésében egy értékelőfüggvény, más néven egy **rátermettségi függvény** (*fitness function*) értékét kell vizsgálni.

A genetikus algoritmus a megoldások egy **populáció**ját tarja fenn, azaz egyszerre több megoldással dolgozik. A genetikus algoritmus az aktuális populációból minden lépésben egy új populációt hoz létre úgy, hogy a szelekciós operátor által kiválasztott legrátermettebb egyedeken (szülőkön) alkalmazza a **rekombinációs** és **mutációs operátorokat.** Az alapgondolat az, hogy általában minden populáció az előzőnél rátermettebb egyedeket tartalmaz, így a keresés során egyre jobb megoldások állnak rendelkezésre. [SWÁ1]

2. A GENETIKUS ALGORITMIUS ÁLTALÁNOS SZERKEZETE, IMPLEMENTÁCIÓ

```
1. Hozzuk létre a P0 kezdeti populációt
2. t:=0
3. While not Kilépés (Pt)
4.          Pt' :=UjElemek (Pt)
5.          Pt+1 :=UjPopuláció (Pt',Pt)
6.          t:= t+1
```

Magyarázat:

Hozzuk létre a P_0 kezdeti populációt:

A kezdő populáció feltöltése gyakran véletlen elemekkel történik, de érdemes lehet valamilyen heurasztika segítségével viszonylag jó teljesítményű megoldásokból kiindulni. A populáció elemszáma futás közben változatlan. Konkrétan az 50-100 körüli értékek a tipikusak, de a genetikus programozásban gyakran több ezres populációval dolgoznak.

A **Kilépés (P_t)** függvény sokszor nem is függ a populációtól, mint ahogy a jelölés sejtetné, a tipikus módszer a már kiértékelt megoldások számának vizsgálata. Az algoritmus futása abban az estben áll le, ha egy előre adott mennyiségű különböző lehetséges megoldás rátermettségét kiszámoltuk. A megengedett kiértékelések konkrét száma függ a problémától, leggyakrabban 1000 és 500000 között van.

P_t':=UjElemek (P_t) függvény feltölti a **P_t'** populációt új elemekkel. Egy új megoldás előállítása legtöbbször úgy történik, hogy szelekció segítségével szülőket választunk, majd a rekombináció ezekből egy utódot hoz létre. Erre az utódra alkalmazható a mutáció, majd az új elem rátermettségének a meghatározása következik.

A **P_t'** populáció elemszáma nem feltétlenül azonos **P_t** elemszámával. Ha mégis azonos az **UjPopuláció (P_t',P_t)** függvény egyszerűen a **P_t'** populációt adja. Ekkor a genetikus algoritmus generációs.

Ha azonban **P_t'** elemszáma kisebb, a megfelelő mennyiségű elemet törli a régi populációból és helyükre **P_t'** elemei kerülnek. [SWÁ2]

3. A BIOLÓGIAI HÁTTÉR

Az evolúciós számítási problémák megoldásánál még a bonyolultabb esetekben is eredményesen használható az evolúció mechanizmusa. A számítási problémák egy jelentős része nagyszámú megoldási lehetőség tanulmányozását kívánja meg. A fehérjék tervezésénél (fehérjemérnökség) például hatalmas mennyiségű aminosav-szekvenciát kell átnézni, hogy a megadott tulajdonságú fehérjét kapjuk. Vagy a tőzsde területén komoly vizsgálódásokat kell folytatni szabályok, összefüggések keresésére, melyekkel megjósolhatjuk a jövőben várható árfolyam-emelkedéseket, -eséseket. Az ilyen keresési problémáknál előnyös és hatékony a párhuzamosítás, amikor a módszer számos különböző lehetőséget vizsgál meg egyidejűleg. Például ahelyett, hogy egyszerre csak egy aminosav-szekvenciát vizsgálnánk, párhuzamosan többel foglakozva sokkal gyorsabban eredményre juthatunk. Ehhez egyrészt gyors párhuzamos számítási kapacitásra van szükség, másrészt olyan intelligens stratégiára, melynek segítségével megadhatjuk a következő kiértékelésre szolgáló megoldássorozatot.

A számítási problémák megoldásánál elvárás az adaptáció képessége, azaz változó körülmények között is elfogadható teljesítmény. Ilyenek például a robotirányítási problémák, amikor a feladatot a robotnak változó környezetben kell elvégeznie, illetve a programnak alkalmazkodnia kell a különböző felhasználói igényekhez. Más problémák esetén olyan programra lehet szükség, amely képes teljesen új eljárásokat, megoldásokat létrehozni, például algoritmusokat generálni. Gyakran előfordul, hogy olyan összetett megoldásokra van szükség, amelyeket nehéz algoritmizálni. Erre sok példát lehet találni a mesterséges intelligencián alapuló módszerek között. A mesterséges intelligencia korai szakaszában úgy gondolták, hogy az lenne a hatékony megoldás, ha a programnak intelligenciát adó szabályokat kódolják. Ilyenek például a korai időkben kidolgozott szakértői rendszerek. Ezeknél a szabályokat manuálisan kódolták „fentről-lefelé” *(top-down)* stílusban. Mivel az intelligenciát adó szabályok túl összetettek ahhoz, hogy azokat teljes mélységben meg lehessen adni, manapság egy másik megközelítést részesítenek előnyben. A „lentről-felfelé” *(bottom-up)* módszereknél egyszerű szabályokkal kell csak közvetlenül foglalkozni. Az összetett eredmények, az intelligencia pedig erősen párhuzamosított rendszerek segítségével bontakozik ki ezen egyszerű szabályok egymásra hatásaiként. Ilyenek például a mesterséges neurális hálózatok, ahol egy-egy szabály jellegzetesen egy egyszerű nemlineáris küszöbfüggvényen (aktivációs függvény) keresztüli terjedést, vagyis a

kapcsolatok erősítését-gyengítését jelenti. Ezzel a módszerrel általában kifinomult megoldások nyerhetők a minta-felismerési és tanulási problématerületeken. A „lentről-felfelé” megközelítésre egy másik példa az evolúciós számítások, melyeknél a szabályok a természetes kiválasztásnak a keresztezéssel és/vagy a mutációval történő kombinálásában testesülnek meg. Ezáltal nehéz problémák esetén is kiemelkedő megoldások nyerhetők, amelyek képesek a változó körülményekhez is alkalmazkodni.

A biológiai evolúció alkalmas alapnak tűnik az ilyen jellegű problémák megoldására, hiszen az evolúció egy olyan módszert valósít meg, amely igen nagy számú lehetséges megoldás között képes „keresni”. Az élővilágban ez a roppant nagyszámú lehetőség a lehetséges génsorrendek összessége, a megkívánt megoldások pedig azok az egyedek, amelyek leginkább képesek a túlélésre, illetve a szaporodásra az adott környezetben. Az evolúciót úgy is tekinthetjük, mint olyan intelligens módszert, melynek segítségével teljesen új megoldásokat lehet kifejleszteni összetett problémák esetén. Erre egy biológiai példa az emlősök immunrendszere, ami a kórokozók elleni védelmet biztosítja. Az evolúció mechanizmusának a számítási kereső módszerekkel való összekapcsolása komoly lökést adott az evolúció megértéséhez. A mesterséges evolúciós módszerek a természetben zajló folyamatokkal azonos módon fejtik ki hatásukat. Összehasonlítva a természetes és a mesterséges problémák feltételrendszerét, könnyen beláthatók a hasonlóságok. A biológiai szervezetek jósága sok tényezőtől függ, például milyen sikerrel versenyez vagy működik együtt az őt körülvevő többi élőlénnyel. Lényeges, hogy a fitneszkritériumok folytonosan változnak, így az evolúció állandóan változó megoldáshalmazban keres. Éppen ez a változó feltételek melletti keresés képessége az, amire egy adaptív számítógépes programnál szükség van.

Az evolúció erősen párhuzamos keresési módszer, ahelyett, hogy egy fajjal foglalkozna egyidőben, a fajok milliót próbálja ki és változtatja meg egyszerre. Ugyanakkor, magas szintről nézve az evolúció szabályai egyszerűek: a fajok véletlenszerű változással (mutáció, rekombináció és más operátorok) fejlődnek, amit a természetes kiválasztódás követ, azaz csak a legjobb irányzatok képesek túlélni és szaporodni, s ezáltal átörökíteni a genetikai jellemzőiket. Mégis ezek az egyszerű szabályok felelősek túlnyomó részben azért a nagy változatosságért, amit az élővilágban láthatunk. [ÁGyHV2]

4. ALAPFOGALMAK

A genetikus algoritmusok könnyebb megértéséhez célszerű néhány alapfogalmat bevezetni, melyeket a továbbiakban is használható. Ezek az alapfogalmak kapcsolatban vannak a biológiai definíciókkal, bár ez esetek többségében leegyszerűsítettek.

Minden élőlény sejtekből épül fel, melyek mindegyike ugyanazon **kromoszómákat** tartalmazza. A kromoszómák, a DNS-füzérek olyanok, mintha az egyedek esszenciái lennének. A természetben egy vagy több kromoszóma összessége adja meg a genetikai előírást a szervezetek kialakulásához és működéséhez. Egy kromoszóma génekre osztható, melyek különböző funkcióhoz kötöttek, egy-egy fehérjét kódolva. Egy-egy gén egy-egy jellemzőt (például szemszín) ír le. A különböző lehetséges „jellemző-beállításokat" **génváltozatoknak** *(allél)* nevezik. Minden gén a kromoszóma egy adott helyén, **pozíciójában** *(locus)* helyezkedik el.

Sok élőlény több kromoszómát tartalmaz a sejtjeiben. A kromoszómák összessége az örökítőanyag teljes **génkészlet**e *(genome)*. **A genotípus** (genotype) a gének egy konkrét halmaza, adott génváltozatokkal. Ha két egyed génkészlete megegyezik, akkor azonos genotípusúak. A genotípus alapján fejlődik ki a magzat. Illetve a későbbi korban a **fenotípus** *(fenotype)* alapján, amely a szellemi és fizikai jellemzőket jelenti (szemszín, magasság, agyméret és az intelligencia). A természetben a fenotípus a teljes genetikai csomag és a környezet kölcsönhatása által kialakult szervezetként jelenik meg.

Azokat a szervezeteket, melyeknél megkettőzött kromoszómák vannak **diploidnak**, míg az egyszeres kromoszómaállománnyal rendelkező egyedeket **haploidnak** hívják. A természetben a legtöbb ivarosan szaporodó faj diploid, így az ember is, akinek 23 pár kromoszóma található sejtjeiben. Az ivaros szaporodás során rekombináció, keresztezés történik: mindkét szülő saját génállományának egy részével járul hozzá a gyermek génállományának kialakulásához. A szétváló kromoszómapárokból egy-egy jelenik meg az ivarsejtekben, majd a két szülőtől származó egyszeres kromoszómák újra párokat alkotnak, így újra kialakul a diploid kromoszómaállomány.

Haploid ivaros szaporodás esetén a gének a két szülői kromoszóma között cserélődnek ki, azaz a létrejövő új kromoszóma állomány bizonyos génjei az egyik, míg a többi a másik szülőtől származik. Az utódok kromoszómáin **mutáció** is történhet, melynek során az egyes gének, géndarabok megváltoznak, s ennek eredményeként módosul a szülőtől az utódhoz érkező genetikai információ.

Az élőlények fitnesze általában kétféle forrásból tevődik össze:

- függ annak a valószínűségétől, hogy az egyed megéri-e a reprodukciót (**életképesség**, *viability*)
- függ a létrejövő utódok számától (**termékenység**, *fertility*)

A genetikus algoritmusoknál a kromoszóma kifejezés általában egy, az adott problémára adható megoldást jelöl, gyakran bitsztringként kódolva. A gének lehetnek egyedülálló bitek vagy a szomszédos bitek csoportjai, melyek az adott megoldás egy bizonyos elemét, jellemzőjét kódolják (például többparaméteres függvényoptimalizálás esetén egy-egy paramétert külön-külön gén kódolhat). Az adott génpozícióhoz tartozó génváltozatok lehetnek (bináris esetben) 0 és 1, vagy egyébként egy nagyobb számosságú ábécének megfelelőek. A keresztezés általában két egyszeres kromoszómájú, haploid szülő génkészlete között zajlik. A mutáció művelete véletlenszerűen változtatja meg a gének értékét: negálja az adott pozíciójú bitet (gént), vagy nagyobb ábécéjű génváltozatok esetén más allélra módosítja azt.

A genetikus algoritmusoknál leggyakrabban haploid egyedeket használnak, általában egy kromoszómával. Az egyed genotípusa a kromoszómájában (például bitszring) található gének (például bitek) adott konfigurációja. [ÁGyHV3]

Biológiai fogalom	**Genetikus algoritmusbeli megfelelő**
Kromoszóma	Sztring
Gén	Jellemző
Génváltozat (allél)	A jellemző értéke
Gén helye (locus)	Sztring pozíció
Genotípus	Struktúra
Fenotípus	Egy megoldás

2. ábra. A biológiai és a genetikus algoritmusbeli terminológia összevetése

5. KANONIKUS GENETIKUS ALGORITMUS

A genetikus algoritmusnak nincs szigorú definíciója, mely mindenki számára elfogadott lenne. A legtöbb módszer melyet genetikus algoritmusnak neveznek, tartalmazza az alábbi elemeket:

- kromoszómákból álló populáció;
- jósági mérték alapú kiválasztás;
- keresztezés új utódok létrehozására;
- véletlenszerű mutáció az új egyedeken.

Az alábbiakban a genetikus algoritmus egy kiindulási változatát mutatom be, melyet szokás egyszerű genetikus algoritmusnak is nevezni. Ez a legegyszerűbb változat, ugyanakkor a genetikus algoritmusok minden lényeges eleme megtalálható benne.

A genetikus algoritmusok populációjában található kromoszómák gyakran bitsztringekből állnak.

Minden lehetséges génpozicióban két lehetséges génváltozat fordulhat elő: a 0 és az 1.

Mindegyik kromoszóma a keresési tér egy-egy pontját reprezentálja. A genetikus algoritmus kromoszómák nemzedékeit hozza létre, melyek rendre váltják egymást az algoritmus iterációiban. A genetikus algoritmus működéséhez szükség van egy **jósági függvényre**, amely az egyes kromoszómákhoz az adott populációban egy jósági mértéket rendel. Ezek az értékek attól függenek, hogy a kromoszóma által képviselt megoldás mennyire kedvező.

A következőkben a Holland-féle kanonikus genetikus algoritmust fogom ismertetni. Itt két fontos összetevőt kell kiemelni:

- a probléma kódolása, más szóval a reprezentáció
- a kiértékelő függvény

A genetikus algoritmus e két összetevője problémafüggő. [ÁGyHV4]

5.1 Reprezentáció

Ez a genetikus algoritmusok egyik sarokköve. Mivel az új egyedek létrehozása a kromoszómán végzett műveletek alapján történik, nagymértékben a reprezentáción múlik, hogy a szülőegyedek - mint megoldások-egyes jellemzőit az utódegyedek - mint új megoldások hogyan fogják örökölni. Ettől függ, hogy működni fog-e a kiválasztódás mechanizmusa.

A kódolásra sokféle lehetőség nyílik. A klasszikus megoldás az, ha a kromoszómák úgy néznek ki mint a természetben, azaz gének sorozatából állnak. Ennél az algoritmusnál a kromoszómák rögzített hosszúságú bináris sztringekből tevődnek össze. Az egyes megoldásokat, egyedeket a keresési térben egy egyszerű bitsorozat reprezentálja:

$$x_i\ ,\ i: \{0,1,2,..L\text{-}1\},\ x_i\ \hat{I}\ [0,1]$$

A sorozatok hossza (L) rögzített az egész algoritmusra nézve.

További fontos kérdés a tulajdonképpeni kódolás módszere azaz, hogy az adott probléma egyes megoldásait milyen leképezés segítségével feleltetjük meg a 2^L darab lehetséges bináris sorozatnak. Ha nincs a megoldáselemek között mélyebb összefüggés, strukturálódás, kézenfekvő, hogy az egyes megoldásokat egy-egy sorszámmal *(j)* megjelölve felsoroljuk. Ekkor alkalmazható közvetlenül a hagyományos bináris kódolás:

$$j = \sum_{i=0}^{L-1} 2^i x_i$$

Emellett még egy másik kódolási forma a Gray-kód használata. Ennek előnye, hogy a szomszédos elemek kódjában mindig csak egy bit tér el.

Másrészről lehetséges, hogy az egyes elemek, bizonyos módon csoportosíthatók, rendszerbe foglalhatók, vagy létezik már kiindulásként valamilyen kapcsolódó információ, aminek alapján egyértelmű megfeleltetéssel hozzárendelhető minden elemhez egy bináris sorozat. [ÁGyHV5]

5.2 Kiértékelési és jósági függvény

A genetikus algoritmusok kidolgozásakor a másik fontos feladat a **kiértékelő függvény** *(evaluation function)*, vagy **célfüggvény** *(objective function)*, illetve az ebből származtatott **jósági függvény** *(fitness function*) kialakítása. Ezek adják meg a keresési elemek jóságát, és így a genetikus algoritmusnak azt a vezérlő mértékét, amelyeknek alapján az algoritmus az egyes elemek között különbséget tud tenni. A kiértékelési függvény nem jut közvetlenül érvényre, hanem a skálázási eljárás után kapott fitnesz értékek használhatók fel a genetikus szelekció műveletében.

A fitneszfüggvény, tehát az egyes egyedek által képviselt megoldások jóságát jellemzi. A jósági függvény kialakítása során fontos szempont, hogy az minél jobban a megoldásegyedeknek a probléma szerinti jóságát, illetve az elemek közötti különbségeket tükrözze. A fitneszfüggvény meghatározása két részből áll:

- A célfüggvényt számítjuk ki. Ez közvetlenül az adott kromoszómákból adódik. Lehetséges, hogy az egyedet csak viszonylag durva lépték jellemzi.
- **Skálázás** *(scaling)*. Egy monoton függvény alkalmazása, melynek következtében az utána kapott értékek már arányosan jól mutatják az egyes egyedek jóságát.

A fitneszfüggvénnyel szembeni legfontosabb elvárás az, hogy egy jobb megoldáshoz nagyobb értéket rendeljen, másrészt két egyed jóságának különbsége legyen arányos a fitneszértékeik arányával.

A célfüggvény kialakításánál lényeges szempont, hogy minél kisebb számítási bonyolultságú legyen. Az algoritmus során sokszor kell alkalmazni, így hozzájárul a futási időhöz.

A célfüggvény kialakításánál a következő módszereket szokták alkalmazni:

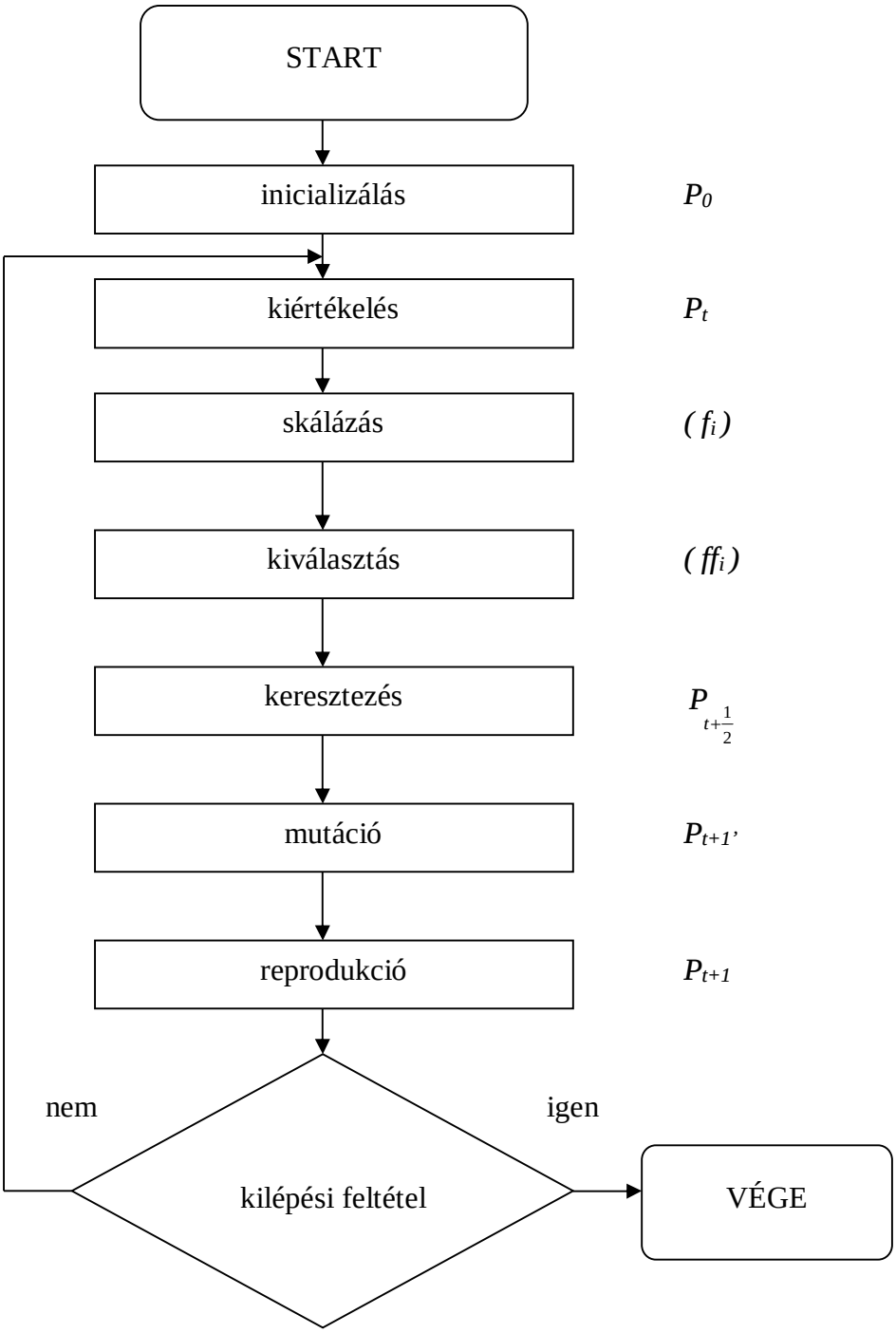
- Az elfogadható megoldásoknál azok jóságát kell megadni, az elfogadhatatlanoknál azt a távolságot célszerű figyelembe venni, amennyire a legközelebbi elfogadható megoldás található a keresési térben.
- **Büntetőfüggvény** alkalmazása. Egy adott megoldás negatívumait adott súlyozással bünteti, majd a fitneszértéket ebből az összbüntetésből számolja, valamilyen szigorúan monoton csökkenő függvény szerint. A büntetés súlyozásánál az elfogadhatatlan hibákat kiugróan nagy súllyal kell figyelembe venni.
- Sokszor hasznos az approximációs (közelítő) függvény használata. Ilyen eset, amikor a problémából kevés információ ismert, és az egyes tényezőket csak közelítve lehet figyelembe venni. Más esetben ismert a pontos fitneszfüggvény, de annak számítása túl sok időt venne igénybe, ezért célszerű valamilyen közelítő függvény használata. Ilyenkor ugyan pontatlanabb lesz a megoldások megítélése, de adott idő alatt, lényegesen több egyedet lehet vizsgálni. [ÁGyHV6]

5.3 A kanonikus algoritmus működése

Itt a genetikus algoritmusok egy általános menetét mutatom be, mely szinte mindegyik algoritmus alapjául szolgál. A különféle sémák csak a sémán belüli részleteket érintik. A konkrét elemekre a kanonikus algoritmus szerinti változatot mutatom be.

Tegyük fel, hogy van egy jól definiált probléma, egy kromoszóma reprezentáció a lehetséges megoldásokra, és egy célfüggvény, amely minden megoldáshoz hozzárendel egy eredményességi értéket.

A genetikus algoritmus működésének lépései. [ÁGyHV7]



3. ábra. A genetikus algoritmus működésének lépései

A következőkben áttekintésre kerülnek az egyes lépések.

5.4 Inicializálás

Az inicializálás létrehozza a kezdő populációt, amelyben meg kell határozni az egyedek kromoszómáinak az értékeit. A genetikus algoritmusok a gyenge módszerekhez tartoznak, tehát az adott problémáról szinte semmilyen apriori ismerettel nem rendelkeznek. Így az algoritmusok a kromoszómák génjeit egy lehetséges értékkészletből teljesen véletlenszerűen választják egyenletes eloszlás szerint.

A kanonikus algoritmusnál L hosszúságú bitsztringeket használunk kromoszómákként. Minden bitet 50-50 % valószínűséggel állítunk be 0-ra vagy 1-re. Így minden kromoszómaváltozat (bitszringminta) $\frac{1}{2^L}$ valószínűséggel adódhat egy konkrét egyed esetén.

A populáció alapvető jellemzője a benne foglalt egyedek száma, azaz a populáció mérete, melyet leggyakrabban állandónak veszünk, és ez minden generációban változatlan marad.

A következőkben a példaképpen vett kanonikus algoritmusban az egyedek száma: 6, a bitsztrig hossza: 8 (*N=6 és L=8)* A kialakított bitsztringek előállítása véletlenszerű.

A genetikus algoritmus lépéseinek vizsgálatához az alábbi populációt használom: [ÁGyHV8]

	1.	2.	3.	4.	5.	6.	7.	8.
1.	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**1**
2.	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**1**
3.	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**
4.	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**
5.	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**
6.	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0**

4. ábra. A vizsgált populáció

5.5 Kiértékelés, skálázás

A kiválasztás a populációban található egyedek közül választ ki egyeseket aszerint, hogy milyenek a fitneszértékeik. Ez az eljárás a természetes kiválasztáshoz hasonlóan történik, ahol az egyes egyedek elsődlegesen a kromoszómáikban kódolt tulajdonságaik alapján bizonyulnak életképesnek vagy kevésbé életképesnek, ami meghatározza a reprodukcióra való esélyüket.

Az aktuális pozícióban lévő egyedek célfüggvényértékeinek *(f_i)* kiszámítására, majd ebből a skálázási módszerrel fitneszértékük meghatározására a szelekciós műveleteket megelőzően van szükség. A fenti populációban minden egyedet egy 8 bites érték jellemez, a hozzátartozó decimális szám (1 bájt) segítségével könnyen előállítható a célfüggvény és, az egyedek fitneszértékei.

A célfüggvényt a feladat határozza meg. Az itt leírt genetikus algoritmusnál a célfüggvénynek az egyedekben szereplő bináris szám négyzetét választottam. Tehát a cél egy olyan populáció kialakítása melynél az egyedekben tárolt bitfüzérek decimális értékeinek négyzete maximális. A várható eredmény az lesz, hogy a genetikus algoritmus lefutása után az egyes egyedek az **11111111** bitfüzért tartalmazzák.

Táblázat a példapopuláció célfüggvényértékenek és fitneszértékeinek számításához:

	1.	2.	3.	4.	5.	6.	7.	8.	x_i	$f_i=x^2$	ff_i	Szelekciós valószínűség (p_i)
1.	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	157	24649	0,97	16,2%
2.	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	81	6561	0,26	4,3%
3.	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	138	19044	0,75	12,5%
4.	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	229	52441	2,07	34,5%
5.	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	44	1936	0,08	1,3%
6.	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	218	47524	1,87	31,2%
									f_i-k összege	152155	6,00	p_i–k összege: 100%
									f_i-k átlaga:	25359		

5. ábra. A vizsgált populáció bináris értékei, célfüggvényértékei, fitneszértékei, szelekciós valószínűségei

A fenti populációnál:

$$f_i = x_i^2 \quad \text{(célfüggvényértékek, az adott egyed kiértékelése)}$$

$$\bar{f} = \frac{\sum_{i=1}^{N} f_i}{N} = \frac{\sum_{i=1}^{N} x_i^2}{N} \quad \text{(célfüggvényértékek átlaga)}$$

Ekkor a populáció egyedeinek fitneszfüggvényértékeit, amely az egyedek jóságát adja , a következőképpen adhatjuk meg (skálázás):

$$ff_i = \frac{f_i}{\bar{f}}$$

Itt fennáll a következő kapcsolat:

$$\sum_{i=1}^{N} ff_i = \sum_{i=1}^{N} \frac{f_i}{\bar{f}} = \frac{\sum_{i=1}^{N} f_i}{\bar{f}} = \frac{\sum_{i=1}^{N} f_i}{\sum_{i=1}^{N} \frac{f_i}{N}} = \frac{\sum_{i=1}^{N} f_i}{\frac{\sum_{i=1}^{N} f_i}{N}} = N$$

Tehát ennél a kanonikus algoritmusnál a fitneszértékek összege egyenlő a populáció elemszámával.

A 5. ábra táblázatában megadott szelekciós valószínűségeket a következőképpen kapjuk:

$$p_i = \frac{ff_i}{\sum_{i=1}^{N} ff_i} = \frac{ff_i}{N}$$

A szelekciós valószínűségek (p_i) útmutatást adnak az algoritmusnak arra, hogy a szelekció műveleténél milyen eséllyel válassza ki az egyes szülőket.

Itt nyilván teljesül:

$$\sum_{i=1}^{N} p_i = \frac{\sum_{i=1}^{N} ff_i}{N} = \frac{N}{N} = 1$$

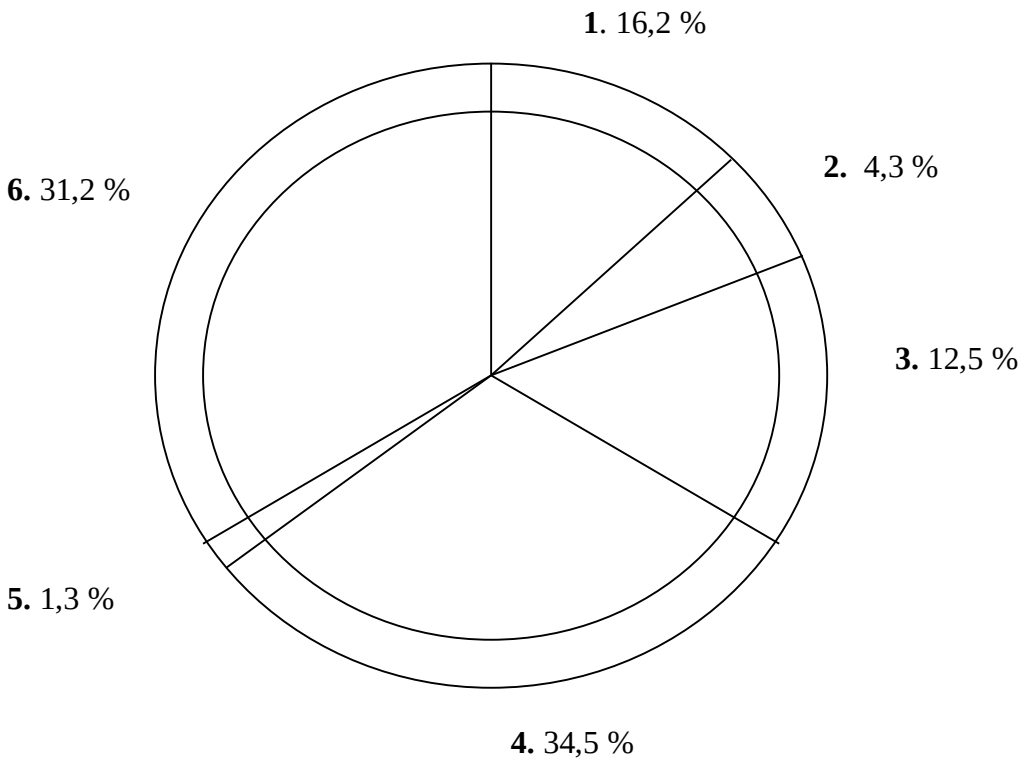
Ez egyszerű eltolás nélküli lineáris skálázás, de az együttható nem konstans, hanem az aktuális populációtól függ. Így az adott egyed skálázásakor szerephez jutnak a többi egyed kiértékelésénél kapott eredmények is. A fitneszérték határozza meg az egyes egyedek reprodukciós esélyeit. A kiválasztás során ez az érték ad útmutatást az algoritmusnak, hogy mely populáción belüli kromoszómák közül válasszon.

5.6 Szelekciók

A genetikus algoritmus a szelekció művelete során az aktuális populációból egyedeket (szülő kromoszómapár) emel ki nemdeterminisztikus módszerrel a későbbi reprodukció céljára. Az egyes egyedek kiválasztásának valószínűsége a fitneszértéküktől függ, annak monoton növekvő függvénye.

5.6.1 Rulett kerék vagy fitneszarányos kiválasztás

A szülőgenerációból egyszerre egy egyedet választunk ki, úgy hogy minden populációbeli egyed kiválasztási valószínűsége a jósági mértékével lesz arányos, mint ahogy az 5. ábrán látható. Az 5. ábrán feltüntetett % értékek a szelekciós valószínűségek. [ÁGyH9]



6. ábra. Rulettkerék szelekció

Ugyanazt a kromoszómát többször is ki lehet választani, az egyes kiválasztási műveletek függetlenek egymástól. Ez úgyis elképzelhető,, hogy 1 és N között véletlenszerűen választunk ki számokat, mert mint az előzőekben beláttuk:

$$N = \sum_{i=1}^{N} ff_i$$

Miközben minden egyedhez a jósági értékének megfelelő számot rendelünk hozzá. A hozzárendelés egyértelműen megmutatja, hogy a módszer segítségével mely egyedeket választottuk ki. A módszer előnye, hogy igen egyszerű implementálni, és sokszor megfelelő eredményt is nyújt. Vannak olyan esetek, amikor más megoldással jobb eredményt lehet elérni.

5.6.2 Pár-verseny szelekció

Ez talán a legegyszerűbb módszer sok alkalmazásban lehet vele találkozni. Kiválasztunk a populációból két megoldást véletlenszerűen. Vizsgáljuk a két egyed rátermettségét, a szelekció által kiválasztott elem a kettő közül a rátermettebb. Ez a technika úgy általánosítható, hogy nem kettő hanem több elem győztesét választjuk ki. [SWÁ3]

5.6.3 Rangsorolás

A rátermettség-arányos szelekcióhoz hasonló. A rátermettség értékét közvetlen módon használó módszerek problémája, hogy túlságosan érzékenyek a rátermettség eloszlásra a populációban. Ha például egy megoldás magas rátermettségi értékkel rendelkezik a többihez képest, akkor szinte mindig ő lesz a kiválasztott szülő, aminek káros hatásai vannak, hiszen a keresés a változatosság gyors csökkenése miatt „beragadhat”; az aktuális legjobb megoldást tovább javítani egyre nehezebb lesz. Ezt a problémát úgy oldhatjuk meg, hogy a populáció elemeit rátermettség szerint sorba rendezzük, és a rátermettség helyett valamilyen „szép”, a rendezésben elfoglalt hely szerint egyenletesen növekvő függvényt használunk. Gyakori a **lineáris rangsorolás**, amikor ez a függvény megközelítőleg egyenes.

A szelekciós műveletet annyiszor hajtja végre az algoritmus, ahány eleme van a populációnak *(N)*. A kiválasztott egyedek egy új átmeneti populációt alkotnak, melyek páronként rendeződnek, a kiválasztásuk sorrendje szerint (felteszzük, hogy *N* páros). Ezzel a módszerrel biztosítani tudjuk, hogy a populáció mérete ne változzék.

Az átmeneti populáció kialakítása után következnek a genetikus műveletek, melyek segítségével kialakul az új generáció populációja. [FI2]

5.6.4 Rátermettség-arányos szelekció

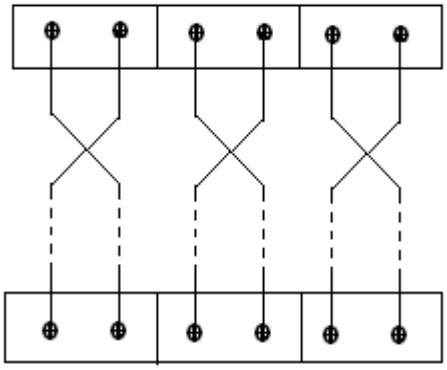
A módszer valószínűségi mintavételt használ. Egy kiválasztás valószínűsége annál nagyobb, minél nagyobb a rátermettsége a populáció rátermettségi átlagához képest. A szelekció matematikai értelemben egy mintavétel a populációból. A populáció minden *e* elemére a kiválasztás valószínűségét megadhatjuk a

$$P(e) = \frac{f(e)}{n \cdot f(pop)}$$ képlettel, ahol

f(e) a rátermettség értéke, *n* a populáció elemszáma, és *f(pop)* a populáció tagjainak átlagos rátermettsége. Itt feltettük, hogy a rátermettség pozitív. [SWÁ4]

5.7 Genetikus műveletek

Az átmeneti populációban található, reprodukcióra kiválasztott szülőpárokból kiindulva jön létre az egyedek utódgenerációja.



7.ábra. Az átmeneti populációból az utódok létrehozása keresztezéssel, majd mutációval

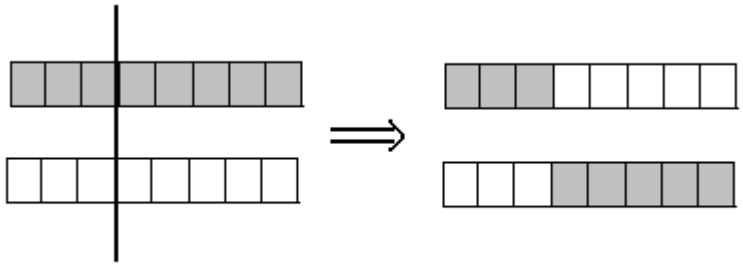
Az utódgeneráció létrehozásához különféle genetikus operátorokat használ fel a genetikus algoritmus. Két leggyakrabban alkalmazott művelet a **keresztezés** és a **mutáció**.

Először a keresztezés kerül végrehajtásra, amely egy szülőpár két tagjából, azok kombinálásával hoz létre két új utódegyedet. Majd az így létrejött utódokon alkalmazza az algoritmus a mutáció műveletét, amely egyszerre egy egyeden működik, véletlenszerűen megváltoztatva annak kromoszómáját. Az így létrejött egyedekből alakul ki a következő nemzedék populációja.

5.8 Keresztezés (rekombináció)

Ez a művelet alapvetően két szülőből hoz létre egy vagy két utódot. Az egyes szülőpárok esetén az algoritmus p_i valószínűséggel fogja őket keresztezni. Amennyiben nem kerül sor a műveletre a két utód a két szülő módosítás nélküli másolata lesz. A keresztezés az élővilágból ellesett műveletek utánzása. Az utódgének a két szülő genetikus állományának a keveredései lesznek, a szülőkromoszómákban a megfelelő pozícióban lévők közül veszik fel valamelyik értéket. Ezzel a művelettel tehát elsősorban a keresési tér már felfedezett területeit hasznosítja az algoritmus.
Bitsztringeknél több változat használatos rekombinációra, a következőkben ezt fogom ismertetni.

5.8.1 Egypontos keresztezés

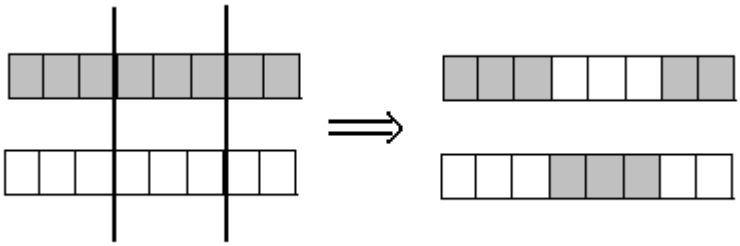


8. ábra. Az egypontos keresztezés

Ez az eljárás két szülőkromoszómát vesz alapul, mindkettőt ugyanabban a véletlenszerűen megválasztott pozícióban elvágja, majd ezeket a két szülő között felcserélve összefűzi ismét. A pozíció megválasztásakor az egyes lehetőségek valószínűsége egyenletes eloszlást mutat. Ez a művelet az élővilágban megfigyelhető két egyszeres kromoszóma (haploid) egyedei között lejátszódó rekombinációt utánozza.

5.8.2 Kétpontos keresztezés

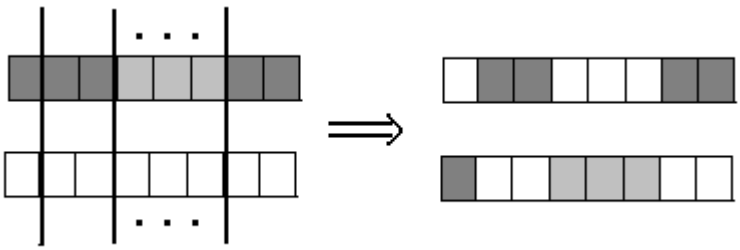
Itt két pontot választunk ki véletlenszerűen elvágásra, majd ezeket a részeket váltakozva ragasztjuk össze. A helyek kiválasztása véletlenszerű, mindegyiknek ugyanannyi a valószínűsége.



9. ábra. A kétpontos keresztezés

5.8.3 N-pontos keresztezés

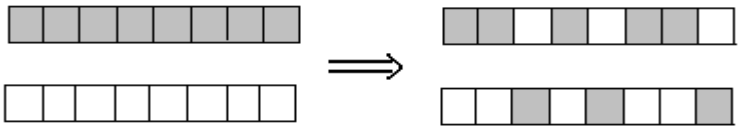
Itt kettőnél több pontot választunk ki véletlenszerűen elvágásra, majd ezeket a részeket váltakozva ragasztjuk össze. A helyek kiválasztása véletlenszerű, mindegyiknek ugyanannyi a valószínűsége.



10. ábra. Az N pontos keresztezés

5.8.4 Uniform keresztezés

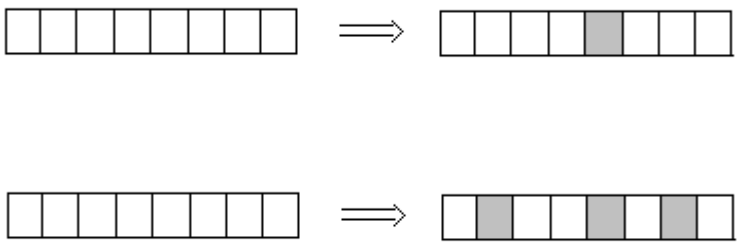
Szintén alkalmazható bináris sztringeknél. A gyermekkromoszómák génjei az azonos pozícióban levő szülőgének közül kerülnek ki adott valószínűséggel az egyik vagy a másik szülőtől. Ez a valószínűség általában 0,5.



11. ábra. Az uniform keresztezés

5.9 MUTÁCIÓK

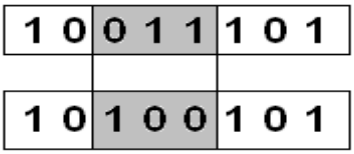
A mutáció művelete egy kromoszómát kis mértékben módosít. Ennek leggyakoribb módja, hogy a gén közül egyet véletlenszerűen kiválaszt, majd ennek értékét szintén egy véletlenszerű értékkel kicseréli.



12. ábra. A mutáció két lehetősége

A mutáció elsődleges funkciója a keresési tér újabb területeinek felkutatása. A művelet a létrejött két utódot egymástól függetlenül mutálja. Lehet ezt egy véletlenszerűen választott pozícióban, vagy minden pozícióban egymástól függetlenül elvégezni. A művelet után az algoritmus az utódegyedeket elhelyezi az új populációban. A mutáció a sztring mindegyik pozíciójában megtörténhet egy megadott valószínűséggel, ami általában kicsi (például 0,001).

Gyakran alkalmazzák több gén mutálására az **inverzió** műveletét. Ez a művelet véletlenszerűen kiválasztja a kromoszóma egy szakaszát, majd azt megfordítva adja vissza.



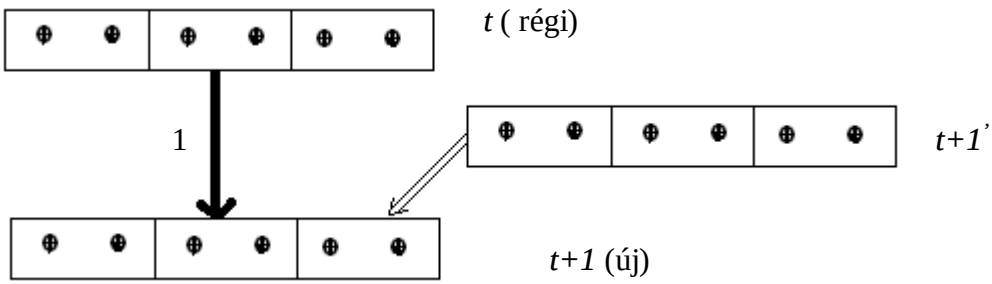
13. ábra. Az inverzió művelete

5.10 REPRODUKCIÓ

Az új generáció alapját az előállított utódpopuláció adja. Az új populációt a reprodukció lépése hozza létre a létrejött új egyedekből és a régi generációból. Erre is többféle változat terjedt el. Az alapul szolgáló kanonikus algoritmusnál a lehető legegyszerűbbet érdemes alkalmazni.

Az **egyszerű vagy generációs reprodukció** teljesen lecseréli a régi populációt. Tehát az összes régi egyed (szülő) elvész, csak az utódok maradnak fenn.

Ehhez kiegészítésként szokták alkalmazni az **elitizmust**, amely kijátssza az előbbi szabályt, a szülőpopuláció legjobb egyedét meghagyja, és az új generáció egyik (véletlenszerűen választott, vagy a legrosszabb) egyede helyébe lép. Ezt szemlélteti a 14. ábra.



14. ábra. Az elitizmus

5.11 KILÉPÉSI FELTÉTEL

Az algoritmus leállását szabályozza a kilépési függvény, amely a kilépésről különféle feltételek, illetve azok kombinációja vagy kapcsolata alapján dönthet.

- Adott számú generáció után véget ér az algoritmus futtatása. Ebben az esetben független a kilépés az aktuális populáció és egyedeinek jóságától és konvergenciájától. A módszer alkalmazása esetén mindenképpen szükséges valamilyen előismeret az adott problémáról, illetve az adott genetikus algoritmus működéséről.
- A leállást szabályozhatja az aktuális generáció állapota. Például leállhat az algoritmus, ha a legjobb egyed fitneszértékét összehasonlítva a populáció átlagos fitneszével, az arány elér egy előre megadott értéket, azaz a generációban található legalább egy nagyon jó megoldás. Másik lehetőség a kilépésre, ha az egyes egyedek szórása - távolságfüggvényük alapján - egy adott érték alá esik. Ez azt jelenti, hogy a keresési térben az egész populáció egy megadott területre koncentrálódik.
- Az algoritmus leállása függhet a populáció vagy a legjobb egyedének konvergenciájától is. A konvergenciát többféleképpen is lehet definiálni. A bináris kromoszómák esetén egy lehetőség: Egy gén konvergál, ha az egész populációban az adott gén értéke például 98 %-ban azonos. Egy populáció

konvergál, ha minden gén konvergál. Egy másik lehetőség, ha a jelenlegi populáció átlagos, maximális fitneszértékét valamely régebbi populációéval hasonlítja össze az algoritmus.

6. A KANONIKUS ALGORITMUS GYAKORLATI MEGVALÓSÍTÁSA

A kanonikus genetikus algoritmust, amit az előzőekben ismertettem Turbo Pascalban valósítottam meg. A populáció elemszáma 256. A kromoszómák 8 gént (bitet) tartalmaznak. *(N=256 L=8)*

6.1 Kezdeti populáció létrehozása:

```
{- Kezdeti populáció véletlenszerű létrehozása-}
N:=256;                                  {a populáció elemszáma}
L:=8;                                    {a bitsztring hossza}
randomize; { a véletlengenerátor bekapcsolása}
 {a  populációt létrehozó ciklus}
for i:=1 to N do begin
xi:='';      { -xi üres sztring-}
    {a kromószómákat létrehozó ciklus}
   for j:=1 to L do begin
     g:=random(2);              {g egész típusú. Így értéke 0 vagy 1 lehet}
     if g=0 then xi:=xi+chr(48);    {chr(48) a „0”  karakter}
     if g=1 then xi:=xi+chr(49);    {chr(49) a „1”  karakter)}
     { az xi-hez való hozzáadás,így létrejön az L hosszúságú bitsztring}
   end;
    kro[i]:=xi;       {az xi kromoszóma tárolása a kro vektorban}
 end;
```

Az 1-es és a 0-ás bit 50-50 %-os valószínűséggel generálódik. A generálásnál a karakterek ASCII kódjait használtam. A 48-as kód a 0, a 49-es az 1. A létrehozott 8 hosszúságú 256 darab bitsztringet a ***kro (kromoszóma)*** vektorban tároltam.

Így megkaptam a kezdeti populációt, amely teljesen véletlen egyedekből áll. A 15. ábrán láthatók a generált bitszringek. Ezen a populáción kell alkalmazni a genetikus algoritmust.

```
11110111 10010000 01111001 01110010 10011011 11011101 10010101 01001110
11111100 00010001 10000101 01011101 00110010 10011101 10000111 10011000
00001011 10011111 10010000 11001000 10001011 00110100 10011010 01110010
10000111 11110010 11001110 10001000 10001001 00011011 00111100 01010001
01010000 00000000 01110101 01100011 01000010 00111100 01101110 10010010
11011001 01001000 01101111 11111111 10001010 11001101 10001001 00110110
11101111 01110110 01001111 01000011 11001001 11000111 00011100 10000010
00110000 01111100 10111000 00001101 11100111 00000111 10101010 00000111
10001110 00100110 11100100 10000100 10000001 00001000 10000010 01010011
10101110 11011110 10001010 11011010 11111110 11011101 10001100 11011111
10001100 00010110 01111010 10010111 01101110 11011100 11111000 11001100
00010010 10110110 00000101 10110011 00101111 00111000 11101111 10001101
01000101 00001111 00010011 10010001 11000011 00010100 11111011 10110111
01010100 00110110 10111101 00100100 00110001 11000011 11011101 10001111
00101101 00000010 01101011 01011000 10110001 00000011 00000101 01100101
11001011 00111000 01101101 01110001 10110011 00011010 11011001 11011001
10111001 00100000 01111101 00111011 00000101 00010101 00001010 00100111
00000010 00011001 00110101 10010000 10010000 11101100 00001000 01110110
01101101 01010001 00001010 01000001 10001101 11010010 10001000 10110010
10100100 01101010 00101110 00011101 10011011 10101111 00010011 11001100
10111001 11011111 11001000 11101101 11111010 11011110 10000010 10000000
11000100 01011111 01101101 11111101 00001101 10111000 00000001 10001111
01111111 00110110 11010000 01010010 10101001 10110010 11101101 00011101
10001101 01010001 10001101 10001001 10100101 10111101 10000111 10101100
11100101 00001110 10110010 00111100 10110000 01111101 01101111 01100001
00110000 00101110 01001011 11111111 10010110 10100001 11100110 01011011
11011010 01111010 11111101 10010011 11111101 10011010 01001110 10101111
10111001 01001101 01010110 01100111 01000100 01011100 10011010 01000011
01011100 10111111 00110110 00000111 00011011 10000011 00101010 00010100
00110010 11000111 01110011 10110001 11011010 00101100 01000000 00011111
11001101 11001011 00010011 01111100 10100010 01100100 00000101 10111011
10001011 10001010 10010010 11101100 00001001 00110001 01100100 00110111
```

15. ábra A generált bitsztringek-kromoszómák

6.2 Az algoritmus működéséhez szükséges számítások elvégzése

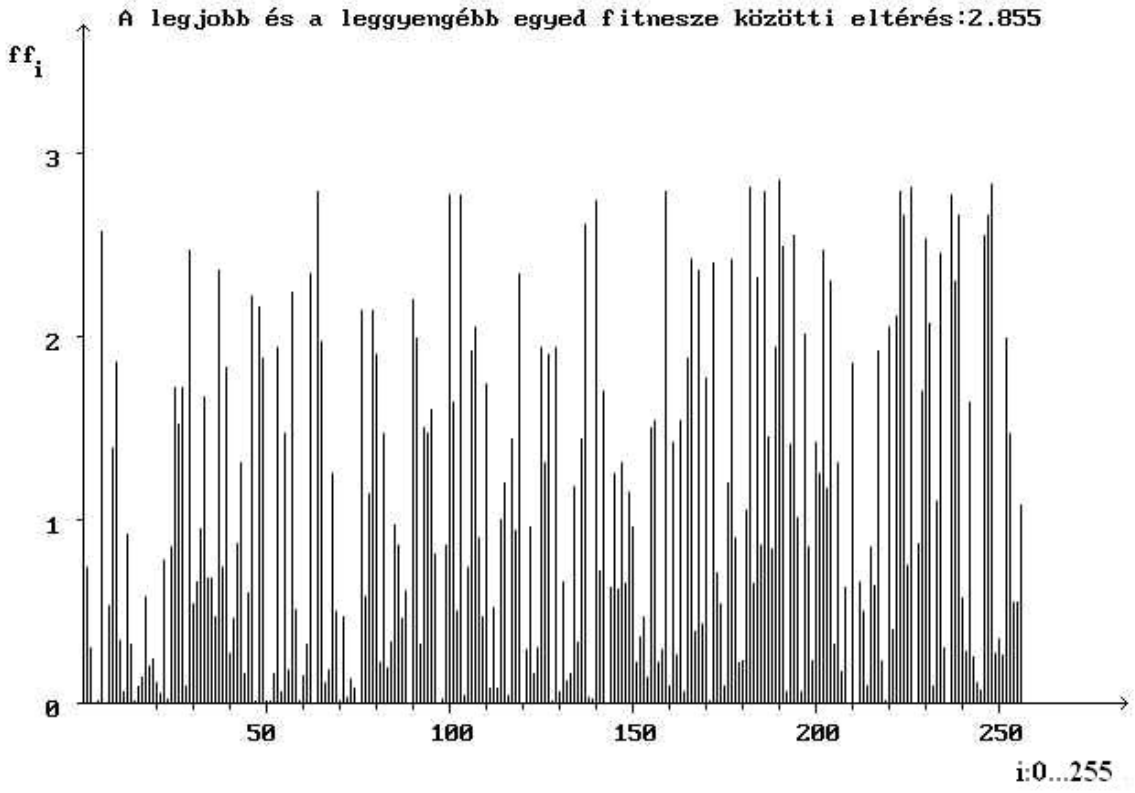
Itt négy fontos feladat van:

- A generált bitsztringeket (kromoszómák) decimális számmá kell alakítani.
- a célfüggvényértékek kiszámítása.
- fitneszfüggvényértékek meghatározása.
- fitneszfüggvény maximumának és minimumának kiszámítása. A maximum és minimum érték különbsége információt ad a populáció egészéről. Ez az előre megadott kilépési feltételhez szükséges.

```
Procedure szamol;
Begin
szum:=0;   { a célfüggvényértékek összege}
```

```
for i:=1 to N do begin
    for j:=1 to L do begin
        xi:=copy(kro[i],j,1); {xi  kromoszóma j-edik bitje ( sztring ) }
        val(xi,b[j],ko);      {a j-edik bit számértéke (0 vagy 1)}
      { a ko változó a sztring-szám konverziót ellenőrző változó}
end;
x[i]:=b[1]*128+b[2]*64+b[3]*32+b[4]*16+b[5]*8+b[6]*4+b[7]*2
+b[8];      { a kromoszóma decimális értéke, melyek az x vektorban vannak
tárolva}
    cfgv[i]:=x[i]*x[i];        {a célfüggvények értékei, melyek a cfgv
vektorban vannak tárolva}
    szum:=szum+cfgv[i];        {a célfüggvényértékek összege}
end;
{- SKÁLÁZÁS - az egyes egyedek fitneszeinek meghatározása --}
Procedure skala;
  Begin
    atl:=szum/n;               {a célfüggvényértékek átlaga}
    for i:=1 to N do begin
      fitn[i]:=cfgv[i]/atl;    {fitneszfüggvény értékei, melyek a fitn
vektorban vannak tárolva}
    end;
{------fitneszfüggvényértékeinek maximuma-minimuma--------- }
    min:=fitn[1];max:=fitn[1];
    for i:=2 to N do begin
       If fitn[i]>max then max:=fitn[i];
       If fitn[i]<min then min:=fitn[i];
    end;
    dh:=max-min;               {a minta terjedelme a kilépési feltételhez}
end;
```

Így megkaptam a kezdeti populációhoz tartozó fitneszfüggvényértékeket, melyet a 16. ábra szemléltet. Látható, hogy a populáció kialakítása miatt ezek az értékek is véletlenszerűek. A fenti eljárást nemcsak a kezdeti populáció létrehozásakor, hanem minden új populáció kialakítása után meg kell hívni.



16. ábra. A kezdeti populáció fitneszértékei

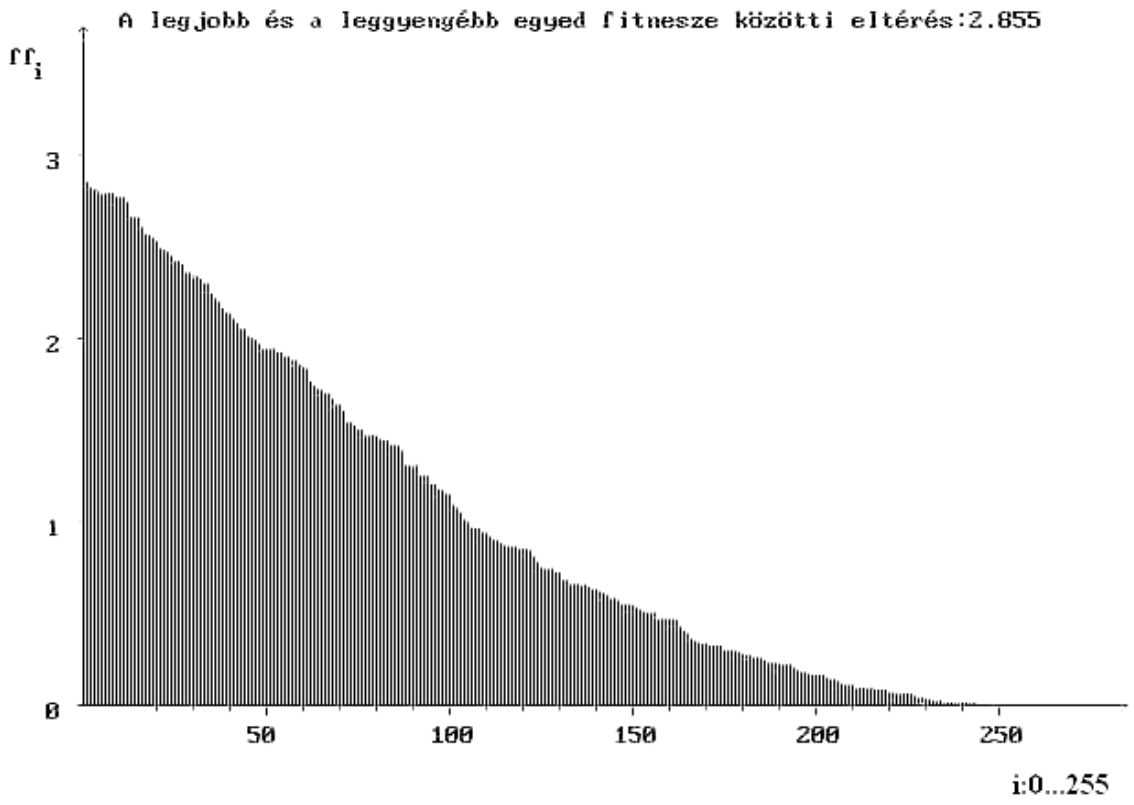
6.3 A kiválasztás művelete

A mintát maximum kiválasztásos rendezéssel csökkenő fitneszértékek szerinti sorba rendeztem. Itt ügyelni kell arra, az egyedek sorrendje is ennek megfelelő legyen. Ennek algoritmusa:

```
procedure rendez;
var cs2:real;    { fitneszértékek csereváltozója}
    ma:integer; {a rendező algoritmus működéséhez szükséges átmeneti változó}
    cs1:string; {a kromoszómák csereváltozója}
begin
for i:=1 to N-1 do begin
    ma:=i;
    for j:=i+1 to N do begin
        if fitn[mi]<fitn[j] then ma:=j;
    end;
    cs1:=kro[i];kro[i]:=kro[ma];kro[ma]:=cs1;
```

```
    cs2:=fitn[i];fitn[i]:=fitn[ma];fitn[ma]:=cs2;
end;
end;
```

Így megkaptam a kezdő populációt csökkenő fitneszértékek szerint rendezve.



17. ábra. A kezdő populáció fitneszértékei csökkenő sorrendben

A kiválasztásnál a következő szempontokat vettem figyelembe:

A csökkenő fitneszértékek alapján a szelekciónál a rangsorolás technikáját alkalmaztam. Az első és második legnagyobb fitneszértékű szülő egypontos keresztezéséből létrejön két utód, ezek a két leggyengébb utód helyébe lépnek. (az N. és a N-1. helyre) Majd a harmadik és negyedik fitneszértékű szülő keresztezéséből létrejött két utód az előzőleg létrejött két utód elé kerülnek (az N-3. és az N-4. helyre), és így tovább. (elitizmus)

6.4 Keresztezés

Az utódok kialakításánál az egypontos keresztezést alkalmaztam. A kiválasztott két szülőnél generáltam egy véletlenszerű pozíciót *(locus)* és erre a pontra alkalmaztam a műveletet.

Ennek az algoritmusa:

```
e:=1;v:=N; {eleje-vége}
{----------egypontos keresztezés--------}
While e<v do begin
   poz:=1+random(7);        {vágási pozíció poz Î [1;7]}
      {x1 az első szülő vágási pozíció előtti génfüzére}
   x1:=copy(kro[e],1,poz);
      {x2 az első szülő vágási pozíció utáni génfüzére}
   x2:=copy(kro[e],poz+1,8-poz);
      {y1 a második  szülő vágási pozíció előtti génfüzére}
   y1:=copy(kro[e+1],1,poz);
      {y2 a második  szülő vágási pozíció utáni génfüzére}
   y2:=copy(kro[e+1],poz+1,8-poz);
      {a létrejött első utód kromoszómája-bekerül a populációba}
   kro[v]:=x1+y2;  {keresztezés I.}
      {a létrejött második utód kromoszómája-bekerül a populációba}
   kro[v-1]:=y1+x2;  {keresztezés II.}
   e:=e+2;v:=v-2;
end;
```

A fenti algoritmus mindegyik szülőpárnál végrehajtódik, mert a pozíció 1-től 7-ig terjed. Az algoritmus a legjobb egyedeket megtartja, a létrejött utódok a leggyengébb fitneszértékűek helyébe lépnek.

6.5 Mutáció

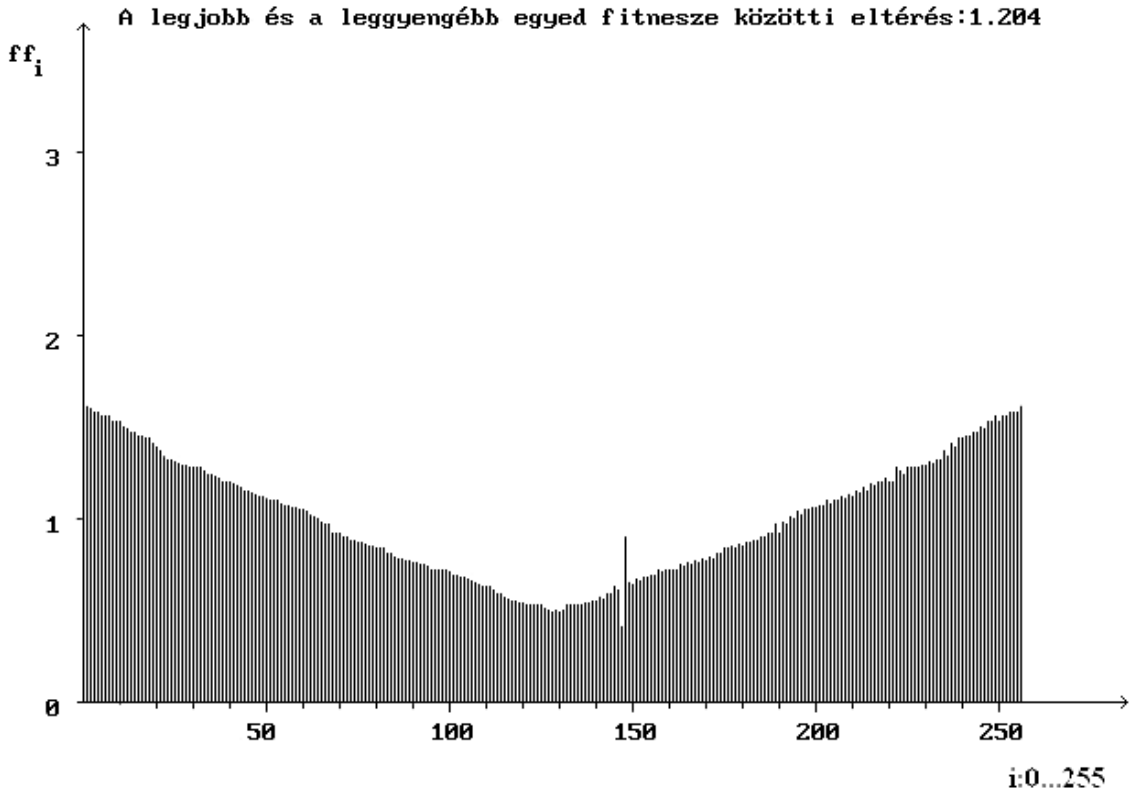
A létrejött két utód valamelyike 5% valószínűséggel mutálódik egy véletlenszerűen kiválasztott pozícióban. Ennek algoritmusa:

{I. utód mutációja}

```
p:=1+random(20);   {p típusa egész;  p Î [1;20] }
```

```
if p=5 then begin   {itt p Î [1;20],  p=5 teljesülési esélye 5%. Más érték is szerepelhetne}
        {pozícióválasztás 1-8-ig. egyszer végrehajtódik}
        poz:=1+random(8);     { poz Î [1;8] }
        {x1 a  kromoszóma pozíció előtti génfüzére}
        x1:=copy(kro[n],1,poz-1);
        {x0 a kromoszóma aktuális  pozícióban lévő génje}
        X0:=copy(kro[n],poz,1);
        {Az aktuális bit ellentettjére váltás}
        if x0='1' then x0:='0' else x0:='1';
        {x2 a kromoszóma pozíció utáni génfüzére}
        x2:=copy(kro[n],poz+1,8-poz);
        { új egyed létrehozása a részek összeillesztésével}
        kro[n]:=x1+x0+x2;
end;
{II. utód mutációja}
if p=10 then begin   {itt p Î [1;20],  p=10  teljesülési esélye 5%. Más érték is szerepelhetne}
        {pozícióválasztás 1-8-ig. egyszer végrehajtódik}
        poz:=1+random(8);    { poz Î [1;8] }
        {x1 a kromoszóma pozíció előtti génfüzére}
        x1:=copy(kro[n-1],1,poz-1);
        {x0 a kromószóma aktuális  pozícióban lévő génje}
        X0:=copy(kro[n-1],poz,1);
        {Az aktuális bit ellentettjére váltás}
        if x0='1' then x0:='0' else x0:='1';
        {x2 a  kromoszóma pozíció utáni génfüzére}
        x2:=copy(kro[n-1],poz+1,8-poz);
        { új egyed létrehozása a részek összeillesztésével}
        kro[n-1]:=x1+x0+x2;
end;
```

Ezzel megtörtént a reprodukció művelete,létrejött az új populáció:



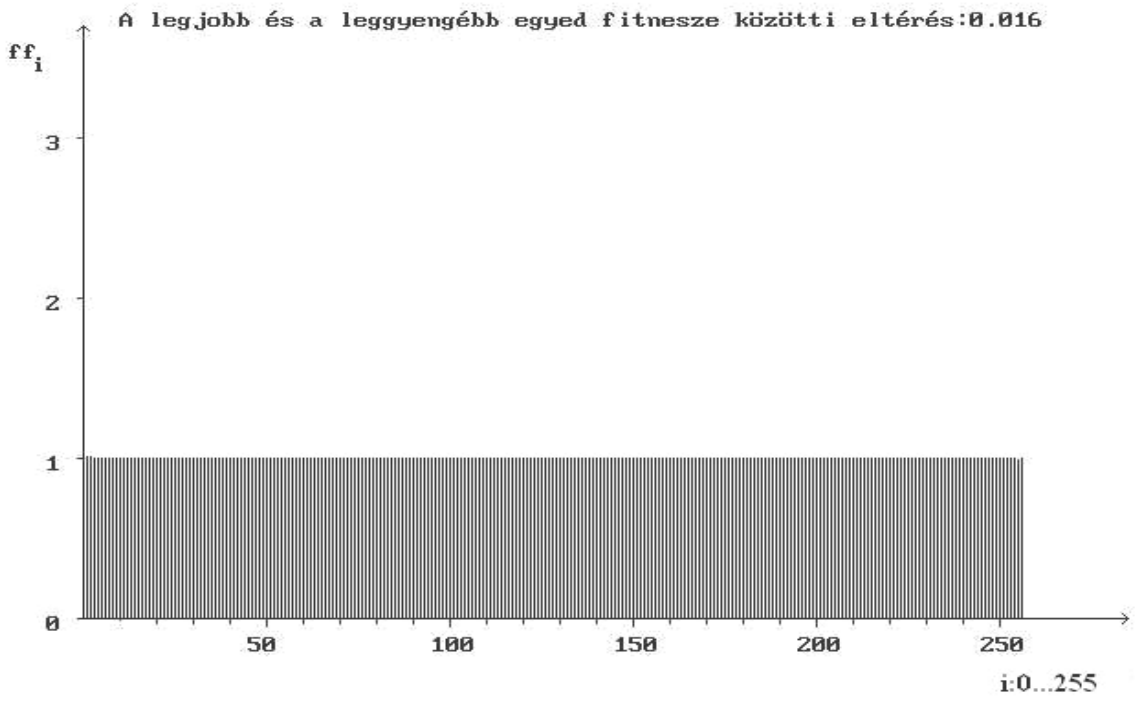
18. ábra. Az algoritmus működése során egy kialakult populáció fitneszértékei

Látható a fenti eloszlásban egy anomália, egy kiemelkedő csúcs, ennél az egyednél megtörtént a mutáció művelete.

6.6 Kilépési feltétel

Minden populációnál vizsgálatra került a populációnak a maximális és minimális fitneszértéke, valamint ennek a két értéknek a különbsége. *(dh=max-min).* Leállási feltételnek azt választottam, ha valamelyik új populációnál ez az érték kisebb mint 0,02.

```
while dh > 0.02 do begin   {a ciklus akkor áll le, ha dh<=0,02}
     kiválasztás
     keresztezés
     mutáció
     reprodukció
     az új populáció fitneszeinek a számítása
end;
```



19. ábra. Az ábrázolt populációnál teljesül a kilépési feltétel

A kilépési feltétel teljesülése esetén a populáció 98% eséllyel egyenlő fitneszű egyedekből áll. A kezdeti káoszból, kialakult egy szinte egységes populáció.

```
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111100 11111100 11111100 11111100 11111100 11111100
```

20. ábra. A genetikus algoritmus lefutása után kapott populáció bitszringjei

Az előbbiekben alkalmazott genetikus algoritmus eléggé gyors, köszönhetően az alkalmazott kiválasztásnak. Ha valamelyik egyednél végrehajtódik a mutáció, akkor a futási idő kismértékben megnő. Tapasztalatom az, hogy körülbelül a 10. populációnál teljesül a kilépési feltétel.

A teljes program forrása a mellékletben található meg. A program futása közben létrejött grafikus képernyőket *.pcx grafikus állományba menti el lehetővé téve a későbbi elemzéseket.

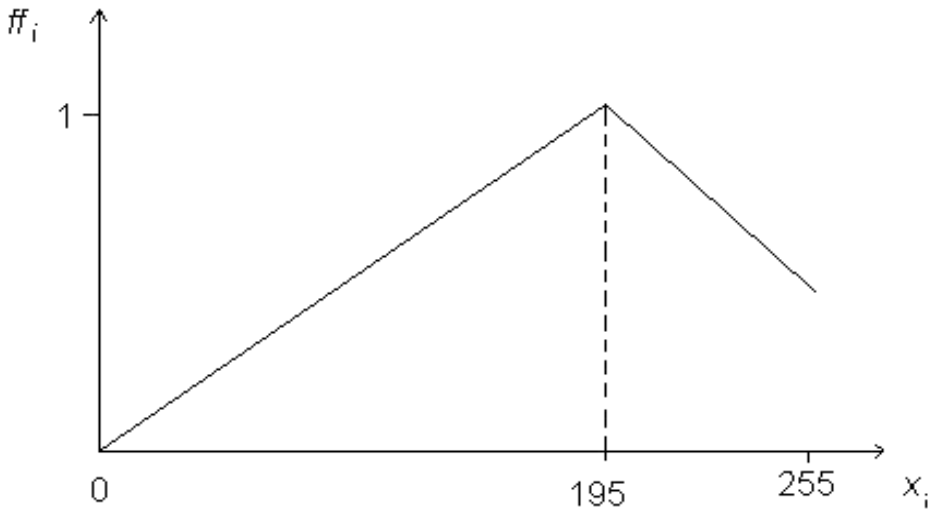
Más célfüggvényt is alkalmazhatunk. A következő példában, hasonlóan mint a biológiai fajtanemesítésnél egy véletlenszerű kezdeti populációból egy egységes populáció jön létre. Tegyük fel, hogy a legjobb egyedek kromoszómái az **11000011** génfüzérből állnak, ennek decimális értéke 195, ezt a kromoszómaállományt kell létrehozni a populáció egyedeiben.

A célfüggvény értéke $f_i=195$. A kromoszómákban lévő bitfüzér decimális értékei x_i

A fitneszfüggvényt a következőképpen választottam meg:

$$ff_i = 1 - \frac{|x_i - f_i|}{f_i}$$

Itt $x_i \in [0;255]$, és a függvény a maximumát az $x=195$ helyen veszi fel. A maximum értéke 1. Így azoknak az egyedeknek fitneszértékei kisebbek lesznek 1-nél, amelyek nem a kívánt bitfüzérből állnak. Így minden egyedhez a jóságának megfelelő értéket rendeltem hozzá. (Lineáris skálázás).



21. ábra. Az alkalmazott fitneszfüggvény.

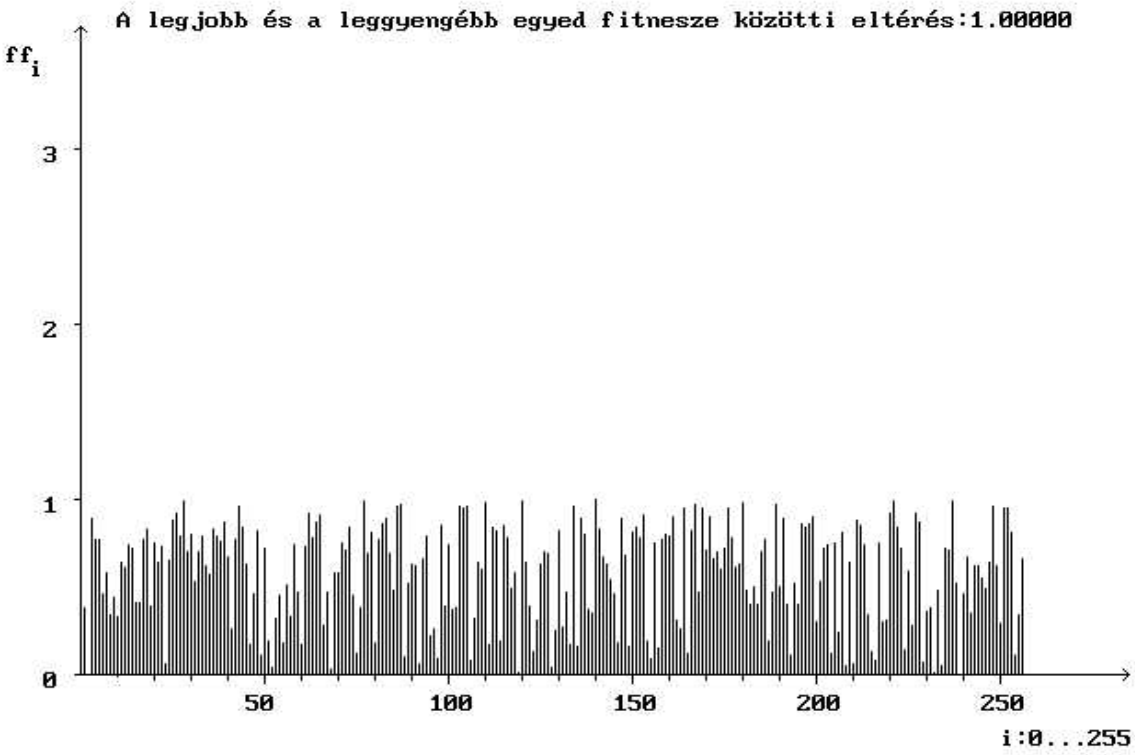
Hogy a genetikus algoritmus ebben az esetben is működjön, a 24. oldalon lévő algoritmusokat a következőképpen kell megváltoztatni:

```
Procedure szamol;
Begin
for i:=1 to N do begin
    for j:=1 to L do begin
        xi:=copy(kro[i],j,1); {xi  kromoszóma j-edik bitje ( sztring ) }
        val(xi,b[j],ko);      {a j-edik bit számértéke (0 vagy 1)}
      { a ko változó a sztring-szám konverziót ellenőrző változó}
end;
x[i]:=b[1]*128+b[2]*64+b[3]*32+b[4]*16+b[5]*8+b[6]*4+b[7]*2
+b[8];      { a kromoszómák decimális értéke, melyek az x vektorban vannak
tárolva}
    cfgv[i]:=195;          {a célfüggvények értékei, melyek a cfgv vektorban
vannak tárolva}
  end;
{- SKÁLÁZÁS - az egyes egyedek fitneszeinek meghatározása --}
Procedure skala;
  Begin
    for i:=1 to n do begin
      fitn[i]:=1-abs(cfgv[i]-x[i])/cfgv[i]; {fitnesz függvény
értékei, melyek a fitn vektorban vannak tárolva}
    end;
 end;
```

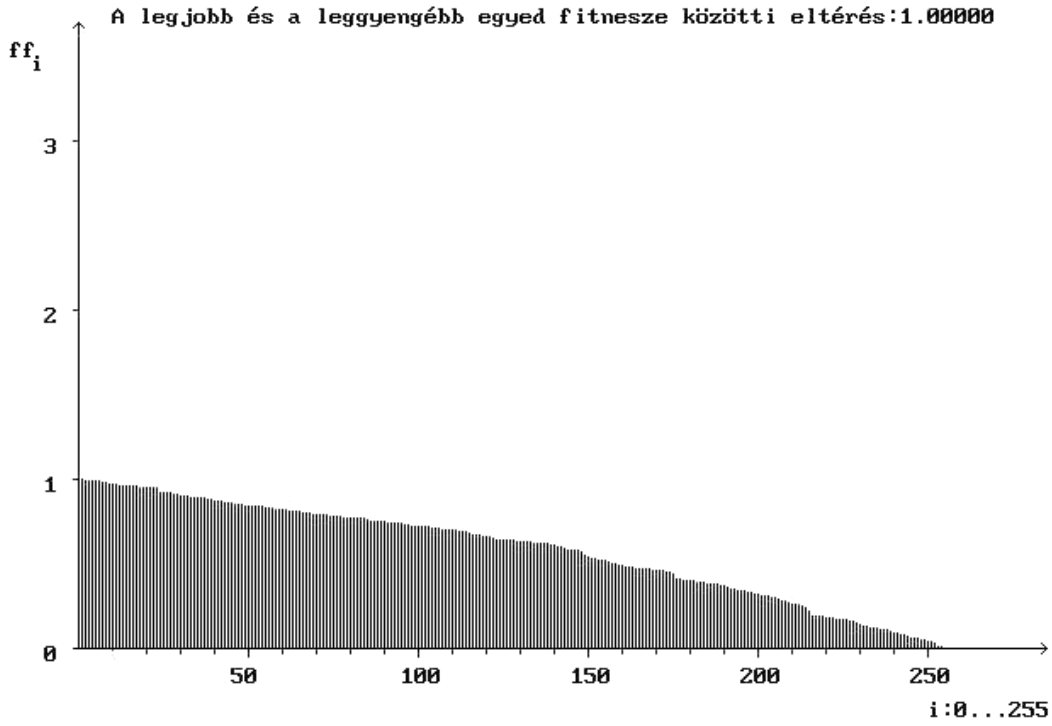
Tehát az itt alkalmazott függvény egy konstans függvény, A genetikus algoritmus lefutása után az egyedek kromoszómái a **11000011** bitfüzért tartalmazzák. Az algoritmus futásának eredményeit az alábbi ábrák tartalmazzák.

```
01001010 00000000 11011001 10010111 10010110 01011010 01110001 01000010
01010110 01000001 01111101 01110111 10010001 11111001 01001111 01001111
11101111 11100101 01001100 11110011 01111100 11111000 00001100 01111110
10101100 11010011 11101011 11000101 10001000 10011100 01101000 11111110
10011011 01111000 01110000 11100100 10011010 11110010 11011101 10000011
00110010 11110000 10111100 11100010 01111011 00100010 01011001 10100000
00010101 10001101 00100110 00001000 00111111 01011000 00100100 01100100
01000001 11110101 01011100 00100010 10001110 11010011 11101101 10101001
10110010 00110110 01011100 00000110 01110010 01110001 10010011 10001010
11100010 01010111 00010111 01001010 11000001 11111111 10011101 00100011
11110000 10101000 10101101 10000110 01011101 11001011 11001000 00010100
01100110 01111011 01111001 00001011 10000001 10011010 00101011 00110011
00010010 10100101 01001101 10010000 01001000 01001011 11001011 10111001
11001010 00010000 00111110 01111101 01110101 11000111 00100010 10100100
10011111 00100101 10100101 10011001 01011111 01110010 00000001 11000100
01111101 01001100 00011001 00111101 01111010 11111110 11111111 00001000
00110001 10100000 00110101 01011100 00100001 11001010 00100000 10101110
10011100 01001000 01000101 11000011 10100010 10000010 01111010 01101001
01011010 00100011 11011000 10000101 00011111 10011110 10100100 11101110
11010101 00100101 00010001 10010010 00011101 10010110 10011100 11101100
10110000 00111101 00110011 11001101 00011000 11100110 11001001 01011100
11001101 10001010 10110000 10000000 10001001 01110101 11111001 11001100
10011000 01110110 01111010 11000111 01011110 01001110 01100001 01001110
11111101 11101111 00100101 01011011 11001001 01100001 11011001 01001110
00010110 01100101 01001110 11011110 11100010 11011111 11010110 00111011
01101000 11111001 10010001 00010111 11110011 00101110 10011101 00001010
01111100 00001100 10101011 10100101 10010000 01000011 00011001 00010000
10010011 00111011 00111101 11010010 11000001 10100011 11111001 00011100
01110011 00110111 10110100 11011100 00001110 01000110 01001010 00000010
01011110 00001010 11111010 11111011 11000010 01100110 00000000 01011001
10000010 01000101 01111000 01111001 01101100 01100000 01111101 11001011
01111000 00111000 10111010 10111010 10011101 00010101 01000010 10000000
```

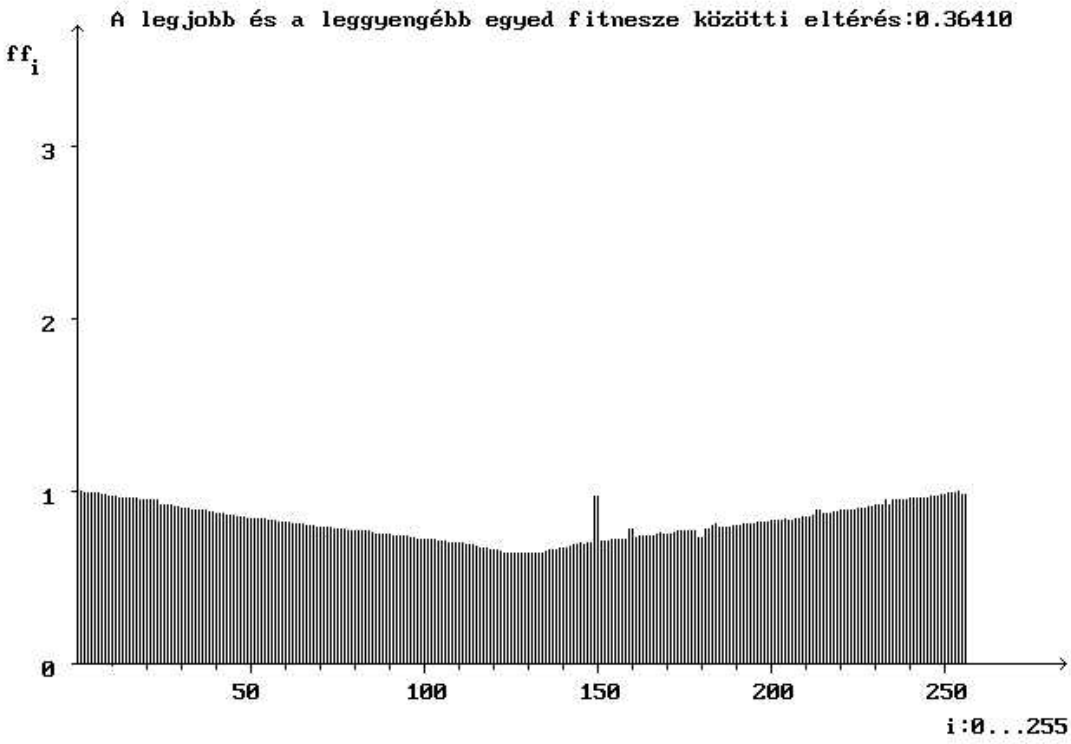
22. ábra. A generált kezdeti populáció kromoszómái.



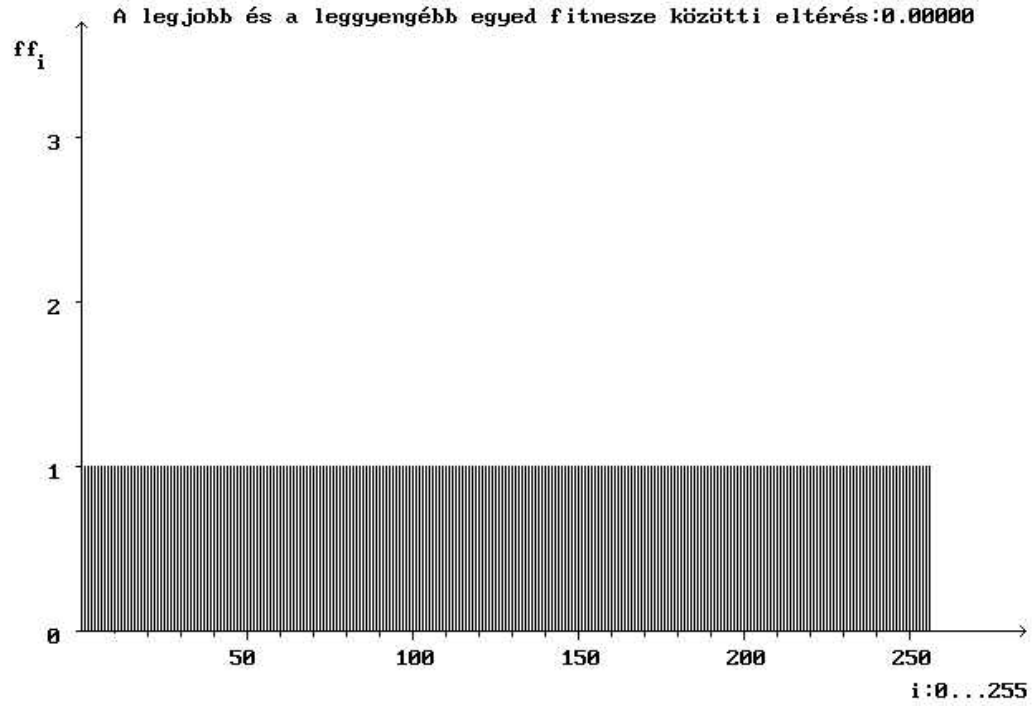
23. ábra. A kezdeti populáció fitneszértékei



24. ábra. A kezdeti populáció fitneszértékei csökkenő sorrendben.



25. ábra. Egy kialakult új populáció fitneszértékei.



26. ábra. A kilépési feltételnél létrejött populáció fitneszértékei.

```
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
11000011 11000011 11000011 11000011 11000011 11000011 11000011 11000011
```

27. ábra. A genetikus algoritmus lefutása utáni kialakult populáció kromoszómái.

Tapasztalatom szerint az algoritmus a 7. populációnál eléri a kívánt eredményt. Ennél az algoritmusnál felléphet az a probléma, hogy az algoritmus „beragad”, nem kapjuk meg a 11000011 kromoszómákat.

Ez akkor fordulhat elő amikor a kezdeti, véletlenszerűen létrehozott populációban eleve nem szerepel a kívánt tulajdonságú egyed, ilyenkor a kilépési feltételnél figyelni kell a létrejött populációk számát. Például a 20. populációnál álljon le az algoritmus, ha nincs meg a kívánt eredmény. Lehetséges, hogy esetleg mutációval létrejön a kívánt tulajdonságú egyed, de ennek az esélye eléggé kevés.

Itt két példán keresztül mutattam be a kanonikus algoritmus működését. Mint látjuk a legfontosabb lépések a következők:

- Kezdeti populáció véletlenszerű létrehozása
- Célfüggvény meghatározása
- Fitneszfüggvény meghatározása (skálázás)
- Genetikus műveletek (keresztezés, mutáció)
- Új populáció létrehozása
- Kilépési feltétel

7. GENETIKUS ALGORITMUSOK ALKALMAZÁSAI

A genetikus algoritmust akkor érdemes használni, ha:

- A probléma keresési tere kezelhetetlenül nagy
- Nem ismert a struktúrája
- Nem áll rendelkezésre területspecifikus tudás
- Nem ismert egzakt, gyors algoritmus
- Nincs szükség a pontos globális optimumra, csak egy stabil, jó megközelítés kell
- A kiértékelő függvény pontatlan, zajos [SWÁ5]

Alkalmazási területek

A genetikus algoritmusokat gyakran mint függvényoptimalizáló eljárásokat emlegetik, ugyanakkor felhasználási területük ennél sokkal szélesebb. A fent vázolt Holland-féle genetikus algoritmus egyszerű struktúrájú, de az alapeljárás különféle változatai igen nagy számban fordulnak elő a tudományos és műszaki élet különböző területein. Néhány példa erre:

Optimalizálás

A genetikus algoritmusokat az optimalizálási feladatok igen széles körében használják megoldási módszerként. Mind a numerikus, mind a kombinatorikus optimalizálási feladatok körében. Például: áramkörök tervezése, munkaütemezés.

Automatikus programozás

A genetikus módszerek alkalmazhatók számítógépes programok kifejlesztésére, adott feladatok megoldására, illetve különféle számítási struktúrák tervezésére. Például celluláris automata, rendező hálózatok.

Gépi tanulás

Számos gépi tanulási alkalmazásban is előkerülhetnek a genetikus algoritmusok, beleértve az osztályozási, predikciós feladatokat mint például az időjárás előrejelzés, vagy a fehérjestruktúrák kutatása. A genetikus algoritmus használható különféle gépi tanulás szempontjainak kialakításánál, például a neurális hálózatok súlyvektoránál, tanuló osztályozó rendszereknél, szimbolikus produkciós rendszerek szabályainál, robotszenzoroknál.

Közgazdaságtan

A genetikus eljárások használhatók gazdasági reformok, újítási folyamatok modellezésére, árverési stratégiák kifejlesztésére vagy piackutatásra is.

Immunrendszerek

Genetikus algoritmussal lehetséges a természetes immunrendszerek különféle szempontjainak modellezése, beleértve az egyén élete során bekövetkező testi mutációt, illetve a multigenetikus családok felismerését az evolúciós idő során.

Ökológia

A genetikus algoritmus szintén alkalmazható ökológiai jelenségek, például biológiai „fegyverkezési verseny”, gazda-élősködő koevolúciója, szimbiózis, erőforrás áramlás modellezésére.

Populációgenetika

A genetikus módszerek segítségével tanulmányozhatók a populációgenetika területének kérdései, például: „Milyen feltételek mellet lesz egy adott gén rekombinációra nézve életképes az evolúció során ?”

Evolúció és a tanulás

A genetikus algoritmussal kutatható, hogyan hat egymásra az egyéni tanulás és a fajok evolúciója.

Szociális rendszerek

A társadalmi rendszerek evolúciós szempontjai is tanulmányozhatók a genetikus algoritmus segítségével. Például rovarkolóniák szociális viselkedésének evolúcíója, vagy általánosabban fogalmazva az együttműködés és kommunikáció fejlődése multi-ágenses rendszerben. [ÁGyHV10]

Az alábbiakban néhány alkalmazást ismertetek a teljesség igénye nélkül.

7.1 Gráfszínezési probléma

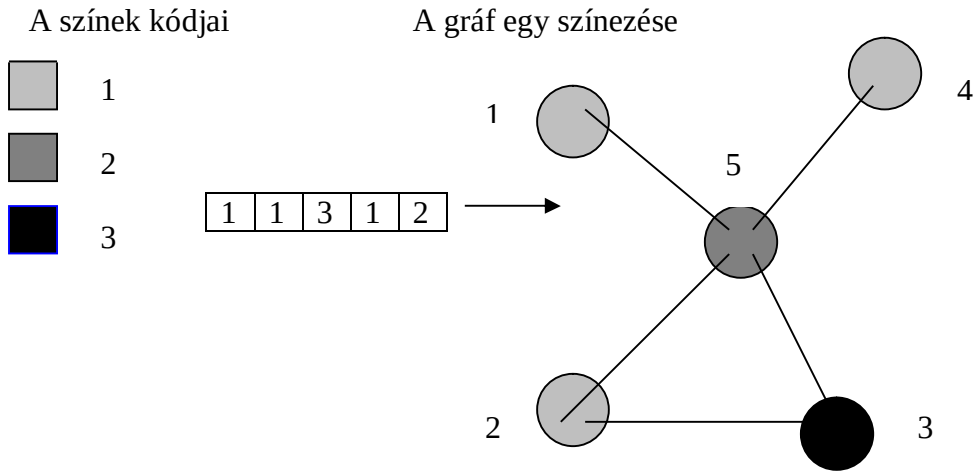
Egy adott irányítatlan gráfhoz találni kell egy jó *k* színezést, ami azt jelenti, hogy minden csúcshoz az előre adott *k* különböző szín valamelyikét kell rendelni úgy, hogy minden él két különböző színű pontot kössön össze.

A genetikus algoritmus a populációk sorozatát állítja elő operátorok segítségével. Ennél a feladatnál egy megoldás a gráf egy (nem feltétlenül jó) színezése. Ahhoz, hogy ezekből a megoldásokból egyszerű **operátorok** segítségével könnyebben új megoldásokat lehessen szerkeszteni, a megoldásokat **kódolni** kell. Ez azt jelenti, hogy a megoldásokhoz hozzárendelünk egy szintaktikailag jól manipulálható betűsorozatot egy **kódoló függvény** segítségével.

Ez a kódolás és a hozzátartozó operátorok a genetikus algoritmus kulcsfontosságú részei, mivel ezek határozzák meg a keresési tér topológiáját. Itt kétfajta kódolást vizsgálok: a direkt és a sorrendi kódolást.

7.1.1 A színezés direkt kódolása

Példánkban a gráf csúcspontjainak száma 5, a rendelkezésre álló színek száma 3.



28. ábra. A gráfszínezés direkt kódolása

A fenti gráf egy lehetséges színezése: **1 1 3 2 1** ez a direkt kódolás. A dekódoláshoz tudni kell, hogy melyik csúcshoz melyik szín tartozik. A felsorolásnál a gráf pontjait egy előre rögzített sorrendben kell érteni.. A kód hossza megegyezik a gráf pontjainak számával.

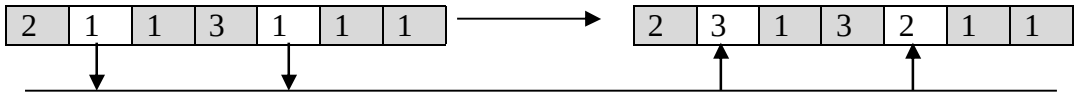
Ennek a kódolásnak az az előnye , hogy segítségével könnyű előállítani véletlenszerű színezéseket, hiszen az egyes színkódokat egymástól függetlenül lehet megadni. A **rátermettségi függvényt** úgy definiálhatjuk, hogy a rosszul színezett, azaz azonos színű pontokat összekötő élek számának a függvénye legyen: Úgy érdemes eljárni, hogy a különböző színű pontokat összekötő élekhez *0* értéket, a megegyező színű pontokat összekötő élekhez *-1* értéket rendelünk. Így a rátermettségi függvény értéke a rossz élek számának a -1 szerese lesz. A függvény maximuma 0 lesz, minden jó színezésnél a rátermettségi függvény értéke 0.

Legyen a gráf éleinek a száma: ***e*** **,** a megegyező színű pontokat összekötő élek száma *k*, akkor a rátermettségi függvény: $f(e) = -k$ Minden jó színezésnél $f(e) = 0$

A kódokon alkalmazott operátorok

- **Mutációk**: egy megoldásból állítanak elő egy új megoldást
- **Rekombinációk**: több kiindulási megoldásból (szülők) állítanak elő új megoldást

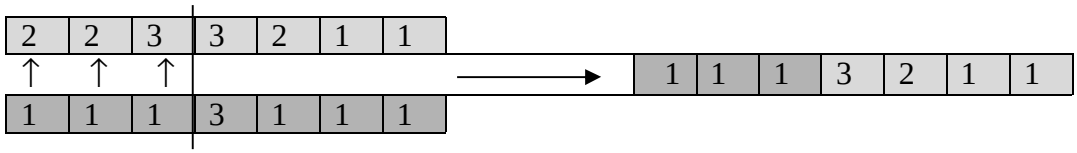
Mutáció: Egy megoldást alakítunk át. Véletlenszerűen kiválasztott pozíciókat változtatunk meg. A pozíció kiválasztására alkalmazható módszer: Minden pozíció egy kicsi (általában rögzített) valószínűséggel mutálódhat. . Sokszor úgy állítják be ezt a valószínűséget, hogy átlagosan egy pozíció változzon egy kódban.



29. ábra. Mutációs operátor

Rekombinációk: két fajtája van . *Az egypontos keresztezés* ötlete a biológiából származik. Az egyenletes keresztezés nem rendelkezik biológiai gyökerekkel. A probléma-megoldásnál a választott operátorokat a megoldani kívánt feladat befolyásolja.

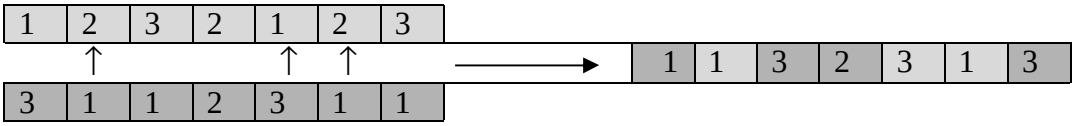
Egypontos keresztezés: kiválasztunk két szülőt, és véletlenszerűen egy keresztezési pontot, és a keletkezett kódrészekből új megoldást rakunk össze.



30.ábra. Rekombináció: egypontos keresztezés

Ez a keresztezés megvalósítható úgy is, hogy több szülőt és keresztezési pontot választunk.

Egyenletes keresztezés: Hasonlóképpen mint a mutációnál, a kiválasztott pontokon a kiindulási kódok értékeket cserélnek. A mutációtól eltérően itt egy pozíció kiválasztásának valószínűsége 0,5 , azaz átlagosan az értékek fele cserélődik ki.



31. ábra. Rekombinációk: egyenletes keresztezés

7.1.2 A színezés sorrendi kódolása

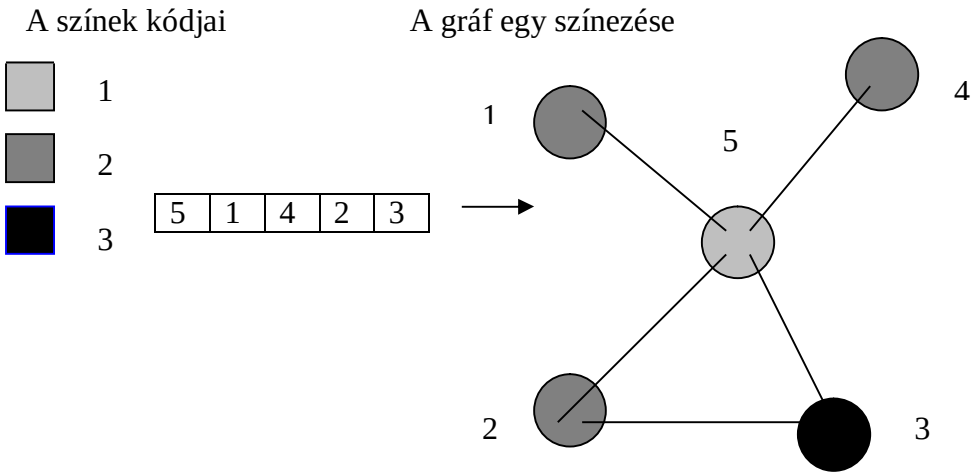
A sorrendi kódolásnál az elemek a színezendő gráf pontjai lesznek. Egy színezést nem lehet pusztán a gráf pontjainak a sorrendjével megadni. Ehhez szükség van egy egyszerű heurisztikára, amely a pontok egy sorrendjéből kiindulva elvégzi a színezést. Vegyük sorra a gráf pontjait, és minden pontot színezzünk arra a minimális sorszámú színre, amelynek használata az adott esetben lehetséges. Ha egyetlen szín sem megfelelő hagyjuk a pontot színezetlenül. Ez az algoritmus minden pontsorrendhez hozzárendel egy színezést. Ha létezik jó színezés, akkor minden pontnak lesz színe, egyébként maradnak színezetlen pontok.

A **rátermettségi** vagy **fitnesz** függvényt a következőképpen definiálhatjuk: azokhoz a pontokhoz, amelyek kaptak színt *0* értéket, melyek nem kaptak színt *-1* értéket rendelünk.

Ha egy *n* csúcspontú gráfnál *k* csúcspontot nem tudtunk kiszínezni, akkor a fitneszfüggvényünk az:

$$f(n) = -k \quad (0 \leq k < n)$$

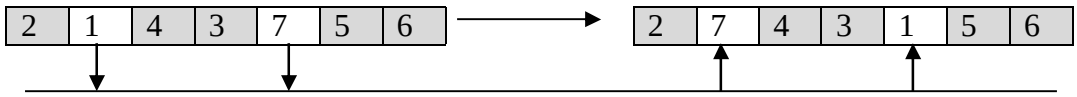
Ennek a függvénynek a maximuma **0** . A lehetséges színezéseknél a függvényértéke nulla, tehát nincs kiszínezetlen pont.



32. ábra. Dekódolás végeredménye az adott pont-sorrendre alkalmazva a feltüntetett szín-sorszámokkal.

Itt az operátorok bonyolultabbak mint a direkt kódolás estén. Itt is alkalmazható a mutáció valamint a rekombináció, melynek neve: **OX** rekombináció *(order crossover)*

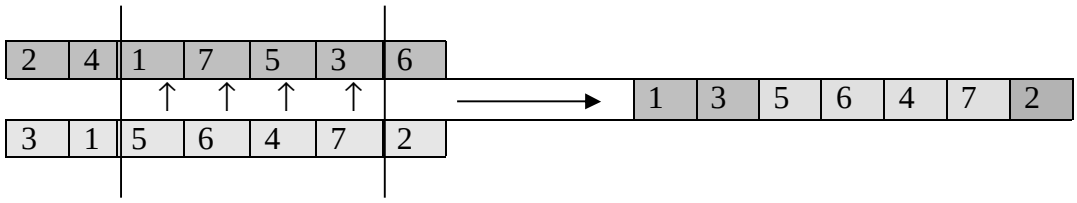
Mutáció: a mutáció leggyakrabban egy véletlenszerűen kiválasztott elemi permutáció végrehajtását jelenti.



33. ábra. Az elemi permutáció mint sorrendi mutáció

Rekombináció: Ennél a kódolásnál több problémát okoz mint a direkt kódolásnál, mert az egyes pozíciókon lévő kódok erősen függnek egymástól. Egy lehetséges sorrendi rekombináció az OX *(order crossover).*

Az OX rekombináció végrehajtása két keresztezési pont véletlenszerű kiválasztásával kezdődik, majd a két pozíció közötti részek kicserélődnek. A fennmaradó részek kitöltéséhez megvizsgáljuk, hogy az eredeti szülőben az új középső részben nem szereplő pontok milyen sorrendben következnek és ezt a sorrendet követve a második keresztezési ponttól kezdve beírjuk a hiányzó pontokat.



34. ábra. Az OX rekombináció

Akár direkt vagy sorrendi akár másfajta reprezentációt használunk egy problémamegoldásra, nincs általános recept az operátorok megválasztására. Az egyes operátorok különböző feladatokon nagyon eltérő teljesítményt produkálhatnak, másfelől nincsenek általánosan alkalmazható elméleti eredmények, amelyek segíthetnének az operátorok megválasztásban. Tehát az operátorok alkalmazása problémafüggő. Érdemes többfélét kipróbálni. [FI3]

Problémafüggetelen komponensek

A gráfszínezési probléma kapcsán különböző kódolási technikákat és a hozzátartozó operátorokat ismertettem. Ezeknek az a feladata, hogy ismert megoldásokból újakat állítsanak elő. A kódolás operátorai akkor megfelelőek, ha rátermett megoldásokból kiindulva a leszármazott megoldások is hasonlóan jó tulajdonságokkal rendelkeznek. Feltéve, ha a kódolás jó, nem mindegy, hogy az operátorokat milyen megoldásokra alkalmazzuk amikor a soron következő populációt szeretnénk előállítani. Ezeknek a „szülőknek” a kiválasztása a **szelekciós operátor** vagy egyszerűen a **szelekció** feladata, amely már problémafüggetlen. Tehát a szelekció az adott populációból kiválaszt számunkra egy darab megoldást, és ezt annyiszor kell megismételni, ahány szülőre szükség van az új populáció létrehozásához.

A szelekció megvalósításának az alapja a rátermettség vizsgálata. Itt három módszert alkalmazható*: Rátermettség-arányos szelekció, pár-verseny szelekció, rangsorolás*. [FI4]

Rátermettség-arányos szelekció

A módszer valószínűségi mintavételt használ. Egy kiválasztás valószínűsége annál nagyobb, minél nagyobb a rátermettsége a populáció rátermettségi átlagához képest. A szelekció matematikai értelemben egy mintavétel a populációból. A populáció minden ***e*** elemére a kiválasztás valószínűségét megadhatjuk a már ismert

$$P(e) = \frac{f(e)}{n \cdot f(pop)}$$ képlettel, ahol

f(e) a rátermettség értéke, ***n*** a populáció elemszáma, és *f(pop)* a populáció tagjainak átlagos rátermettsége. Itt feltettük, hogy a rátermettség pozitív. [SWÁ6]

Pár-verseny szelekció

Ez talán a legegyszerűbb módszer sok alkalmazásban lehet vele találkozni. Kiválasztunk a populációból két megoldást véletlenszerűen. Vizsgáljuk a két egyed rátermettségét, a szelekció által kiválasztott elem a kettő közül a rátermettebb. Ez a technika úgy általánosítható, hogy nem kettő hanem több elem győztesét választjuk ki. [SWÁ7]

Rangsorolás

A rátermettség-arányos szelekcióhoz hasonló. A rátermettség értékét közvetlen módon használó módszerek problémája, hogy túlságosan érzékenyek a rátermettség eloszlásra a populációban. Ha például egy megoldás magas rátermettségi értékkel rendelkezik a többihez képest, akkor szinte mindig ő lesz a kiválasztott szülő, aminek káros hatásai vannak, hiszen a keresés a változatosság gyors csökkenése miatt „beragadhat”; az aktuális legjobb megoldást tovább javítani egyre nehezebb lesz. Ezt a problémát úgy oldhatjuk meg, hogy a populáció elemeit rátermettség szerint sorba rendezzük, és a rátermettség helyett valamilyen „szép”, a rendezésben elfoglalt hely szerint egyenletesen növekvő függvényt használunk. Gyakori a **lineáris rangsorolás**, amikor ez a „szép” függvény egyenes. [FI5]

7.2 Órarend készítés

A oktatásban kap meghatározó szerepet. Régebben a feladatot emberi erővel oldották meg az iskolákban. Ma már rendelkezésre állnak órarendkészítő programok, amelyek a különböző feltételek megadása után gyorsan elkészítik az órarendet. Az adatok bevitele és a feltételek megadása sokkal hosszabb időt vesz igénybe mint a program futási ideje.

A munka során nagyon sok feltételt kell figyelembe venni.

- Minden tanárnak fix számú órája van bizonyos osztályokban
- Nincs olyan osztály és tanár, akinek egyszerre két órája van.
- Minden tanárnak egy héten lehetőleg legyen egy szabad napja.
- Az osztályoknak csak a nap elején vagy végén lehet lyukas órájuk.
- Ha egy tanárnak több órája van egy nap egy osztállyal, akkor lehetőleg azok egymás után legyenek.
- Egy tanárnak lehetőleg ne legyen sok lyukas órája [SWÁ8]

Egy átlagos intézmény órarendjén egy gyakorlott pedagógus is közel egy hétig dolgozik, ennek ellenére az elkészült órarend gyakran nem ideális, nem felel meg az elvárásoknak. Sőt ha változik az órarend, a bemeneti feltételek módosulnak (egy új tanár érkezik vagy egy tanár távozik, vagy egy termet nem lehet átmenetileg használni), az eddig elvégzett munka szinte semmibe vész, lehet elölről kezdeni az optimalizálást.

A kézi órarend készítés során többnyire csak a legfontosabb szempontokat tudják figyelembe venni, ugyanis ez a redukált feladat is rendkívül nehéz. Olyan megoldást kell keresni, amely nemcsak azokat a feltételeket tartja be, amelyek elkerülhetetlenek egy működőképes órarendhez, hanem a közel hasonlóan fontos didaktikai és szervezési szempontokat is. A genetikus algoritmus kellően erős egy közelítőleg optimális órarend elkészítéséhez. Az alkalmazott genetikus algoritmussal kezelhetők az előző problémák, sőt kezelhetők a bonyolult óraösszevonások és szétosztások.

Órarend reprezentációk

Hagyományos órarendreprezentáció

A kézi órarendkészítés hagyományait veszi alapul ez a kétdimenziós ábrázolás. Ekkor általában a függőleges tengelyen az osztályokat, a vízszintes tengelyen az órák időintervallumát vesszük fel. Az így kapott kétdimenziós mátrix (*i,j)* eleme azokat a tanárokat tartalmazza, akik a *j*-edik órában az *i*-edik osztályban tartanak órát.

	1. nap				…	n. nap			…
	1. óra	2. óra	…	*n*. óra	…	…	*j*. óra	…	…
1. oszt	Tóth L.	Biró I.							
2. oszt	Szabó L.	Kovács G.							
. . .									
i. oszt							Kiss I.		
. . .									

35. ábra. Kétdimenziós órarendreprezántáció

Egy ehhez hasonló másik reprezentáció szintén kétdimenziós, a vízszintes tengelyén az időt, míg a függőleges tengelyén a tanárokat tüntetjük fel. A mátrix *(i ,j)* eleme azokat az osztályokat tartalmazza, amelyeket a *j*-dik órában az *i*-edik tanár tanít.

Ezeknek az ábrázolásoknak az az előnye, hogy ránézésre is könnyű megállapítani, hogy egy tanár egy elfoglalt-e egy adott időben vagy ki tart órát egy bizonyos időpontban és osztályban. Az ilyen ábrázolások alapesetben sem támogatják a bontott órákat, amikor egyidőben az osztály egyik felének egy bizonyos tantárgyból bizonyos tanárral, míg az osztály másik felének más tantárgyból más tanárral van órája. Ez a módszer azt az esetet sem támogatja, amikor több osztálynak egyidejűleg több tanár tart órát. Például bontott nyelvóráknál 2 osztályt 4 tanár is taníthat. [ÁGyHV11]

Halmazos reprezentáció

A hagyományos reprezentációk esetén felmerülő problémákat igyekszik orvosolni a halmazos reprezentáció, és emellett biztosítja a legnagyobb szabadságot az órabontások és összevonások valamint a tanár és tantárgy hozzárendelések terén.

A legkisebb adategység a halmaz, egy olyan struktúra, amely tetszőleges számú osztályt, tanárt és tantermet tartalmaz.

Amennyiben egy osztályba két tanárt kell felvenni (Például emelt szintű és normál matematika oktatás esetén) –akkor a két tanárt fel kell venni a halmazba.

Abban az esetben amikor két osztályhoz egy tanárt rendelünk (például összevont testnevelés óra esetén) a két osztályt és az egy tanárt felveszzük a halmazba.

Bonyolultabb problémák is könnyen megoldhatók. Idegen nyelv oktatás során lehet olyan eset, amikor 2 osztályt 5 tanár tanít, ilyenkor az osztályokat és a tanárokat felvesszük a halmazba.

1. nap			2. nap			…
1. óra	2. óra	…	…	…	…	…
$Halmaz_1$: $osztály_{0_{1,1}}, osztály_{0_{1,2}}, \ldots$ $\tan ár_{t_{1,1}}, \tan ár_{t_{1,2}}, \ldots$ $terem_{te_{1,1}}, terem_{te_{1,2}}, \ldots$	$Halmaz_{N+1}$: $osztály_{0_{N+1,1}}, osztály_{0_{N+1,2}}, \ldots$ $\tan ár_{t_{N+1,1}}, \tan ár_{t_{N+1,2}}, \ldots$ $terem_{te_{N+,1}}, terem_{te_{N+1,2}}, \ldots$					
$Halmaz_2$	$Halmaz_{N+2}$					
. . .	. . .	…				
$Halmaz_N$	$Halmaz_{2N}$					

36. ábra Halmazos órarend-reprezentáció

Ebben a reprezentációban nem két, hanem csak egy dimenzióban dolgozunk, amely nem más mint az idő. Az időtengelyen lévő időrésekbe melyek a lehetséges órákat jelentik-kell halmazainkat elhelyezni. Az $o_{i,j}$, $t_{i,j}$, $te_{i,j}$ értékekkel az i-edik halmazban szereplő osztályok, tanárok, termek sorszámát jelöljük, j pedig a halmazon belüli sorszámok indexe.

Egy időrésbe több halmaz is kerülhet, itt ügyelni kell arra, hogy ne legyen ütközés, tehát ne kerüljön egy időrésbe két olyan halmaz, melyben közös tanár vagy közös osztály szerepel.

[ÁGyHV12]

Egy órarend jóságának elemzése.

Az órarendjelöltek rangsorolására, értékelésére büntetőpontokat alkalmaznak. Meg kell határozni, hogy milyen szituációkat kell elkerülni ahhoz, hogy közelítőleg optimális eredményt kapjunk. Ehhez be kell vezetni az úgynevezett **kemény** és **lágy** megkötéseket

A kemény kikötéseket szigorúan kell venni, ha ezeket a szabályokat megszegjük az órarend használhatatlan. A lágy megkötések megszegése esetén az órarend még használható, de didaktikai szempontból előnytelen. Kellemetlenséget okoz a diákoknak, tanároknak, melyek nehezítik az ismeretek elsajátítását.

Kemény kötések

- **Tanárütközés**ről akkor beszélünk, amikor egy tanárnak egyidejűleg több osztály számára kéne órát tartania. Bontott órák esetén, ha több osztályból gyűlik össze a csoport, akkor nem beszélhetünk tanárütközésről.
- **Teremhiány** jelentkezhet akkor, ha egy osztály számára nem jut tanterem, ahol az órát megtudnák tartani. (korlátozott erőforrás)
- **Tanárok kemény ráérése.** A tanároknak iskolán kivűli feladatai is lehetnek, melyeket kötelező ellátni. Így például előfordulhat, hogy egy tanárnak nem lehet órája a pénteki napon.

Lágy kötések

- **Lyukasóra**. Általános iskolában nem megengedhető, középiskolában is nehezen tolerálható, ha valamelyik osztály órarendjében lyukasóra van.
- **Nulladik óra.** A tanulók számára megterhelő a nulladik vagy esetleg a 7. 8. óra.

- **Többszörös óra**. Egy osztálynak, ne legyen egy tárgyból egy nap több órája. Például heti két fizika óra esetén ne essen ez a két óra egy napra. Vannak olyan esetek heti nagy óraszámnál, hogy a dupla óra indokolt, például magyarból dolgozatírás esetén.
- **Héten belüli egyenletes óraeloszlás**. A napi óraszámok között lehetőleg kis eltérés legyen az egyenletes terhelés érdekében. Például a hét egy napján a tanulóknak 4 a többi napokon 6 órájuk van
- **Tanárok lágy ráérése.** Előfordulhat, hogy egy tanár, más intézményben is tanít. Ilyenkor kérést fogalmaz meg, hogy a hét melyik napján, milyen időben ne osszák be.

A kémény illetve a lágy feltételek megsértéséért **büntetőpontokat** adunk, ami azt fejezi ki, hogy milyen mértékben teszi használhatatlanná az órarendet az adott probléma fennállása. Legnagyobb gondot a kemény feltétek megszegése jelenti, ezért ezeknek a pontszámát kell a legnagyobbra állítani, a lágy feltételek megsértéséért arányosan kisebb pontszámot adunk. [ÁGyHV13]

Egy lehetséges büntetőpontozás a következő:

Osztályütközés	1000
Tanárütközés	1000
Teremhiány	1000
Kemény tanárráérés	500
Többszörös órák	100
Napközi lyukasóra	100
Első nap eleji lyukasóra	50
Napi óraszám egyenetlensége	50
Heti egyenletesség	20
Napon belüli szakadozottság	20
Lágy tanárráérés	10

Az órarendkészítésnél az optimalizálást egy kezdeti populáció felvételével kezdjük, ez általában véletlenszerű. Ez úgy történik, hogy az elhelyezésre váró halmazok közül sorszámuk szerint egyet kisorsolunk és az időtengelyre tesszük. Amennyiben rendelkezésünkre áll egy korábbi hasonló probléma megoldása, például a tavalyi órarend, akkor ez is vehető kiindulási alapnak.

Kiválasztási módszernek egyszerűbb esetben a rulettkerék kiválasztás használható. A konvergenciasebesség növelése érdekében javulás érhető el ennek továbbfejlesztett változatával, a versengő szelekcióval.

Az új populációk létrehozása során konzisztencia megőrző egypontos keresztezés használható, ilyenkor az elemtípusok száma változatlan. (Például: egy adott osztályban az adott 3 biológia óra nem csökkenhet, vagy nem nőhet meg) Ezeknél a módszereknél bonyolultabb, de sokkal hatékonyabban használhatók a speciálisan a permutációkra kidolgozott operátorok A kilépési feltételnél figyelembe kell venni a büntetőpontok értékét, hogy mekkora büntetőpontértéknél elfogadható az órarend. [ÁGyHV14]

7.3 „Fogoly dilemma” játékmodell

Szenteste foglyul ejtenek két embert, egy közeli bankrablás két gyanúsítottját. Az ügyész felkeresi mindkét - külön cellában elhelyezett - embert, és a következő javaslatot teszi nekik: amennyiben egyikőjük a másik ellen vall, és a másik nem vall, a vallomását felhasználják, őt pedig szabadon engedik. Amennyiben mindketten egymás ellen vallanak, akkor mindkettőjüket elítélik fegyveres rablásért, de az együttműködési készség miatt előbb szabadulnak, mint abban az esetben, ha csak egyikőjük vall. Ha egyikőjük sem vall a másik ellen, akkor a legsúlyosabb büntetést kapják, ami tiltott fegyverviselésért kiszabható. A rabok nem tudnak egymás döntéséről, és mindketten sokkal jobban aggódnak a személyes szabadságukért, mint a társukéért, ráadásul Karácsony van, és hazafelé igyekeznének a családjukhoz. Kérdés, hogy mit választanak, az együttműködést (kooperálást), vagyis nem vallanak, vagy a másik feladását?

A „Fogoly dilemmája” elnevezésű játékmodell jól szemlélteti, hogy a genetikus algoritmus miként találja meg a kutatás és a kiaknázás egyensúlyát. A játékban mindkét játékos az „együttműködés” és az „árulás” kötött választhat. Ha együttműködnek, az mindkettőjüknek kifizetődik. Ha az egyik áruló lesz, még többet nyer, míg a másik fél nem nyer semmit. Ha pedig mindketten árulók lesznek nyereségük minimális.

A politológusok és a szociológusok azért tanulmányozták behatóan a „Fogoly Dilemmáját”, mert ez a játék egyszerűen és szemléletesen példázza az együttműködés nehézségeit. A játékelmélet szerint mindkét játékosnak a lehető legcsekélyebbre kell csökkenteni azt a kárt, amelyet a másik okozhat neki, vagyis várható, hogy mindketten árulók lesznek. A valóság viszont az, hogy ha a játszma többször ismétlődik, a résztvevők közös hasznuk növelése érdekében megtanulnak együttműködni. A „Fogoly

Dilemmájának” egyik leghatékonyabb megoldási stratégiája a „szemet szemért elv”: kezdetben a két játékos együttműködik, majd mindegyik utánozza a másik legutolsó játszmáját, vagyis az árulásra árulással válaszol.

Robert Axelrod (Michigani Egyetem) és Stephani Forrest (Új-Mexikói Egyetem) megvizsgálta, hogy a genetikus algoritmus fel tudja-e fedezni a „szemet szemért” stratégiát? A genetikus algoritmus alkalmazásához a lehetséges stratégiákat először füzérekre kell lefordítani. Ennek egyik egyszerű módszere, hogy a soron következő játék a három megelőző játék kimenetele alapján választják meg.

Minden játéknak négy lehetséges kimenetele van, így a három játék összesen 64 lehetőséget kínál. Egy 64 bites füzér egy-egy gént (illetve bitet) tartalmaz mindegyik lehetőség számára. Az első gént például a három egymást követő együttműködés esetéhez rendelik, míg az utolsó gén három egymást követő árulást jelent. Minden gén értéke 1 vagy 0 lehet aszerint, hogy az előző játszmák alapján az együttműködés vagy az árulás lenne-e a megfelelő válasz. A csupa 0-ból álló 64 bites füzér azt jelenti, hogy minden esetben az árulás a követendő stratégia. Még ilyen egyszerű játék esetében is 2^{64} (kb. 18 trillió) a lehetséges stratégiák száma.

Axelrod és Forrest a stratégiákat megjelenítő füzérek véletlenszerű gyűjteményét táplálták be a genetikus algoritmusba. A füzérek rátermettségét egyszerűen az fejezte ki, hogy az általuk képviselt ismételt játszmák során átlagosan mennyire fizetődik ki. Mindegyik füzér rátermettsége csekély volt, hiszen a „Fogoly Dilemmában” a legtöbb stratégia nem valami jó. A genetikus algoritmus gyorsan felfedezte és kihasználta a „szemet szemért” elvet, de a további evolúció egy újabb felfedezést is hozott. Az új stratégia abból állt, hogy amikor a genetikus algoritmus már magas szinten játszott, kihasználta a becsapható játékosokat: együttműködésre csábított, miközben valójában az árulás stratégiáját követte. Amikor viszont az előző játékok története az jelezte, hogy az ellenfél nem csapható be, visszatért a „szemet szemért” elvhez. [SWÁ9]

7.4 Földgáz vezeték vezérlése

David E. Goldberg (Illionois-i Egyetem) algoritmusai megtanulják az Egyesült Államok délnyugati részéről kiinduló földgáz vezeték alapján mintázott modell vezérlését. A csővezeték sok elágazást tartalmaz, s ezek mind különböző mennyiségű gázt szállítanak. Vezérlés eszközei az egyes ágakban uralkodó nyomás növelésére szolgáló kompresszorok, tárolótartályok és a csövek közötti gázáramlást szabályozó szelepek. A

szelepek, valamint a vezetékben bekövetkező nyomásváltozás között hatalmas az időeltolódás, így a vezérlés problémájának nincs analitikus megoldása. A vezérlést végző személynek akárcsak Goldberg algoritmusának tanulóidőre van szüksége.
Goldberg nem csupán a gázszükséglet fedezését oldotta meg az eddigieknél nem magasabb költséggel, hanem egy alapértelmezési szabályokból álló hierarchiát is kialakított, amely képes helyesen reagálni, ha például a csővezeték kilyukad márpedig ez a valóságban időnként előfordul. [SWÁ10]

7.5 Kommunikációs hálózatok

Lawrence Davis (Tica Assosiates, Cambridge, Massachusetts állam) hasonló módszert alkalmazott a kommunikációs hálózatok fejlesztésében. Szoftvere a lehető legtöbb adat átvitelére hivatott a lehető legkevesebb adatátviteli vonalon és a lehető legkevesebb átkapcsolással. [SWÁ11]

7.6 Repülőgép sugárhajtóművek

A General Electric és a Rensellaer Műegyetem kutatócsoportja a gyakorlatban is kipróbálta a repülőgép-sugárhajtóművek tervezésére kidolgozott genetikus algoritmust. A turbinák tervezésénél legalább száz változót kell figyelembe venni, s ezek mind különféle értéktartományba tartoznak. Így több mint 10^{387} pontot magába foglaló keresési tér jön létre. A sugárhajtómű „rátermettsége attól függ, mennyire elégíti ki a mintegy félszáz kényszerfeltételt ilyen például a belső és külső falak alakja, az áramlás nyomása, sebessége és turbulenciája. A genetikus algoritmussal a tervváltozat kiértékelése csak 30 másodpercet vesz igénybe a számítógépes munkaállomásokon. [SWÁ12]

ÖSSZEFOGLALÁS

Dolgozatomban elsősorban az egyszerű genetikus algoritmusokkal foglalkoztam. A genetikus algoritmusok másik fajtája a fejlett genetikus algoritmusok. Ezek jóval bonyolultabbak. A fejlett genetikus algoritmusok témakörébe tartozik a szkémaelmélet, párhuzamos genetikus algoritmusok (egyszerre több populációt tart fenn), fa-alapú kódolás, szigetmodellek, többkritériumos genetikus algoritmusok, mesterséges fitneszdomborzatok stb. Dolgozatomat elsősorban azoknak ajánlom, akik még nem foglalkoztak a genetikus algoritmusokkal, és az első lépéseket szeretnék megtenni megismerésük felé. Ez a tevékenység nem lehet csak elméleti megalapozás, hanem valamilyen gyakorlati alkalmazás megvalósítása is fontos hozzá. Erre jó alapot adhat a bemutatott két program további fejlesztése. Mindig a probléma megfogalmazása a fontos, hogyan lehet a problémát kromoszómákkal kódolni (kezdeti populáció), mi a cél, (célfüggvény), mi jellemzi a jó megoldásokat (fitneszfüggvény), valamint milyen a genetikus műveleteket (keresztezés, mutáció) alkalmazzunk. A részletesebben tárgyalt kanonikus algoritmus szerintem középiskolában, szakkörön is feldolgozható, csak ötletek kellenek a célfüggvényre.

Végül azokat a gondolatokat szeretném megfogalmazni, melyek dolgozatom írása közben megfogalmazódtak bennem.

A világban számos helyen folyik a genetikus programozás kutatása. Gondolom kiváló szakemberek csúcs technológiájú eszközökkel, nagy költségvetéssel, teamekbe szerveződve vizsgálják ezt a módszert. A véleményem azonban az, hogy az fiatal amatőrök részére is van keresni való, sőt lehet, hogy ők fognak nagy áttörést elérni, különösen ha összefognak és megosztják egymással az "Információt” tapasztalataikat, ötleteiket sikerüket és kudarcukat is. Az amatőrök egyik nagy előnye a profikkal szemben, hogy nincsenek megcsontosodott, a szakmában beléjük nevelt prekoncepcióik így minden "vad" ötletüket szabadon kipróbálhatják. Maga a genetikus programozás elve is erre épül - teljesen véletlenül, a logikát kikapcsolva, generál egy kezdeti populációt melynek nagy része halva született ötlet, de vannak köztük a többinél jobbak is, majd a szelekció segítségével ezeket a jobbakat kiválogatja, és egymás keresztezésével pedig fejlesztgeti. A mutáció révén aztán újabb vad ötleteket tud bevinni a rendszerbe. Logika, prekoncepció nem játszik szerepet, csak az a kőkemény

korlát, hogy a teszt során melyik egyed mennyire felel meg az előírt teszt feltételeknek. Tehát, ha 100 amatőr közül egynek van egy jó elgondolása amit megoszt a többiekkel, akik beépítik a saját rendszerükbe (keresztezés) és az így kapott legjobb megoldásokat fejlesztgetve tovább, majd újabb "vad" ötleteket kipróbálva (mutáció) építkeznek akkor talán nagyobb eredményt tudnak elérni mint a pénzelt teamek.

A genetikus programok, algoritmusok a természetben jól működő evolúció durva szimulációja alapján jönnek létre, működnek. Ha az emlősök igen kifinomult, összetett és bonyolult keresztezési folyamatát tekintjük ˙(DNS, bázispárok, géntérkép, hibajavítás, de maga a párválasztás is) akkor nevezhetjük-e keresztezésnek azt az algoritmust amely nemes egyszerűséggel körülbelül a felénél felcserél két kódot? A mutáció a természetben ritka és általában tragédiával végződő folyamat. Nevezhetjük-e mutációnak azt az algoritmust ami a teljes populáció genetikus kódjának 60%-át megváltoztatja?

A fenti mondatokban nem a genetikus algoritmusokkal kapcsolatos kételyeimet akartam megfogalmazni, csak arra szeretném felhívni a figyelmet, hogy az alkalmazott módszer nem az élővilág utánzása, hanem valamilyen optimalizálási feladat számítógépes megoldása az evolúcióból ellesett folyamatokkal.

MELLÉKLET

```
{Készítette:Keveházy Balázs KL/III}
program holland;
uses crt,graph,grplus,makepcx;
var
   g,i,j,N,d,gm,e,k,v,p,L:integer;
   xi:string[8];
   kro:array[1..256] of string[8];
   x:array[1..256] of integer;
   cfgv:array[1..256] of word;
   fitn:array[1..256] of real;
   szum:longint;
   atl,max,min,dh:real;
   ko,poz:integer;
   x0,x1,x2,y1,y2,sr:string;
   b:array[1..8] of byte;
 {-------------------------------------------}
procedure egavga;external; {$L egavga.obj}
{------------grafika bekapcs------------}
 Procedure grstart;
    begin
           gd:=detect;
           initgraph(gd,gm,' ');
           setbkcolor(7);
           screeninit;
           setfillstyle(0,7);
           bar(0,0,640,480);
    end;
 {--- kezdőpopuláció létrehozása------}
procedure kezdopop;
begin
randomize;
          for i:=1 to n do begin
              xi:='';
              for j:=1 to 8 do begin
                  g:=random(2);
                  if g=0 then xi:=xi+chr(48);
                  if g=1 then xi:=xi+chr(49);
              end;
              kro[i]:=xi;
          end;
end;
 {------------------célfüggvény- fitneszfügvény---------}
 procedure vizsgal;
 begin
 szum:=0;
         for i:=1 to n do begin
             for j:=1 to 8 do begin
             xi:=copy(kro[i],j,1);
             val(xi,b[j],ko);
             end;
 x[i]:=b[1]*128+b[2]*64+b[3]*32+b[4]*16+b[5]*8+b[6]*4+b[7]*2+b[8];
 cfgv[i]:=x[i]*x[i];
 szum:=szum+cfgv[i];
 end;
 atl:=szum/n;
             for i:=1 to n do begin
                 fitn[i]:=cfgv[i]/atl;
             end;
```

```
 {---fitneszfüggvény minimuma-maximuma--------}
 min:=fitn[1];max:=fitn[1];
             for i:=2 to n do begin
                 If fitn[i]>max then max:=fitn[i];
                 If fitn[i]<min then min:=fitn[i];
             end;
 dh:=max-min;
 end;
{----------rendezés-----------}
procedure rendez;
var cs2:real;
    mi:integer;
    cs1:string;
begin
     for i:=1 to n-1 do begin
         mi:=i;
         for j:=i+1 to n do begin
             if fitn[mi]<fitn[j] then mi:=j;
     end;
    cs1:=kro[i];kro[i]:=kro[mi];kro[mi]:=cs1;
    cs2:=fitn[i];fitn[i]:=fitn[mi];fitn[mi]:=cs2;
end;
end;
{-------------------------------------}
procedure  rajzol;
begin
{--koordináta tengelyek -----}
setcolor(15);
line(50,400,620,400);
line(50,400,50,30);
line(620,400,617,397);line(620,400,617,403);
line(50,30,47,33);line(50,30,53,33);
writemode(2);
_outstrxy(10,40,'ff');_outstrxy(22,48,'i');
_outstrxy(550,430,'i:0...256');
         for i:=1 to 25 do begin
             line(50+i*20,400,50+i*20,403);
             if i mod 5 =0 then begin
                line(50+i*20,400,50+i*20,405);
                str(i*10,sr);
                writemode(2);
                _outstrxy(40+i*20,410,sr);
                writemode(0);
             end;
         end;
         For i:=0 to 3 do begin
             line(47,400-i*100,50,400-i*100);
             writemode(2);str(i,sr);
             _outstrxy(30,397-i*100,sr);
             writemode(0);
         end;
{ --fitneszértékek grafikonja----}
         for i:=1 to n do begin
             line(50+i*2,400,50+i*2,Round(400-100*fitn[i]));
         end;
         str(dh:6:5,sr);
         writemode(2);
_outstrxy(70,20,'A legjobb és a leggyengébb egyed fitnesze közötti
eltérés:'+sr);
         writemode(0);
end;
```

```
{-----------FOPROGRAM--------------}
Begin
If RegisterBGIdriver(@egavga)<0 then halt;
n:=256;
kezdopop;
grstart;
{ --kezdő populució bitsztringjei---}
i:=10;j:=0;
writemode(2);
for k:=1 to 256 do begin
      _outstrxy(i,j,kro[k]);
      i:=i+75;
      if k mod 8=0 then begin
            i:=10;j:=j+14;
      end;
      _outstrxy(120,460,' A  kezdeti populáció bitsztringjei
<ENTER>');
end;
 writepcx('kezdbit.pcx',getbkcolor);
{---------------------------}
readln;
vizsgal;
cleardevice;
rajzol;
writemode(2);
_outstrxy(140,460,' A  kezdeti populáció fitneszei  <ENTER>');
writemode(0);
writepcx('k0.pcx',getbkcolor);
readln;
rendez;
vizsgal;
writepcx('kr.pcx',getbkcolor);
{-------------Keresztezés mutáció--------------}
k:=0;
while dh> 0.001 do begin
      e:=1;v:=n;
{-----------egypontos keresztezés--------}
      While e<v do begin
          poz:=1+random(26);
          x1:=copy(kro[e],1,poz);
          x2:=copy(kro[e],poz+1,8-poz);
          y1:=copy(kro[e+1],1,poz);
          y2:=copy(kro[e+1],poz+1,8-poz);
          kro[v]:=x1+y2;
          kro[v-1]:=y1+x2;
          e:=e+2;v:=v-2;
       end;
{------mutáció-------------}
p:=1+random(20);
if p=5 then begin
   poz:=1+random(8);
   x1:=copy(kro[n],1,poz-1);
   x0:=copy(kro[n],poz,1);
   if x0='1' then x0:='0' else x0:='1';
   x2:=copy(kro[n],poz+1,8-poz);
   kro[n]:=x1+x0+x2;
 end;
 if p=10 then begin
   poz:=1+random(8);
   x1:=copy(kro[n-1],1,poz-1);
   x0:=copy(kro[n-1],poz,1);
```

```
   if x0='1' then x0:='0' else x0:='1';
   x2:=copy(kro[n-1],poz+1,8-poz);
   kro[n-1]:=x1+x0+x2;
  end;
vizsgal;cleardevice;rajzol;
k:=k+1;str(k,sr);writemode(2);
_outstrxy(200,460,sr+'. populáció fitneszei  <ENTER>');
writemode(0);
writepcx(sr+'fut.pcx',getbkcolor);
rendez;
vizsgal;
readln;
end;
cleardevice;
i:=10;j:=0;
writemode(2);
             for k:=1 to 256 do begin
                 _outstrxy(i,j,kro[k]);
                 i:=i+75;
                 if k mod 8=0 then begin
                     i:=10;j:=j+14;
                  end;
_outstrxy(120,460,'A kilépési feltételnél a bitsztringek
<Vége:ENTER>');
             end;
writepcx('vegso.pcx',getbkcolor);
readln;
closegraph;textmode(80);Clrsr;
end.
```

Felhasznált irodalom, hivatkozások

[SWÁ1] Starkné Werner Ágnes: Mestersége intelligencia, Veszprémi Egyetemi Kiadó, 225, 2004

[SWÁ2] Starkné Werner Ágnes: Mestersége intelligencia, Veszprémi Egyetemi Kiadó, 225-226, 2004

[SWÁ3] Starkné Werner Ágnes: Mestersége intelligencia, Veszprémi Egyetemi Kiadó, 228, 2004

[SWÁ4] Starkné Werner Ágnes: Mestersége intelligencia, Veszprémi Egyetemi Kiadó, 228, 2004

[SWÁ5] Starkné Werner Ágnes: Mestersége intelligencia, Veszprémi Egyetemi Kiadó, 226, 2004

[SWÁ6] Starkné Werner Ágnes: Mestersége intelligencia, Veszprémi Egyetemi Kiadó, 230, 2004

[SWÁ7] Starkné Werner Ágnes: Mestersége intelligencia, Veszprémi Egyetemi Kiadó, 228, 2004

[SWÁ8] Starkné Werner Ágnes: Mestersége intelligencia, Veszprémi Egyetemi Kiadó, 230, 2004

[SWÁ9] Starkné Werner Ágnes: Mestersége intelligencia, Veszprémi Egyetemi Kiadó, 229, 2004

[SWÁ10] Starkné Werner Ágnes: Mestersége intelligencia, Veszprémi Egyetemi Kiadó, 230, 2004

[SWÁ11] Starkné Werner Ágnes: Mestersége intelligencia, Veszprémi Egyetemi Kiadó, 230, 2004

[SWÁ12] Starkné Werner Ágnes: Mestersége intelligencia, Veszprémi Egyetemi Kiadó, 230, 2004

[FI1] Futó Iván: Mesterséges intelligencia, Aula Kiadó, 550-551, 1999

[FI2] Futó Iván: Mesterséges intelligencia, Aula Kiadó, 549, 1999

[FI3] Futó Iván: Mesterséges intelligencia, Aula Kiadó, 554, 1999

[FI4] Futó Iván: Mesterséges intelligencia, Aula Kiadó, 557, 1999

[FI5] Futó Iván: Mesterséges intelligencia, Aula Kiadó, 558, 1999

[ÁGyHV1] Álmos Attila, Győri Sándor, Várkonyiné Kóczy Annamária: Genetikus Algoritmusok: Typotex Kiadó , 21, 2002

[ÁGyHV2] Álmos Attila, Győri Sándor, Várkonyiné Kóczy Annamária: Genetikus Algoritmusok: Typotex Kiadó , 36-40, 2002
[ÁGyHV3] Álmos Attila, Győri Sándor, Várkonyiné Kóczy Annamária: Genetikus Algoritmusok: Typotex Kiadó , 38-46, 2002
[ÁGyHV4] Álmos Attila, Győri Sándor, Várkonyiné Kóczy Annamária: Genetikus Algoritmusok: Typotex Kiadó , 41, 2002
[ÁGyHV5] Álmos Attila, Győri Sándor, Várkonyiné Kóczy Annamária: Genetikus Algoritmusok: Typotex Kiadó , 41-42, 2002
[ÁGyHV6] Álmos Attila, Győri Sándor, Várkonyiné Kóczy Annamária: Genetikus Algoritmusok: Typotex Kiadó , 42-43, 2002
[ÁGyHV7] Álmos Attila, Győri Sándor, Várkonyiné Kóczy Annamária: Genetikus Algoritmusok: Typotex Kiadó , 45, 2002
[ÁGyHV8] Álmos Attila, Győri Sándor, Várkonyiné Kóczy Annamária: Genetikus Algoritmusok: Typotex Kiadó , 46, 2002
[ÁGyHV9] Álmos Attila, Győri Sándor, Várkonyiné Kóczy Annamária: Genetikus Algoritmusok: Typotex Kiadó , 45, 2002
[ÁGyHV10] Álmos Attila, Győri Sándor, Várkonyiné Kóczy Annamária: Genetikus Algoritmusok: Typotex Kiadó , 34-35, 2002
[ÁGyHV11] Álmos Attila, Győri Sándor, Várkonyiné Kóczy Annamária: Genetikus Algoritmusok: Typotex Kiadó , 198, 2002
[ÁGyHV12] Álmos Attila, Győri Sándor, Várkonyiné Kóczy Annamária: Genetikus Algoritmusok: Typotex Kiadó , 199, 2002
[ÁGyHV13] Álmos Attila, Győri Sándor, Várkonyiné Kóczy Annamária: Genetikus Algoritmusok: Typotex Kiadó , 200, 2002
[ÁGyHV14] Álmos Attila, Győri Sándor, Várkonyiné Kóczy Annamária: Genetikus Algoritmusok: Typotex Kiadó , 206, 2002