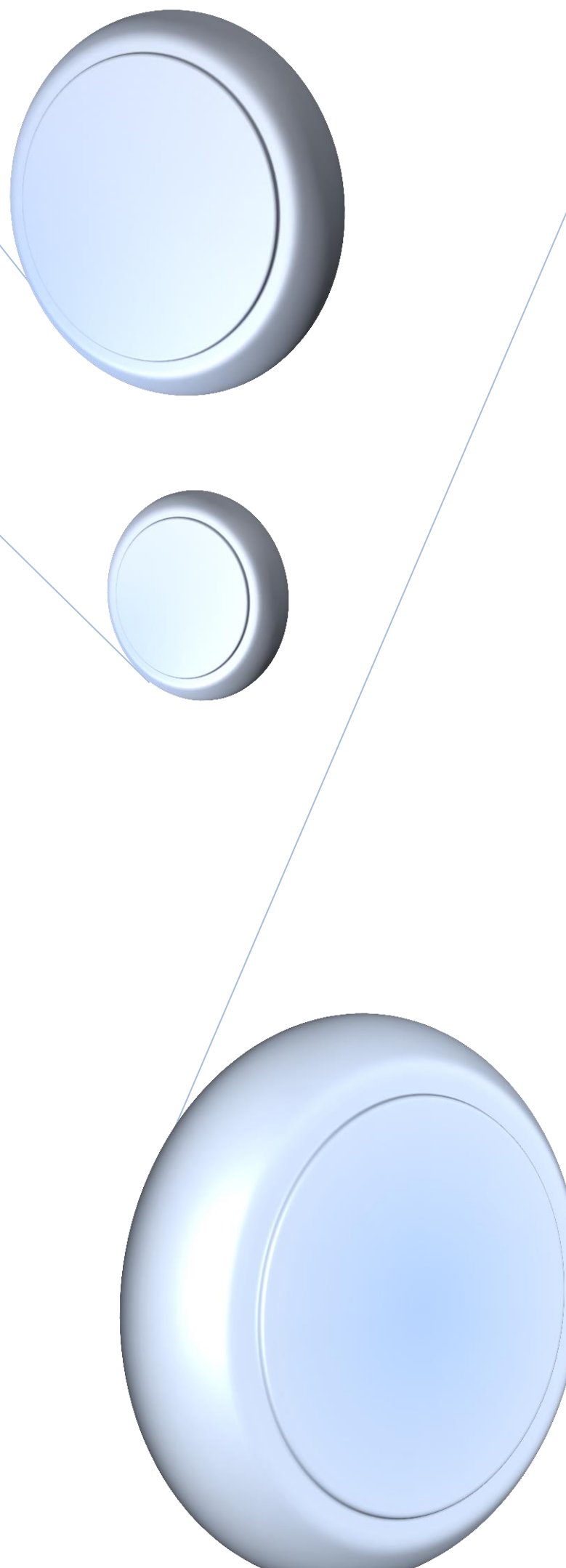




Metaheurisztikus algoritmusok hatékonyságvizsgálata adott karaktersorozat előállításában

Tudományos Diákköri Konferencia

Budapesti Corvinus Egyetem
Gazdálkodástudományi Kar
Számítástudományi Tanszék

Konzulens: Dr. Láng Blanka

adjunktus

Kovács László

Alapképzés

II. évfolyam

2013. március 18.

Tartalomjegyzék

1. Bevezetés.....	3
2. Metaheurisztikus algoritmusokról általában.....	3
3. Probléma bemutatása	7
4. Genetikus algoritmus	9
4.1. Populáció mérete	14
4.2. Mutációs valószínűség.....	14
4.3. Genetikus algoritmus alkalmazásának összegzése.....	17
5. Ősrobbanás (Big Bang-Big Crunch).....	18
6. A harmónia kereső algoritmus	22
6.1. HMS	27
6.2. PAR valószínűség	28
6.3. HMCR.....	28
6.4. A többemű memóriájú harmónia kereső algoritmus összefoglalása	29
7. A módosított harmónia keresés	30
7.1. HMCR.....	32
7.2. PAR	33
7.3. A módosított harmónia keresés alkalmazásának összefoglalása	33
8. Sztochasztikus hegymászó.....	34
9. Szimulált hűtés	38
9.1. Kezdeti hőmérséklet.....	41
9.2. Hőmérsékleten tartózkodás ideje	42
10. Sztochasztikus hegymászó és Szimulált hűtés.....	43
11. Összegzés.....	44
Mellékletek.....	48
Irodalomjegyzék	50

1. Bevezetés

Dolgozatunkban egy szándékosan egyszerű probléma (adott karaktersorozat előállítása) különböző metaheurisztikus megoldásai kerülnek bemutatásra. Mindez abból célból történik, hogy megmutassuk a polinomiális idő alatt nem megoldható, tehát NP-nehez kombinatorikus optimalizálási problémák megoldásában egyre népszerűbb metaheurisztikus algoritmusok azon sajátos tulajdonságát, hogy már a megfelelő algoritmus megtalálása, és a feladatra hangolása is alapos hatékonyságvizsgálatot igényel, már egy ilyen egyszerű probléma esetében is.

A dolgozatban bemutatjuk a választott probléma természetét, jellemzőit, valamint hasonlóságát a genetikus programozási feladattal. Ezután különböző metaheurisztikus megoldásokat alkalmazunk a feladatra. Ezzel alapvetően nem a feladat megoldása lesz a célunk, hanem annak vizsgálata, hogy a választott metaheurisztika milyen módon illeszthető az adott problémánkhoz. A dolgozat tárgya tehát maga az illesztés folyamata, és annak komplexitásának bemutatása, az analógia, az egyedek és az operátorok megfelelő kiválasztása. Bemutatjuk, hogy ez a komplexitás metaheurisztikák „meta” jellegéből adódik, azaz kétszintű voltukból. A dolgozat során belátjuk, hogy metaheurisztika megfelelő beállítása nagymértékben növeli vagy csökkenti az algoritmus hatékonyságát, sőt képes a probléma megoldhatóságára is hatást gyakorolni.

2. Metaheurisztikus algoritmusokról általában

A metaheurisztikák tudományterületének létrejötte és gyors fejlődése a tudományban és az iparban egyaránt egyre fontosabbá váló kombinatorikai optimalizálási problémának köszönhető.

Egy $P = (S, f)$ kombinatorikus optimalizálás esetén adott:

- A változók halmaza: $X = \{x_1; \dots; x_n\}$
- Az értékkészletek: $D_1; \dots; D_n$
- A változók között fennálló összefüggések
- Egy $f: D_1 \times \dots \times D_n \rightarrow R^+$ célfüggvény, amit minimalizálunk vagy maximalizálunk.
- A keresési tér:

$$S = \{s = \{(x_1; v_1); \dots; (x_n; v_n)\} \mid v_i \in D_i; s \text{ kielégíti a fenti korlátokat}\}$$

Ekkor keressük azon $s^* \in S$ értékeket, melyekre a célfüggvény f értéke minimális vagy éppen maximális. Az s^* értékeket állapotoknak nevezzük, globális optimumnak pedig azt az s^* értéket hívjuk, melyre a célfüggvény értéke maximális vagy minimális.

Ilyen kombinatorikus optimalizálási problémák több gyakorlati jelentőségű feladatot is magukba foglalnak, mint például:

- utazó ügynök probléma
- kínaipostás-probléma
- gráfszínezés
- diszkrét esetű ütemezési problémák

Ezek többsége NP-nehézségű probléma, azaz nincs olyan algoritmus, amely polinomiális idő alatt megoldja. Éppen ezért csak egy „elfogadható mértékben optimális” megoldást keresünk S -ben.

Az NP-nehézség és a problémák alapvetően bonyolult matematikai modellezhetőségéből adódóan az ilyen feladatok megoldására a kutatók heurisztikus algoritmusokat alkalmaznak. Az utóbbi 20 évben egy új fejezet nyílt a heurisztikus megoldások történetében. A kombinatorikus optimalizálási NP-nehézségű problémák megoldása során sokszor hatékonyabbnak bizonyult, ha az alapvető heurisztikákat egy magasabb szintű keretbe foglalják, amely keret vezérel a különböző heurisztikus keresési eljárásokat, amik így hatékonyabban járják be a keresési teret. Az ilyen módon keletkezett algoritmusokat nevezték el metaheurisztikáknak.

Mielőtt a metaheurisztikus algoritmusok konkrét definiálását elvégezzük, tekintsük át azon alapvető fogalmakat, melyeket a metaheurisztikák használnak.

Populációnak nevezzük azon különböző s^* állapotok halmazát, melyekkel a metaheurisztika dolgozik. A populációk lehetnek egy vagy éppen többbeleműek. A többbelemű populációk szimultán képesek kezelni több megoldásvektort. A populációk az algoritmus minden lépésében felülíródnak. Az egyes s^* állapotokat hívjuk *egyednek*. Az *állapotátmenet* során az adott állapot egy variánsát állítjuk elő. A lépés egyaránt növelheti vagy csökkentheti a célfüggvény értékét. Az állapotátmenet során előállított új egyed(eket) nevezzük *utódnak*.

A metaheurisztikus algoritmusok alapvetően háromféle operátorral dolgozhatnak futásuk során. Most csak egy rövid áttekintést adunk róluk, részletesebb ismertetésükre a mind a

hármát alkalmazó genetikus algoritmus bemutatása során kerül sor. A *mutációs* operátor egy megoldásból (állapotból vagy egyedből) állít elő újabbat, ami az eredeti egy variánsa. Legalább kettő megoldás kombinálásából állít elő új megoldást a *rekombinációs* operátor. A *szelekció* során pedig egy adott darabszámú megoldás kerül kiválasztásra, melyek átkerülnek az új populációba.

Mivel a lehetséges megoldások halmaza igen nagy, a metaheurisztikák futását egy időkorláthoz vagy maximális lépésszámhoz szokás kötni. A metaheurisztikák tulajdonképpen kétszintű heurisztikákként is fölfoghatók. Az alsó szint jelképezi a probléma azonosítását, megfogalmazását, az input és az output meghatározását. A „meta” szint jelenti a heurisztikákat vezérlő elveket. Ezek az elvek szinte minden esetben természeti jelenségek analógiájára épülnek (pl.: evolúció, fizikai lehűtési eljárás, fekete lyuk keletkezése). Ezen a szinten döntjük el, hogy melyik analógiát alkalmazzuk, majd meg kell határoznunk az egyedeket, illetve kiválasztjuk az operátorokat is. Ezt a kétszintűséget alkalmazva az eljárások tulajdonképpen a véletlenszerűséget egyesítik a szabályszerűséggel. Ezen gondolatok alapján a metaheurisztikák koncepcióját a következő ábrával lehet szemléltetni:



1. ábra: Kétszintű heurisztikák

Forrás: Láng Blanka, 2010

Ha áttekintjük a metaheurisztikák definícióit megfigyelhetjük, hogy azok nem egységesek, elsősorban a metaheurisztikák keletkezéséből adódó alapkoncepciókban egyeznek meg.

A különböző definíciók közül a legelterjedtebbek a következők:

„A metaheurisztika egy iteratív vezérlő eljárás, amely az alsóbb szintű heurisztikák műveleteit vezérli, és módosítja azokat, hogy hatékony, magas minőségű megoldást állítson elő. Dolgozhat egyedi megoldással, vagy a megoldások egy halmazával minden iterációs lépésben. Az alsóbb szintű heurisztikák lehetnek magas (vagy alacsony) szintű eljárások, vagy egyszerű lokális kereső eljárások, vagy épp egy konstrukciós eljárás.” (Volf, Martello, Osman, Roucairol: Meta-Heuristics – Advances and Trends in Local Search Paradigms for Optimization, 1999, 8. oldal)

„A metaheurisztika formálisan definiálható egy iteratív generációs eljárásaként, amely alsóbb szintű heurisztikákat vezérel, különböző elképzelések intelligens kombinálásával, a keresési tér feltérképezéséhez és hasznosításához. A tanuló stratégiákat az információ strukturálásához használjuk, annak érdekében hogy hatékonyan találjunk optimális közeli megoldást.” (Osman, Laporte: Metaheuristics: A bibliography. Annals of Operations Research, 63 (5), 1996, 513-514. oldal)

„Magas szintű algoritmikus keret, vagy szemléletmód, mely optimalizációs problémákra specializálódott.” (National Institute of Standards and Technology, <http://www.nist.gov/>)

A fenti definíciók nem tekinthetők egységesnek, ám az alapvető tulajdonságok tekintetében átfedést mutatnak. A közös jellemzők Láng B. (2010) alapján a következők:

- „A metaheurisztikák olyan stratégiák, melyek a kereső eljárásokat vezérlik.
- Céljuk a keresési teret hatékonyan feltérképezni a közel optimális megoldás megtalálása érdekében.
- A metaheurisztikus algoritmusokat alkotó technikák az egyszerű kereső eljárásoktól a komplex tanuló eljárásokig terjednek.
- A metaheurisztikus eljárások közelítőek, és rendszerint nem determinisztikusak.
- A metaheurisztika akkor is használható, ha a feladat struktúrája kevésbé ismert, mivel nem igényel területfüggő tudást, tehát használhatjuk abban az esetben is, ha nem tudjuk a problémát leírni, de a lehetséges megoldások minőségét valamilyen szempont szerint értékelni, rangsorolni tudjuk.
- A metaheurisztika alapcélkitűzéseit tekintve globális optimumot keres, ezért tartalmaz olyan mechanizmusokat, melyek megóvnak attól, hogy a keresési térnek csupán egy elzárt területét térképezzük fel, így az esetek jelentős részében globális maximumhoz

(minimumhoz) közeli értéket adni megoldásként a kezdőpont választásától függetlenül, ellentétben korábban használt algoritmusokkal, melyek gyakran lokális optimumot adtak eredményül.

- A metaheurisztikák a felsőbb szintű stratégia által kontrollált heurisztikákon keresztül hasznosíthatnak területfüggő tudást.
- A fejlettebb metaheurisztikák a keresési eljárás alatt a korábbi lépések során szerzett tapasztalatokat használják a további lépések vezérléséhez.”

A metaheurisztikák alapvető tulajdonságainak és jellemzőinek ismertetése után a dolgozat metaheurisztikus algoritmusokkal megoldandó problémájának bemutatása következik. Majd a különböző algoritmusok jellemzéseire és a dolgozat feladatán történő alkalmazásának hatékonyságvizsgálata kerül sor.

3. Probléma bemutatása

Az előző fejezetben már általánosságban ismertetett metaheurisztikus algoritmusokat a dolgozatban egy szándékoltan egyszerű problémára kívánjuk alkalmazni, ami nem más, mint egy adott karaktersorozat előállítása. Segítségével megmutatható, hogy a metaheurisztikák alkalmazása, elsősorban nem determinisztikus viselkedésük miatt alapos kutatást, és körütekintést igényel már ilyen, nem komplex feladat esetén is. Alkalmazásukat tovább bonyolítja meta voltuk, azaz, hogy a választott analógiát megfelelően alkalmazzuk a feladatainkra, hogy biztosan jól irányítsák a keresési eljárást. A dolgozat tehát szándékosan egy egyszerű probléma lehetséges metaheurisztikus megoldásait vizsgálja, mivel nem a probléma megoldása az elsődleges cél, hanem annak vizsgálata, hogyan illeszthető egy adott feladathoz egy metaheurisztikus megoldás. A vizsgálat tárgya tehát maga az illesztési folyamat.

A választott feladat természete lehetővé teszi, hogy kiemelje a metaheurisztikák azon tulajdonságát, hogy hatékonyan alkalmazhatók matematikailag nem jól definiálható (strukturálható) optimalizálási problémákra is. Egész egyszerűen csak a különböző megoldásokról kell tudnunk egyértelműen eldöntenünk, hogy jobbak-e, mint más alternatív megoldások, nem szükséges komoly matematikai modell építése. Esetünkben ez egyszerűen meghatározható, hiszen ha a karaktereket egy megfelelő kódolási technikával pl. ASCII kóddal reprezentáljuk, akkor mérni tudjuk az adott karakter távolságát a keresett optimumtól (az adott karaktersorozat megfelelő karakterétől), egy egyszerű különbségképzéssel. A karaktersorozat távolsága a keresett szövegtől pedig ezen karakterszintű különbségek abszolút

értékeinek összege. A továbbiakban ezt a távolságot nevezzük a lehetséges megoldás fitness értékének, mivel megmutatja, hogy a lehetséges megoldás mennyire „fitt” vagyis rátermett az optimumhoz képest. Minél kisebb ez a számérték, annál fittebb a megoldás. Tehát egyértelműen meg tudjuk állapítani egy alternatíváról, hogy jobb-e, mint más lehetséges megoldás. Ezen értékek kiszámítása tulajdonképpen egyszerű zérus hely számítás is lehetne, hiszen (kissé paradox módon) a 0 fitness értékű megoldás az optimum, ami pontosan ugyan ott helyezkedik el a keresési térben, mint az előállítandó karaktersorozat, szöveg. Viszont, ha az eljárást automatizálni kívánjuk bármilyen szövegre, olyan módon, hogy egy számítógépen futó szövegmegjelenítő program a lehetőségeket felmérve, bármiféle beavatkozás nélkül magától álljon be a kívánt megoldásra, mindenképpen optimumkeresési eljáráshoz kell nyúlni. A teljesen véletlen lineáris keresési eljárás (haladunk előre a keresési térben, míg optimumot nem találunk) viszont ilyen nagy keresési térben (csak az ékezet nélküli kis-és nagy betűk, valamint az alapvető írásjelek figyelembe vételével a keresési tér: 95^n , ahol n a keresett szöveg hossza) nem lesz célravezető:



```

8044. lépés: !Kd{K#aixEvcFitnessz: 272
8045. lépés: yJ6,J4%zR=R,Fitnessz: 420
8046. lépés: ;)Z5S<Gd U-Fitnessz: 460
8047. lépés: RW^H"5n)CJ]IFitnessz: 374
8048. lépés: AZN1xsxbxX{/Fitnessz: 303
8049. lépés: x&SE)N}xt(XrFitnessz: 501
8050. lépés: T\w<NTgT^<9YFitnessz: 375
8051. lépés: %+Sk,q4xZNOzFitnessz: 475
8052. lépés: "ltKbw8"Z[7Fitnessz: 386
8053. lépés: !V4DoFP]x?IFitnessz: 361
8054. lépés: z2!"s^7%,{Z$Fitnessz: 445
8055. lépés: _r){D WINjTFitnessz: 341
8056. lépés: }o)4}r.g)vx9Fitnessz: 426
8057. lépés: utDBCu+${/dlFitnessz: 519
8058. lépés: =T7kAuN_G7D$Fitnessz: 369

```

2. ábra: A véletlen keresés futása

Ezért mindenképpen irányítani kell a keresési folyamatot, ám a véletlen megoldásokat sem hanyagolhatjuk el teljesen, hiszen a túlzottan irányított keresés könnyen beragadhat egy lokális optimumban (pl.: 2 fitness értékű megoldás), mivel az adott irányítási szisztéma adott időn belül nem talál jobb karakterkombinációt. Azaz a véletlen megoldásokra is szükség van, hogy biztosítsák a megoldási alternatívák kellő változatosságát. Ezt a kétféle megközelítést pedig a metaheurisztikus algoritmusok ötvözik, és alkalmazzák szimultán módon.

A fentiekből adódik az, hogy a neurális hálók és egyéb mesterséges intelligenciák tanításában a metaheurisztikus eljárások kulcsszerephez jutnak. Hiszen a tanításuk problémájának természete hasonlít az előbb ismertetett automatikus szövegkereséshez. Adott egy

megoldandó feladat, melynek megoldásainak jóságát értékelni tudjuk egy fitnessz értékkel, és az intelligencia feladata, hogy egy optimális vagy éppen elfogadhatóan optimális megoldást, számítógépprogramot találjon, építsen ki a lehetőségeit, szubrutinjait mérlegelve, bármiféle egyéb segítség nélkül. Ezt az alapgondolatot először John R. Koza ismerte fel 1992-es tanulmányában, melynek címe, Genetikus Programozás (Koza: Genetic Programming, 1992).

Így, tehát még az adott karaktersorozat előállítása is felfogható egy durván primitív mesterséges intelligencia tanítási feladatának, melyben az egyes karakterek a szubrutinok. A metaheurisztikák hatékonyságvizsgálatának eredménye tehát egy ilyen természetű egyszerű problémán nem közömbös, irányt adhat bonyolultabb problémákat megoldó rendszerek tervezéséhez is.

A hatékonyságvizsgálatot először az algoritmusok saját paramétereinek függvényében fogjuk elvégezni, az optimum eléréséhez szükséges lépésszámot használva a hatékonyság mértékegységeként. Ezek után az egyes metaheurisztikákat a legjobb eredményt nyújtó paraméterezésükkel véve vizsgáljuk, egymással összevetve, másodpercben mért futásidőt véve alapul az összehasonlításhoz.

4. Genetikus algoritmus

Tanulmányában Koza egy speciális genetikus algoritmust dolgozott ki adott problémákat megoldó programok automatikus kifejlesztésére. Továbbá észrevehetjük, hogy az aktuális feladat (adott karaktersorozat előállítása) megoldására is ezt az algoritmust szokás a leggyakrabban felhasználni. Valamint általánosságban is ez a kutatók körében legnépszerűbb eljárás. Ezért először mi is ezt a metaheurisztikát kívánjuk megvizsgálni.

A genetikus algoritmust 1975-ben Holland fejlesztette ki, majd 1989-ben Goldberg továbbfejlesztette. Az algoritmus alapgondolata a biológiai genetikán és Darwin „természetes kiválasztódás” elvén alapul. Az egyedek az egyes megoldások, melyeket többemű populációba szervez, ezzel biztosítja, hogy a keresési tér minél nagyobb részét járjuk be. Ráadásul képes egy, az egyed alatti szintet is definiálni, a gének szintjét, amikből az egyedek felépülnek, ezzel kiváló lehetőséget biztosítva a megoldások (egyedek) finom, aprólékos változtatására, azaz a keresési tér még pontosabb bejárására. A géneket általában bitsorozattal ábrázoljuk, mint a legáltalánosabb megoldás, de lehetséges valós értékeket is alapul venni, megoldásunkban mi is ezt tesszük, az ACSII kódolás ezt megengedi. Ebből is látszik, hogy a metaheurisztikák alapvetően az egyszerűsége törekednek, hogy minél több, változatosabb

problémára alkalmazhatók legyenek. A három szint (populáció, egyed, gén) megválasztása után az algoritmus minden lépésben egy új populációt hoz létre a fitness értékek alapján (maga a fitness elnevezés is ebből az algoritmusból ered, hiszen Darwin szerint: „**Survival of the fittest**”), így folyamatosan halad az optimális megoldás (egyed) felé a populációnk. Ezt a folyamatot három ún. genetikai operátor képviseli: szelekció, rekombináció, mutáció.

A szelekciós eljárás segítségével választjuk ki az egyes populáció legjobb egyedeit, melyeket továbbviszünk a következő populációba, és amelyek a rekombinációs eljárás bemenetei lehetnek. Alapvetően három szelekciós operátor közül választhatunk:

- **Rulett kerék:** A kiválasztás valószínűsége a fitness értékkel arányos, olyan módon, hogy egy megoldás kiválasztásának valószínűsége annál nagyobb, minél nagyobb a rátermettsége a populáció rátermettségi átlagához képest. A kiválasztás úgy működik, mintha az egyedeket egy olyan rulett kerék egy-egy mélyedéséhez (cikkelyéhez) rendelnénk, ahol a mélyedés nagysága arányos az egyed rátermettségével.
- **Verseny szelekció:** Egy csoportot választunk ki a populációból és a fitness értékük alapján versenyeztetjük őket. Az optimumhoz közelebb eső fitness értékű egyed győz.
- **Elitista:** Az eddig kapott legjobb $\varphi * m$ db egyedet mindig áthelyezi az új populációba, ahol m a populáció mérete, φ pedig az elitek aránya, egy $(0; 1)$ közötti érték.

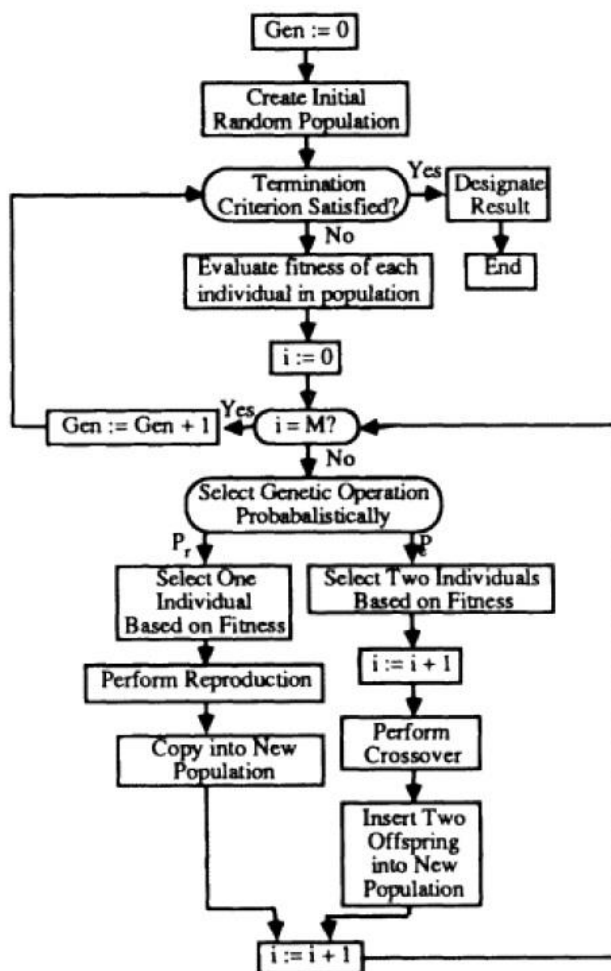
A rekombinációs operátor két egyedből hoz létre új egyedet. A biológiában ez egy biológiai utód létrehozásának felel meg. Egyik leggyakoribb kombináció az egyponthoz keresztezés. Ennek során a műveletben résztvevő egyedeket egy véletlenszerűen kiválasztott génnél elvágjuk. A vágás előtti géntulajdonságokat örökli az új egyed az egyik szülőtől, a vágás utániakat a másik szülőtől.

A másik gyakori kombináció az egyenletes keresztezés. Ez esetben az utód az egyes géneit adott ω valószínűséggel örökli az egyik szülőtől, és $1 - \omega$ valószínűséggel a másik szülőtől. (ω egy 0 és 1 közé eső valószínűség érték).

A mutációs operátor, egyetlen megoldást igényel, és ebből állít elő egy újat, az eredeti egy variánsát, mutációját. A mutáció célja a populáció frissítése. Ezzel a művelettel biztosítható, hogy egy eljárás képes legyen kitörni a lokális megoldás környezetéből. A mutációt általában

úgy oldják meg, hogy rendszerint az egyed (megoldás) egy véletlen kiválasztott génje változik meg.

Koza algoritmus a Genetikus programozási probléma megoldására többemű populációt használ, melyben az egyedek a problémát megoldó programok, a gének a program szubrutinjai. A következő populációba kerülés egy a program fitness értéke alapján meghatározott valószínűség szerint történik. Ha ez a valószínűség kellően magas, az egyed átkerül a következő populációba. A maradék helyekre pedig a szubrutin szinten működő keresztezéssel előállított utód-programok kerülnek. Tulajdonképpen egy Rulett-kerék elvet alkalmaz, egy/többpontos keresztezéssel, mutáció nélkül. Azon kívül, hogy a mutációt szubrutinok esetében körülményes lehet megoldani, hatékonyságvizsgálatunkból kiderül, hogy nem is feltétlen célszerű Rulett kerék szelekció esetén a mutációs operátort alkalmazni.



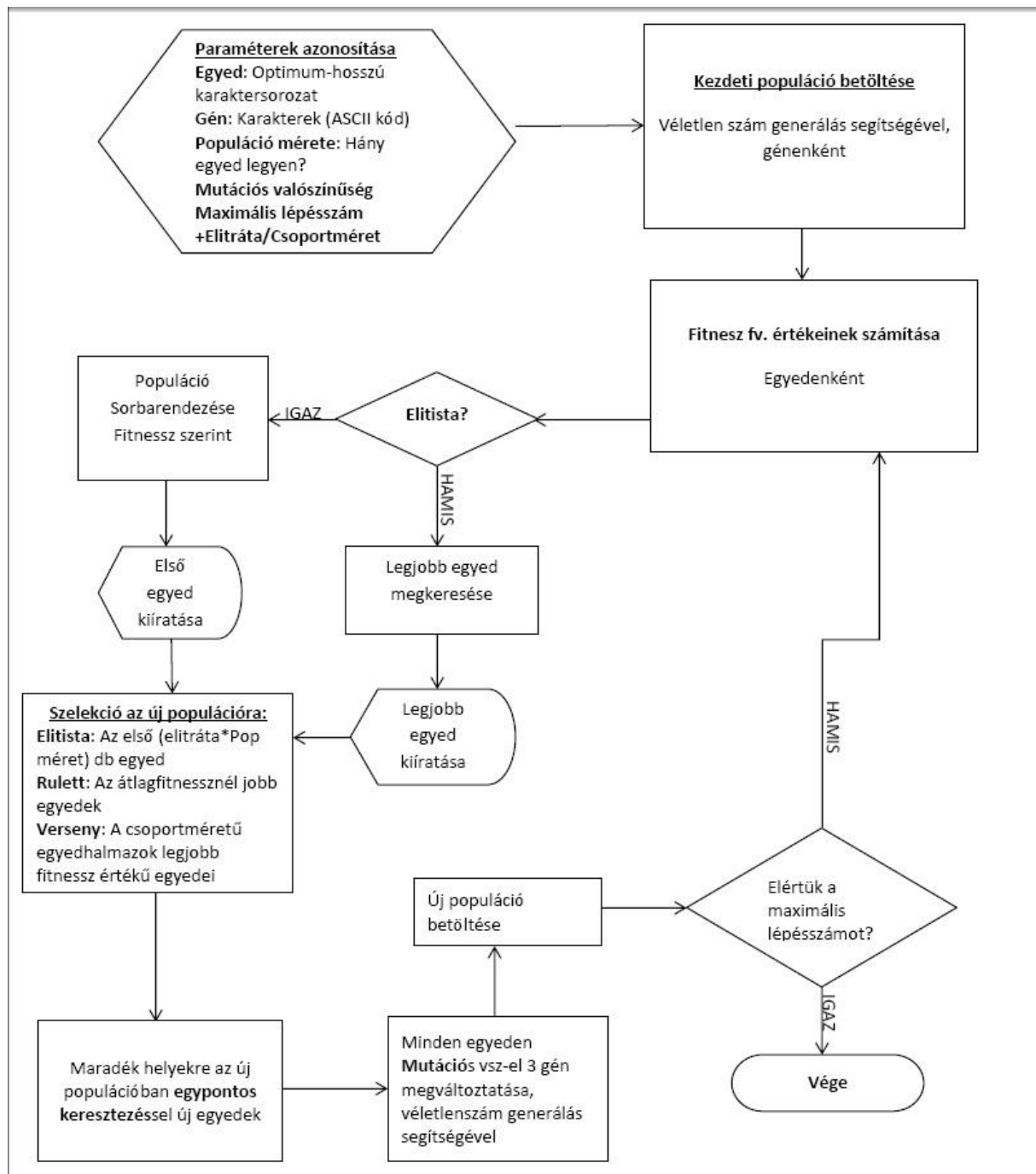
3. ábra: A genetikus programozás folyamatábrája

Forrás: Koza, 1992

Ha a genetikus algoritmust a választott problémára, adott karaktersorozat előállítására kívánjuk alkalmazni, igen könnyű dolgunk van. Nagyon egyszerűen tudjuk azonosítani a biológiai analógia elemeit. Gének az egyes karakterek ASCII kódjai, valós értékkel ábrázolva, az egyedek pedig a különböző lehetséges karaktersorozatok, melyeket egy adott elemszámú populációba rendezünk.

Rekombinációs eljárásnak egy pontos keresztezést választotjuk, mivel ezen karaktersorozatok jobb esetben értelmes szövegek, így hatékonyabb, ha csak egy keresztezési pont mentén bontjuk meg az egyedeinket, hogy a már esetleg meglévő „értelmes” szövegrészek ne essenek szét. Ha egyenletes keresztezéssel dolgozunk, akkor génenként ki vagyunk szolgáltatva a ω valószínűségnek, ami módosíthatja az egyes géneket, ezzel megbontva az esetleg már kialakulóban lévő vagy éppen kialakult, értelmesnek mondható szövegrészt.

Az így kialakított algoritmust a következő folyamatábra mutatja be:

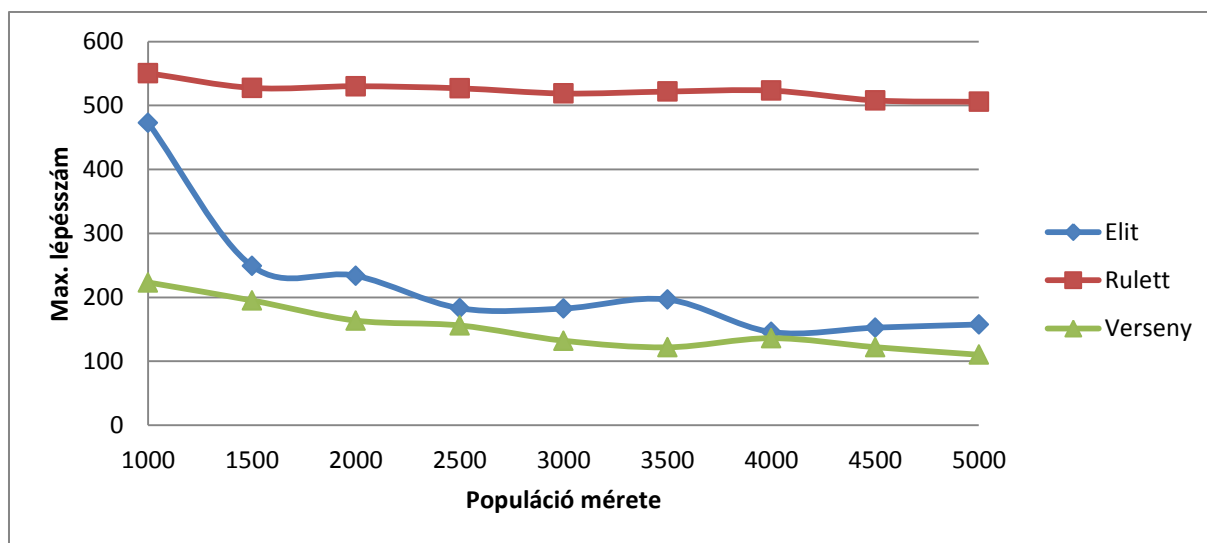


4. ábra: A genetikus algoritmus alkalmazása a dolgozat problémájára

A hatékonyság szempontjából megvizsgálandó paraméterek tehát a következők: populáció mérete, mutációs valószínűség, és szelekciótól függően: elitráta és a versenyző csoportok mérete. A hatékonyság mértékegysége pedig az optimum megtalálásához szükséges lépés/generációs szám. A futásidő az algoritmuson belüli paraméterváltoztatások esetén a lépésszámmal egyenes arányban nő.

4.1. Populáció mérete

A paraméter vizsgálata során (többi paraméter rögzített, egy paraméter értéke 20 futási eredmény átlaga) azt tapasztaljuk, hogy ha a populáció méretét növeljük mind a három szelekciós stratégia esetén, akkor a befejezéshez szükséges lépésszám lecsökken. Ezt a két ismérv átlagos korrelációja is alátámasztja: $-0,875$ -ös értéket ad, azaz a lépésszám csökkenését 76,6%-ban befolyásolja a populáció méretének növelése. Az eredményeket az alábbi grafikon is szemlélteti:



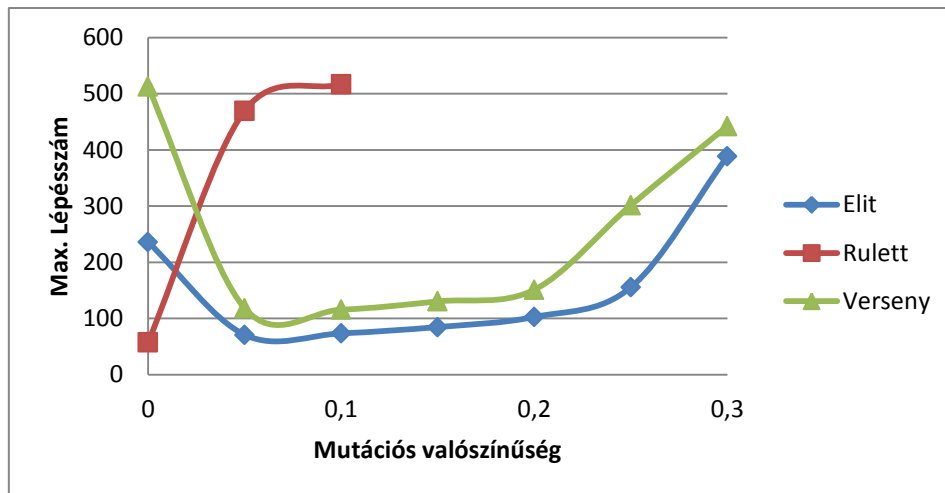
5. ábra: A populáció méretének változtatásának eredményei

Megfigyelhető az ábra alapján, hogy a Rulett kerék szelekción nem sokat javít a populáció méretének növelése. Ennél a szelekciónál egy másik paraméter, kizárásos alapon a mutációs valószínűség, nagyobb befolyással bír.

Általánosságban viszont elmondható, hogy minél nagyobb a populáció mérete, egy lépésben annál nagyobb részét tudjuk megvizsgálni a keresési térnek, így kevesebb lépésből vagyunk képesek optimumot találni.

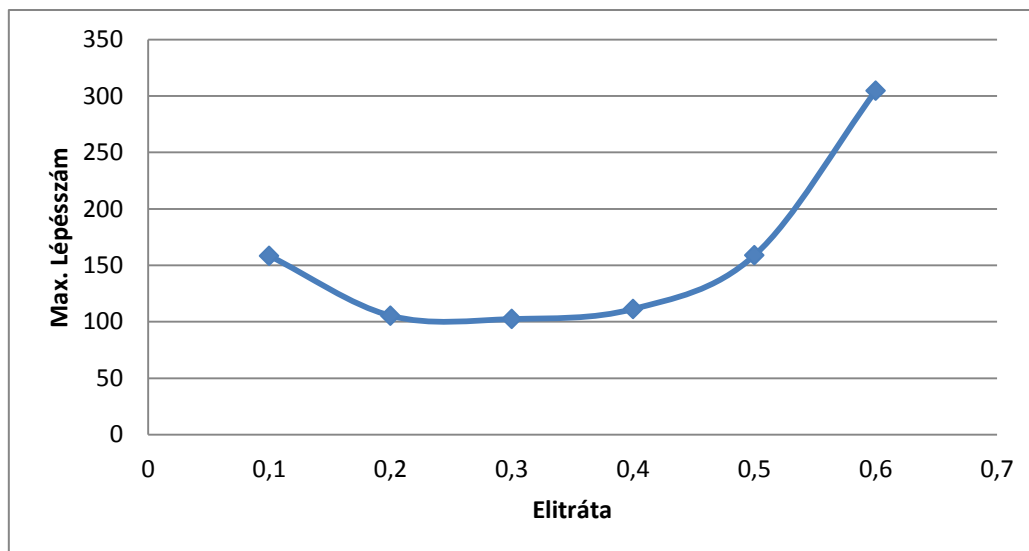
4.2. Mutációs valószínűség

A paraméter vizsgálatakor a következő grafikont alkotják az eredmények, a populáció méretének áttekintésekor alkalmazott feltételeket, eszközöket alkalmazva:

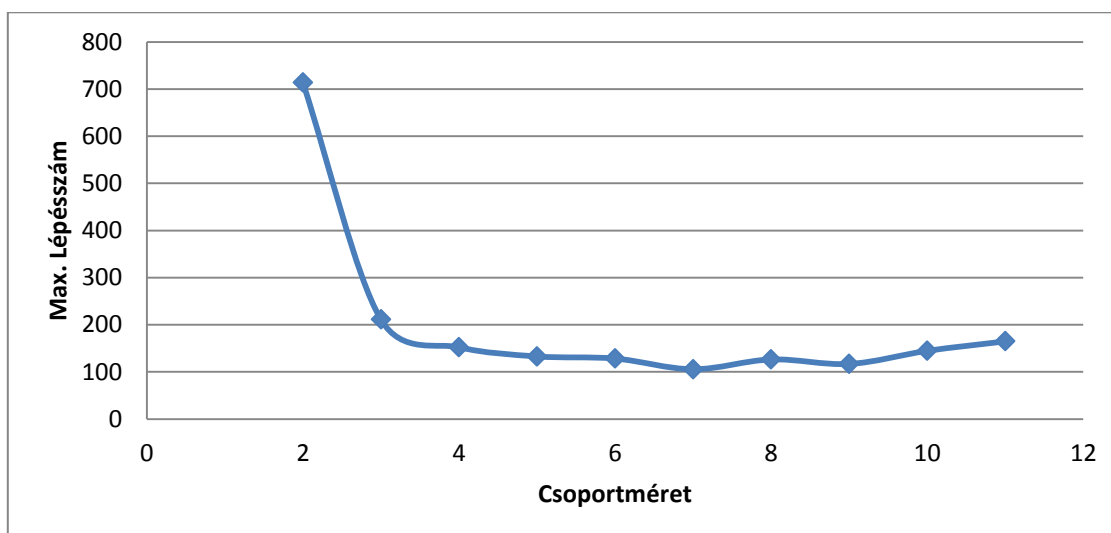


6. ábra: A mutációs valószínűség változtatásának eredményei

Ahogy sejtettük, a mutációs valószínűség nagyban befolyásolja a Rulett-elv hatékonyságát. A korrelációs együtthatója is kimagasló, 0,909-es értéket mutat, míg a másik kettő szelekciós stratégia eredménye csupán 0,431 (Elitista) és 0,088 (Verseny). Érdekes módon az szolgáltatja a legjobb hatékonyságot a Rulett kerék esetén, ha a mutációs operátort elhagyjuk, valószínűsége 0. Ennek okát az evolúció filozófiájában érdemes keresni. Ugyanis Darwin szerint a szelekció, a természetes kiválasztódás az átlagosnál jobb fittséggel bíró egyedeket hagyja életben. Az alkalmazott Rulett kerék elv direkt módon ezt a fajta gondolkodást testesíti meg, hiszen az átlagos fitness értéknél jobb egyedeket engedjük át a következő populációba. Míg a másik két szelekció csak közvetve, különböző paraméterek (elitráta, csoportméret) megfelelő szabályozásával próbálja elérni ugyan ezt az eredményt, azaz az átlagosnál jobb elemek kiválasztását az új populációba.



7. ábra: Az elitráta változtatásának eredményei



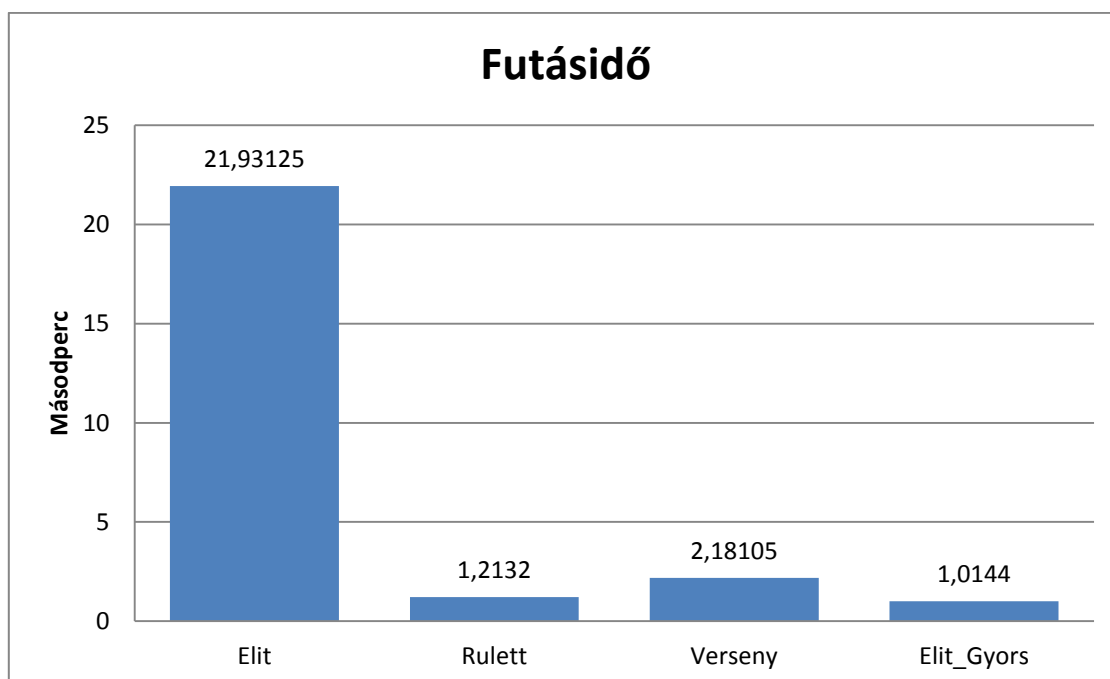
8. ábra: A csoportméret változtatásának eredményei

A fenti két grafikonon látszik, hogy ha a másik két szelekciós stratégia paramétereit egy megfelelő értékre állítjuk, akkor kapunk egy a többi esetnél kisebb lépésszámot. Ekkor sikerült eltalálni nagyjából azt, hogy az átlagosnál jobb egyedeket válogassunk az új populációba. De még ekkor sem biztos, hogy a paraméterek pontosan az átlagnál rátermettebb egyedeket adják vissza, így egy kis rásegítést igényelnek, a mutációs operátor pedig pont megfelel a célnak. Látszik, hogy ezt sem kell túlzásba vinni, 5-10%-os mutációs valószínűség a leghatékonyabb, amint a grafikonon is megfigyelhető.

4.3. Genetikus algoritmus alkalmazásának összegzése

A szelekciós stratégiák közül tehát az előző részben említett szempontok miatt a Rulett kerék elv a legcélravezetőbb a problémánkra. Koza is ezt alkalmazza a Genetikus programozás feladatára. Ez a jellemző is mutatja, hogy a két probléma természete valamennyire hasonlít.

Ezekén kívül még érdemes lehet megvizsgálni a különböző szelekciós stratégiákat a legjobb paraméterezésükkel, egymással összehasonlítva. Itt, a többi metaheurisztikával való összehasonlítás érdekében, egy jobb egységességet biztosító mértékegységet, a futásidőt másodpercekben mérve alkalmazzuk.



9. ábra: A genetikus algoritmus futásideje

Mielőtt értelmezzük az eredményeket, szót kell ejtenünk, arról, ami már az algoritmus folyamatábráján is látszik: az Elitista szelekció alkalmazásakor kénytelen vagyunk egyszerű minimum-keresés helyett költséges rendező eljárásokba belemenni, hogy az elitrátát használva, az első X helyen valóban a legjobb X egyed is legyen. Bizonyítottan az összehasonlító rendezések futásidejének alsó korlátja egy n elemű tömb esetén: $n \cdot \log_2(n)$. Viszont a legtöbb rendezés esetében ez az idő n^2 -es érték lehet. Az először alkalmazott buborékrendezésnél ez az érték az alsó korlát is, ebből adódik a majdnem fél perces futásidő. Ám Gyorsrendezést alkalmazva, ez az érték csupán felső korlát marad.

Éppen ezért a gyorsrendezés nem egy biztonságos eljárás, abból a szempontból, hogy bizonyos összetételű populációkban (például, ha több azonos fitness értékű egyed van) nem

tudja javítani a buborék rendezés n^2 -es felső korlátját, így az Elitista szelekció továbbra sem annyira megbízható, mint a rendezés nélküli Rulett kerék (Tasnádi Attila: Számítástudomány gazdaságinformatikusoknak, 2006).

Tehát, bár a Gyorsrendezést használó Elitista szelekció jobb futási eredményeket mutat, a rendezési eljárások sajátossága, és a Rulett kerék elv már korábban ismertetett előnyei (a Darwini természetes kiválasztódás leképezésében) miatt, inkább a Rulett kerék szelekciót tartjuk a leghatékonyabbnak a genetikusan szelekciós stratégiái közül.

5. Ősrobbanás (Big Bang-Big Crunch)

A metaheurisztikát Erol és Eskin fejlesztette ki 2006-ban, a genetikusan algoritmus hatékonyságának javítását tűzve ki célul. Éppen ezért én is megpróbálom alkalmazni a saját problémámra, és megnézni valóban képes-e egy ilyen természetű probléma esetén is javítani a futási eredményeken.

Az algoritmus alapgondolata a világegyetem 15-20 billió éve lezajlott ősrobbanásra, és a világegyetem a jövőben talán valamikor bekövetkező feketelyukká történő összeomlására építkezik. Alapvető hasonlósága a genetikusan algoritmussal, hogy szintén evolúciós algoritmus, azaz többemű populáció egyedeit próbálja a generációkon keresztül a fitness értékek segítségével az optimum felé terelni. Azonban a módszere erre nem a természetes kiválasztódás, hanem az ősrobbanás szimulálása, leképezése.

A metaheurisztika véletlenül generált kezdeti populációval indul (a keresési tér határainak figyelembevételével), amely egyedeire kiszámoljuk a fitness értékeket. Ezután az úgynevezett tömegközéppontot kell azonosítanunk a populációban. Erol és Eskin szerint erre a legcélravezetőbb a következő egyenletet használni:

$$x^c = \frac{\sum_{i=1}^N \frac{1}{f_i} x_i}{\sum_{i=1}^N \frac{1}{f_i}}$$

Az egyenletben x^c a tömegközéppont, f_i és x_i pedig rendre az aktuális fitness érték, illetve populációbeli egyed. Azonban, ha az egyenlet használata a probléma természete miatt nem lehetséges, vagy ha ez a célravezetőbb, akkor megengedett a legjobb fitness értékkel rendelkező egyedet tömegközéppontnak tekinteni. Eddig tartott az ún. Big Bang fázis, azaz a robbanás.

A következő, Big Crunch, azaz az összeomlás fázisában a megtalált tömegközéppontot felhasználva új populációt hozunk létre, szintén egy egyenletet használva, ahol egy új egyedet a következőképpen határozzunk meg:

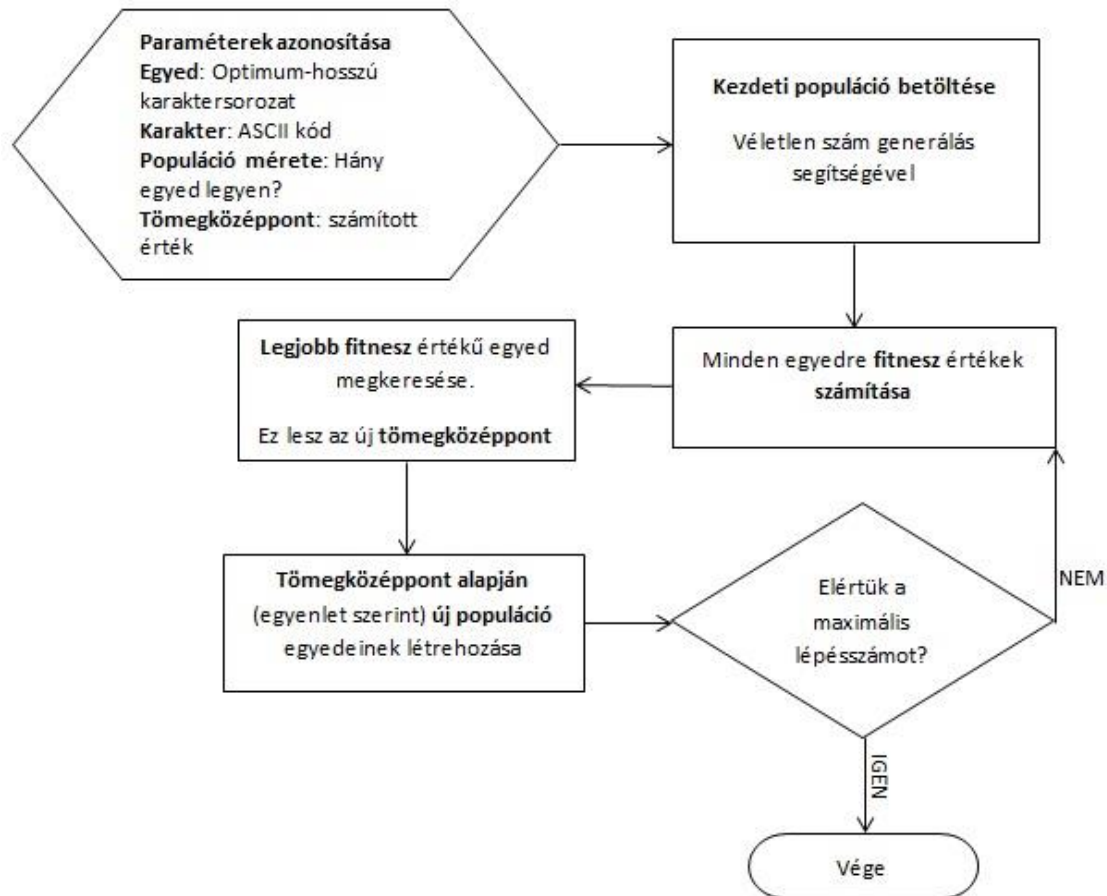
$$x^{új} = x^c + \frac{l * r}{k}$$

Az egyenletben x^c a már ismert tömegközéppont, l egy olyan szám, melynek értékét úgy választjuk, hogy az elemek a keresési térbe essenek, r egy véletlen szám a $(-1;1)$ intervallumon, végül k pedig az aktuális generációs szám.

Ezt a két fázist a megállási kritériumig ismételve a populáció a keresési tér egy pontjára fog beszűkülni, és reményeink szerint ez a pont az optimális megoldás.

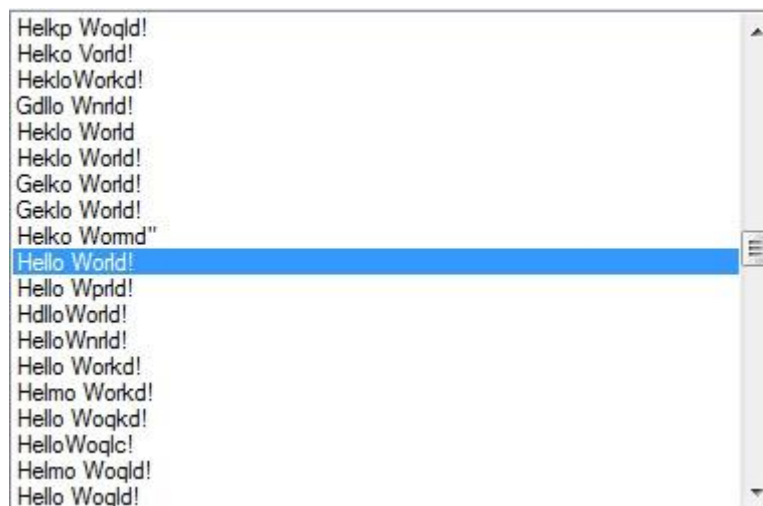
Látszik tehát, hogy az Ősrobbanás algoritmus úgy akar javítani a genetikus algoritmuson, hogy a populációk fejlesztésében szerepet játszó három operátort, a szelekciót és a mutációt ötvözi a tömegközéppont segítségével. Hiszen a tömegközéppont számításakor biztosítjuk, hogy a populáció legjobb tulajdonságai öröklődjenek, ez a szelekció. Az új egyedeket meghatározó képlet pedig egyszerre irányított módosítási stratégia (a tömegközéppont felhasználása, l paraméter), azaz keresztezés valamint a véletlen hatásnak (mutáció) is szerep jut az r paraméteren keresztül. Lássuk, hogyan is alkalmazható a dolgozat problémájára ez a metaheurisztika.

A genetikus algoritmusból kiindulva az egyedek az egyes karaktersorozatok lesznek, ezeket szervezzük populációba. A tömegközéppont meghatározása az egyenletet használva nem fog működni, mivel az egyes x_i értékeink karaktersorozatok, a karakterek ASCII kódjait nem lehet az egyed fitness értékének reciprokjával beszorozni. Ha esetleg karakterenként számolnánk fitness értékeket, és úgy próbálkoznánk szorzással, akkor ha egy karakter már a megfelelő helyen áll a keresési térben, azaz fitness értéke nulla, akkor a reciprokképzés miatt nullával való osztás történné, ami nem lehetséges. Így megmarad számunkra a legjobb egyed kiválasztása tömegközéppontnak. Az egyenlet alkalmazása egyénként is inkább matematikailag jobban modellezhető problémák esetén javasolt, pl. többváltozós függvény szélsőérték keresése esetében (Erol-Eskin: A new optimization method: Big Bang–Big Crunch, 2006). Az ezen elvek szerint kialakított algoritmus folyamatábrája a következő:



10. ábra: Az Ősrobbanás algoritmus alkalmazása a dolgozat problémájára

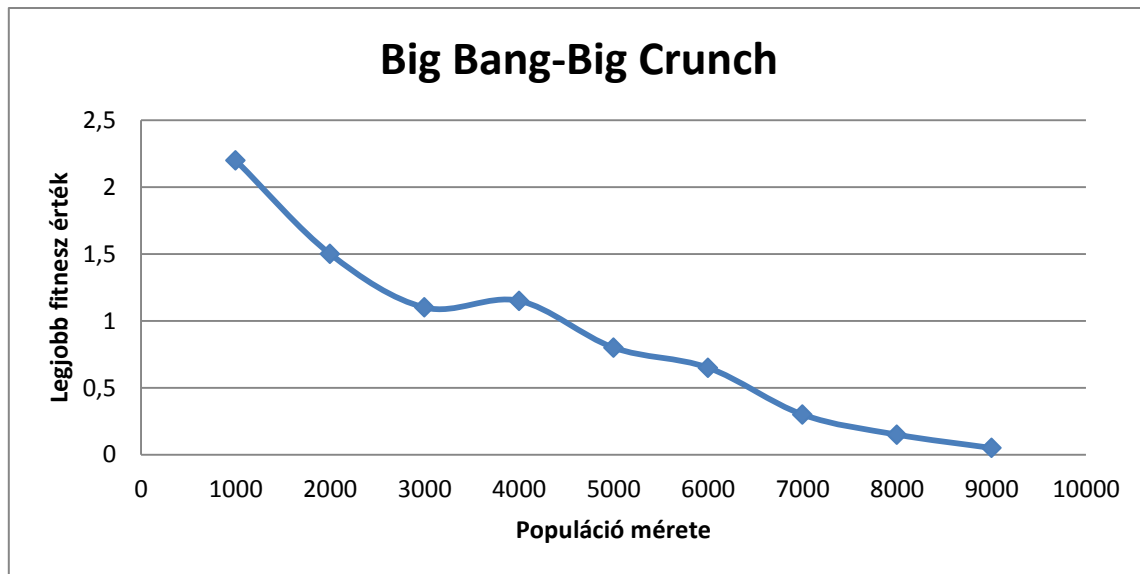
Az algoritmus lefuttatásakor az Ősrobbanásnak egy nagyon fontos tulajdonsága kell, hogy feltűnjön. Mégpedig az, hogy ha valamelyik populáció meg is találja, tehát tartalmazza az optimumot, ha tovább futtatjuk az algoritmust, akkor a tömegközéppontot (ebben az esetben pont az optimumot) alapul vevő képlet szinte biztosan az optimumtól eltérő értékeket fog számolni az új populációra. Egyedül akkor tudná úgymond „tartani” az eredményét, ha az r paraméter értéke nulla lenne, ez a valószínűség pedig a $(-1;1)$ folytonos intervallumban már majdhogynem szintén nulla. Így a következő futási képet fogjuk tapasztalni:



11. ábra: Az Ősrobbanás algoritmus futása

Az Ősrobbanás algoritmus láthatóan nem képes a megtalált optimum megtartására, így például nem hatékony az alkalmazása gépi tanulási problémák alkalmazására, mint pl. a genetikus programozás, hiszen hiába találná meg egy probléma megfelelő megoldását, ha az algoritmust nem állítjuk le, nem képes „tárolni” és később alkalmazni az eredményt. Hatékonyságvizsgálatán pedig látjuk, hogy nem feltétlenül képes jobb eredményt szolgáltatni a genetikus algoritmusnál a vizsgált probléma esetében.

A metaheurisztika egy nagyon fontos tulajdonsága, hogy a keresési tér szűkülése megfelelő ütemben menjen végbe. Ha túl gyors ez az ütem, akkor nem biztos, hogy bejárjuk a teljes keresési teret, és nagy valószínűséggel lokális optimumban ragadunk. Ezt az ütemet az algoritmus egyetlen szabályozható paramétere segítségével, a populáció méretével tudjuk szabályozni. Minél nagyobb a populáció, annál jobb, pontosabb tömegközéppontokat tudunk választani, amikből kiindulva a keresési tér legmegfelelőbb részeit tudjuk bejárni. Ez a tulajdonság hasonlít a genetikus algoritmusban alkalmazott darwini természetes kiválasztódás elvéhez, hiszen ott is egy nagyobb mintából jobban ki tudtuk választani az átlagosnál jobb egyedeket. Ebből is látszik, hogy maga az Ősrobbanás a genetikus algoritmus javítását célozza. A populáció méretét szabályozva végül a következő futási eredményeket kapjuk:



12. ábra: A populáció méretének változtatásának eredményei

Az eredményeket elemezve látszik, hogy ha a szűkülés ütemét nem megfelelően választjuk meg, akkor nagyon könnyen lokális optimumokban ragadunk. 9000-es populáció méretnél viszont már sikerül a globális optimumot megtalálni, és ekkor nagyjából a hatékonyságot tekintve a genetikusan verseny szelekciójával egyezik meg az Ősrobbanás, átlagos futásidőt (2,563 másodperc) vizsgálva. Tehát látszik, hogy megfelelő ütemezés mellett az Ősrobbanás valóban hatékonyan kombinálja a genetikusan három operátort, hiszen 9000 egyedű populáció egyedében is azt a hatékonyságot érte el, mint a verseny szelekció 5000-es populációval, azaz hatékonyabban kezel szimultán több vektort. Ám az ütemezés bizonytalansága miatt, és a jobb mutatókkal rendelkező rulett kerék szelekciót (1,213 másodperc), illetve gyorsrendezéssel elitista szelekciót (1,014 másodperc) is figyelembe véve nem igazán tekinthető hatékonyabbnak ez a metaheurisztika a problémánkra alkalmazva. Ez adódik a feladat nem könnyű matematikai kezelhetőségéből. Mivel ha tudnánk a tömegközéppont számításához Erol és Eskin egyenletét használni valószínűleg egy kisebb populációból is pontosabban be tudnánk járni a keresési teret.

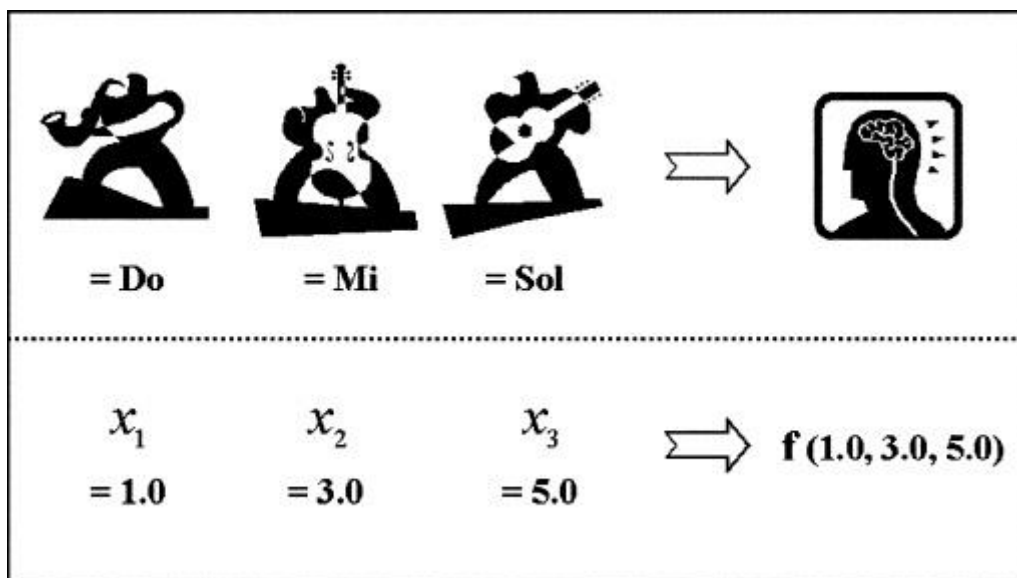
Mivel ez a lehetőség most nem áll rendelkezésünkre, és ez sem garantál biztos javulást ezért érdemesebb egy másik módszert keresni a genetikusan algoritmus eredményeinek javítására.

6. A harmónia kereső algoritmus

A metaheurisztikát 2005-ben alakította ki, Lee és Geem. Elsősorban mérnöki optimalizálási problémák megoldására alkalmazták, az algoritmus igen hatékonynak bizonyult ezekben az esetekben. Eredményesen oldott meg szerkezettervezési, energia megtakarítási és

hálózattervezési problémákat. Érdekes lehet megnézni képes-e felülmúlni a genetikus algoritmus hatékonyságát, ha a mi problémánkra alkalmazzuk.

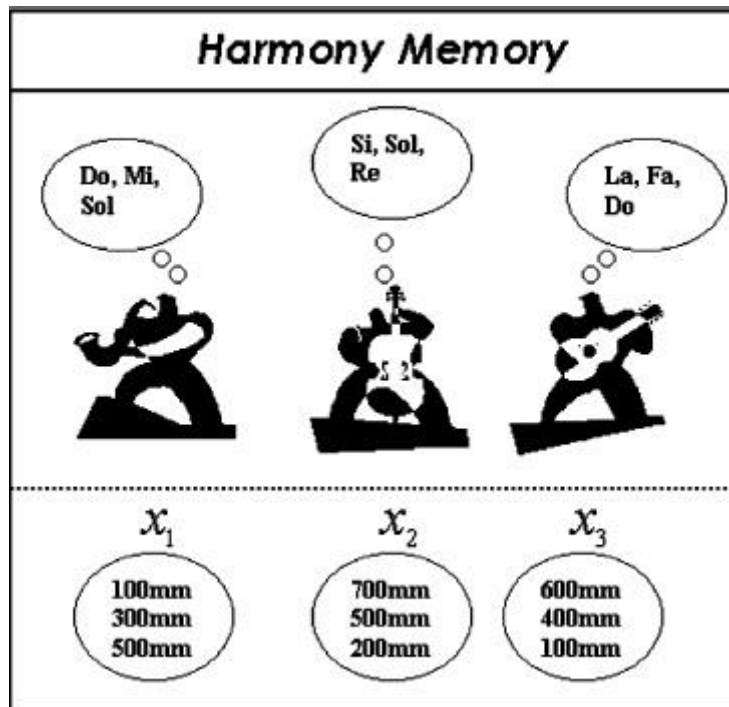
Az algoritmus a zenei világból merít ötleteket. Azt a keresési folyamatot kívánja megragadni, amely során egy improvizáló zenekar végül eljut a zeneszerző vagy éppen karmester által a legjobb hangzásúnak nyilvánított harmóniáig. Improvizáció során a zenekarban minden zenész öltetszerűen lejátsszik egy dallamot, és ezen dallamokból kialakul egy egységes hangzás. Jó esetben a dallamok egy harmóniát, azaz a fülünknek kellemes hangzást alakítanak ki. Tehát az egyes megoldásaink a dallamok, és a kialakult hangzás esztétikai minősége a fitnessz érték:



13. ábra: Az improvizáció zenei analógiája

Forrás: Lee és Geem, 2005

A karmester megjegyzi a kialakult hangzást, és ehhez tudja viszonyítani a további improvizációs eljárások során kialakult új hangzásokat, illetve azok esztétikai minőségét. Ha valamelyik új hangzással jobb harmóniát kapunk, akkor lecseréli az addigi legrosszabbul sikerült hangzást az újra. A zenemű hossza határozza meg, hogy egyszerre hány dallamra emlékszik a zeneszerző. Matematikailag így tehát kapunk egy lehetséges megoldásokból álló memóriát (Harmony Memory) és ennek méretét (HMS):



14. ábra: A harmónia keresés memóriája

Forrás: Lee és Geem, 2005

Ha a korábban vizsgált metaheurisztikákat nézzük, akkor ez a populációt és a populáció méretét jelentette. Tehát ez az algoritmus is evolúciós jelleget mutat, azaz a harmónia, vagy repertoár dallamait próbálja fitness értékek segítségével (jelenleg hangzások) több improvizációs eljárás egymás utáni alkalmazásával az optimum (harmonikus hangzás) felé terelni.

Most érdemes áttekintenünk, hogy ez egyes zenészeink az improvizáció során milyen módszereket alkalmaznak. A folyamat alapvetően három szabály alapján zajlik:

- A zenészek emlékezetből lejátszanak egy ismert dallamot.
- Más hangfekvésben vagy más ritmusban játszanak le egy már lejátszott dallamot.
- Teljesen ötletszerűen, érzéseikre hagyatkozva játszanak le egy új dallamot.

Hasonló elven működik az improvizáció a harmóniakereső algoritmusban, bár a három alapszabály egy picit más:

- A repertoárból lejátszunk egy már ismert dallamot.
- Egy ismert dallamon módosítunk.
- Teljesen véletlen dallamot generálunk.

Ezen három alapszabály alkalmazását kell lefordítanunk a matematika nyelvére. Erre a feladatra két valószínűséget fogunk bevezetni.

Az egyik a HMCR (Harmony Memory Consideration Rate), ami a memóriából való választás valószínűségét adja meg. Annak a valószínűsége, hogy teljesen véletlen megoldást generálunk $1 - \text{HMCR}$. Ha a HMCR túl alacsony értékű, akkor csak kevés dallam kerül kiválasztásra a repertoárból és az algoritmus lassan konvergál. Ha ez az érték túl magas, közel 1, akkor majdnem az összes hang felhasználásra kerül a repertoárból, s ekkor csökken más dallamok felfedezésének lehetősége.

A következő valószínűség a dallam módosításának (hangmagasságának megváltoztatása) esélyét határozza meg, azaz ha már a memóriából választottunk egy dallamot, akkor ezzel a PAR (Pitch Adjusting Rate) valószínűséggel módosítunk rajta, és $1 - \text{PAR}$ valószínűséggel hagyjuk változatlanul.

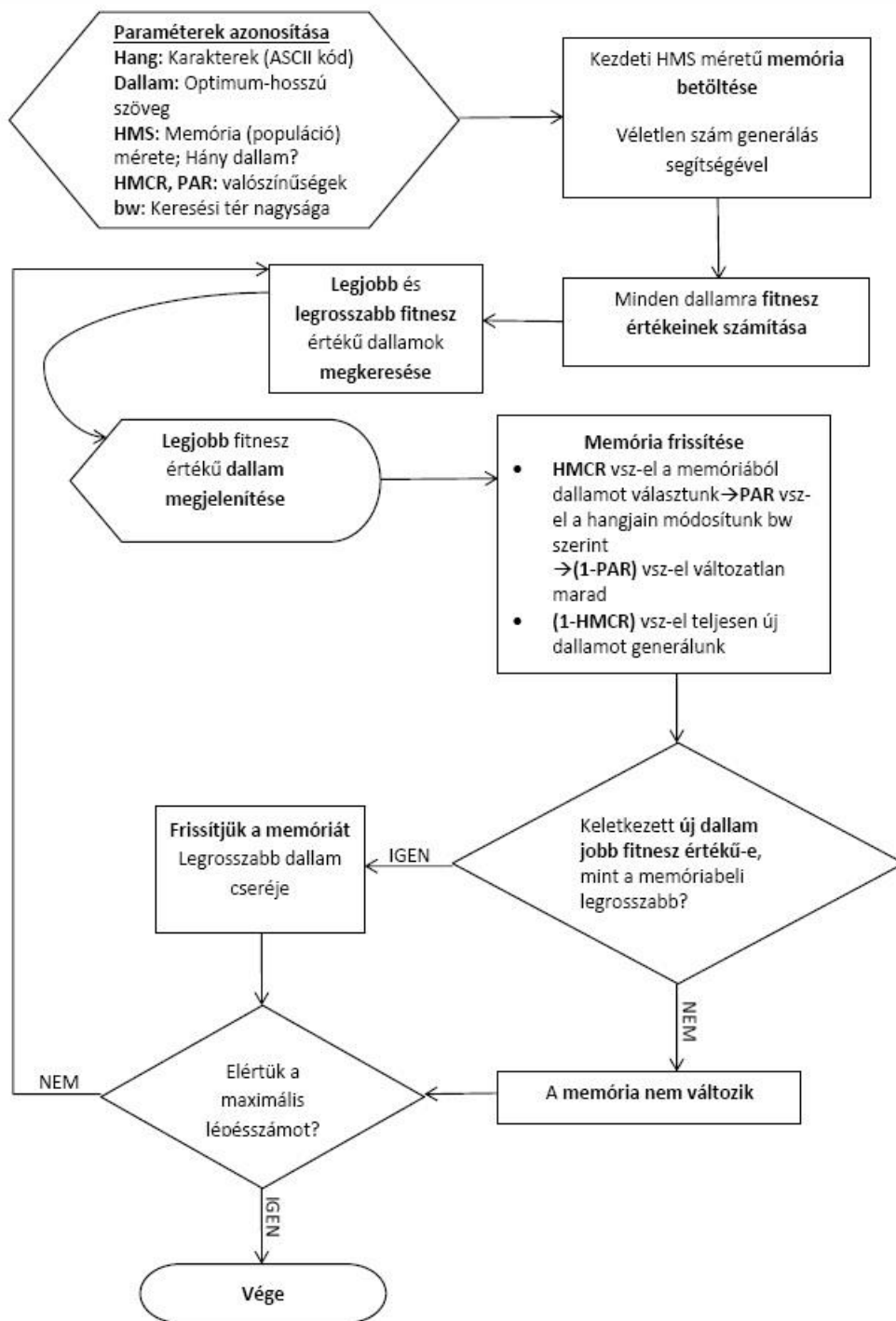
A hangmagasság megváltoztatása a zenei gyakorlatban lineárisan történik. Ezt a lépést a következő egyenlet írja le:

$$x_{új} = x_{régi} + bw * r$$

A képletben $x_{új}$ a keletkezett új dallam, $x_{régi}$ pedig a memóriából kiválasztott már lejátszott dallam. A sávszélességet (bandwidth), azaz a keresési tér nagyságát a bw paraméter jelöli. Végezetül, r egy egyenletes eloszlású véletlen szám a $(-1;1)$ intervallumban.

Áttekintve a metaheurisztikát elmondható, hogy a harmóniakereső algoritmus is tulajdonképpen a genetikus algoritmus egy változatának tekinthető. A géneknek megfeleltethetjük a hangokat, az egyedeknek a dallamokat és a harmóniának (repertoár vagy memória) pedig a populációt. A szelekció egyszerűsödik, de jelen van: a memóriabeli legrosszabb hangot lecseréljük, tehát a kevésbé rátermett egyedek „kihullnak”. A keresztezés operátora teljesen kiveszett az algoritmusból, ám helyette kétféle mutációs operátort kapunk: a hangmagasság változtatása bizonyos géneket módosít, ez felel meg a klasszikus (genetikus algoritmusban is alkalmazott) mutációnak, míg a teljesen új dallam generálása egy eldurvult, nagyon agresszív mutációt jelent. További érdekesség a harmóniakereső algoritmusban, hogy ezek az operátorok nem sorban egyszerre, hanem bizonyos valószínűség szerint következnek be, azaz váltakoznak az egyes ciklusokban.

Éppen ezért, amikor a problémánkra alkalmazzuk a harmóniakereső algoritmust, akkor az Ősrobbanáshoz hasonlóan szintén a genetikus algoritmus paraméterezése során szerzett tapasztalatokat próbáljuk meg fölhasználni, ami jelen esetben is gond nélkül sikerül. Így a következő folyamatábrát kapjuk:



15. ábra: A harmónia kereső algoritmus alkalmazása a dolgozat problémájára

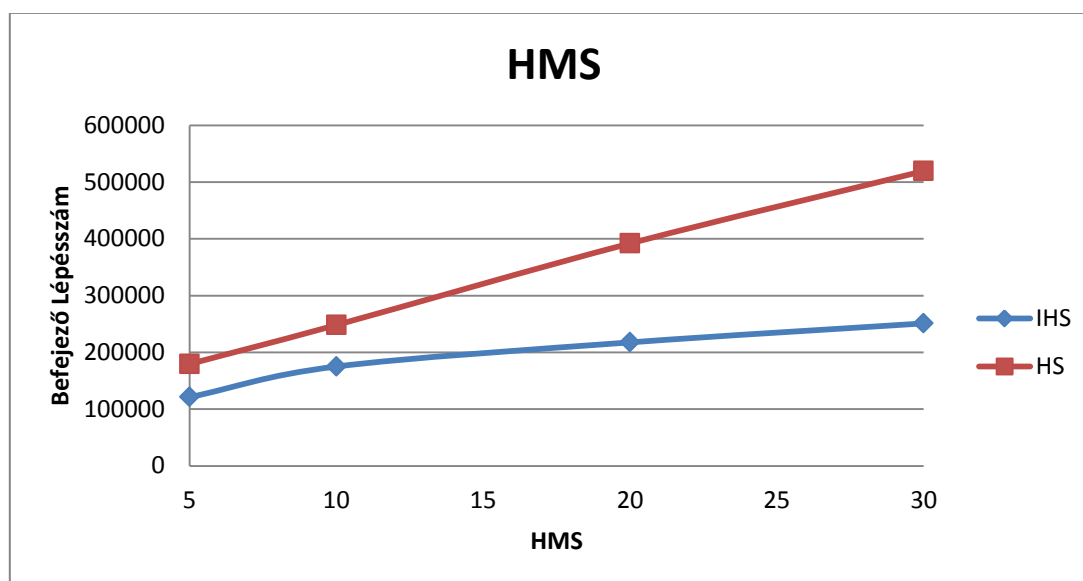
A folyamatábrából leszűrhető, hogy az algoritmus hatékonyságvizsgálatához a következő paraméterek hatását kell megvizsgálni: A memória méretét (HMS) és a két valószínűséget (HMCR, PAR). A mellékletben szereplő 35. ábrán látható, hogy a fejlesztett harmónia kereső algoritmus esetében sem szükséges másként eljárni. A hatékonyságvizsgálat mértékegysége a saját paraméterek vizsgálatakor, a genetikus algoritmushoz hasonlóan, az optimum megtalálásához szükséges lépésszám, a futásidő ezzel arányos.

Érdemes megemlíteni, hogy a harmóniakereső algoritmust (Harmony Search, HS) 2007-ben Mahdavi, Fesanghary és Damangir továbbfejlesztette (Improved Harmony Search, IHS). Az új elem ebben az algoritmusban csupán annyi, hogy a PAR érték a futás során nő, míg a módosítás mértéke (*bw*) a futás során csökken. Ez a finomhangolás jelentős javulást eredményez a hatékonyságban, melyet a szerzők számítási eredményekkel igazolnak a tanulmányban. Érdemes még megvizsgálni, hogy ez a fejlesztés a feladatunk esetében eredményez-e jelentős javulást a hatékonyságban.

Természetesen az egyes paraméterek hatékonyságvizsgálatát a többi paraméter változatlanul hagyásával, és egy többelemű minta átlagaként végezzük el.

6.1. HMS

A paraméter vizsgálata során arra jutunk, hogy mivel a harmóniakereső algoritmus esetén nem igazán tudjuk a genetikus algoritmushoz hasonló módon kezelni (keresztelés, szelekció) a több egyedből álló memóriát (HMS), csak különböző mutációs operátoraink vannak bizonyos valószínűséggel (módosítás, teljesen új generálása), ezért a HMS növelésével tulajdonképpen csak bővítjük a keresési teret a megoldások változatosságának növelése nélkül (ezt a mutációs operátor már megoldotta). Ezzel romlik a hatékonyság.



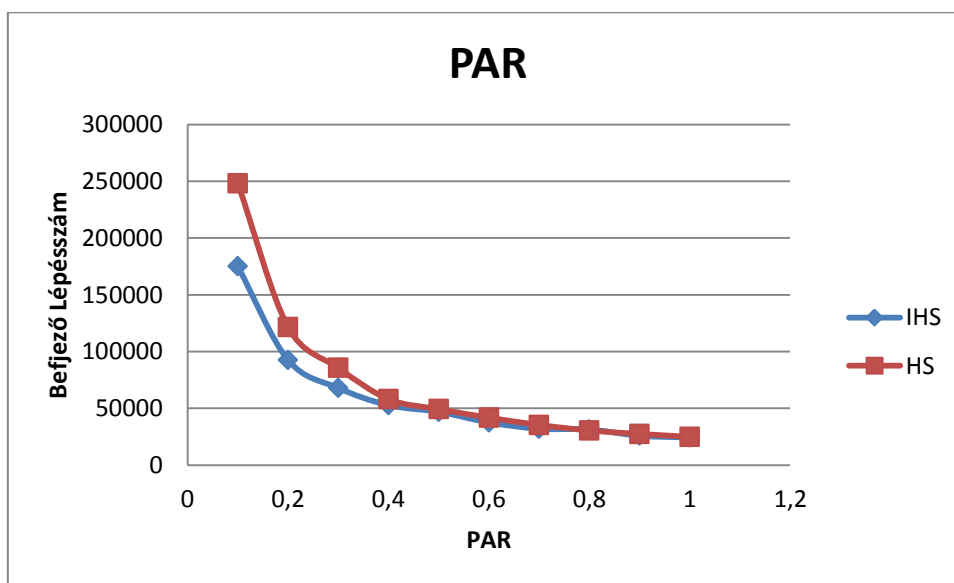
16. ábra: A HMS változtatásának eredményei

A grafikonon látszik, hogy egyszerű HS esetén a lépésszámok egy egyenest határoznak meg. Ha az optimum megtalálásához szükséges lépésszámot, illetve a HMS méretét két statisztikai ismérvnek tekintjük, akkor a 0,9996-os értékű korrelációs együttható is megerősíti a függvényszerű kapcsolatot. Tehát a kisebb memória majdnem determinisztikusan jobb lépésszámot eredményez.

Ha a fejlesztett harmóniakereső algoritmus eredményeit is vizsgáljuk, akkor látható, hogy az IHS kimagaslóan jobban teljesít, mint a sima HS. Ennek oka az, hogy az IHS jobban kihasználja a nagyobb memória méretet, mivel a futás során nő a PAR valószínűsége és csökken a módosítási tartomány, azaz a sávszélesség. Így ebben az esetben segítség a lokális optimumokból való „kilökéshez” a többelemű populáció, míg a sima HS gyakrabban hagyja a változatlanul a memóriát (az állandó PAR valószínűség miatt), így jobban meglátszik a hatékonyságán, ha növeljük annak méretét.

6.2. PAR valószínűség

Mindkét esetben, HS-t és IHS-t vizsgálva, a nagyobb PAR valószínűség eredményez nagyobb a hatékonyságot, hiszen kis fitness érték (lokális optimum) esetében már a teljesen véletlen megoldás túlságosan is kimozdítja a lokális optimumból a fitness értéket, így ekkor csak egy finom módosítás kell már csak a dallamokon, és ezt a nagyobb PAR valószínűség tudja biztosítani.



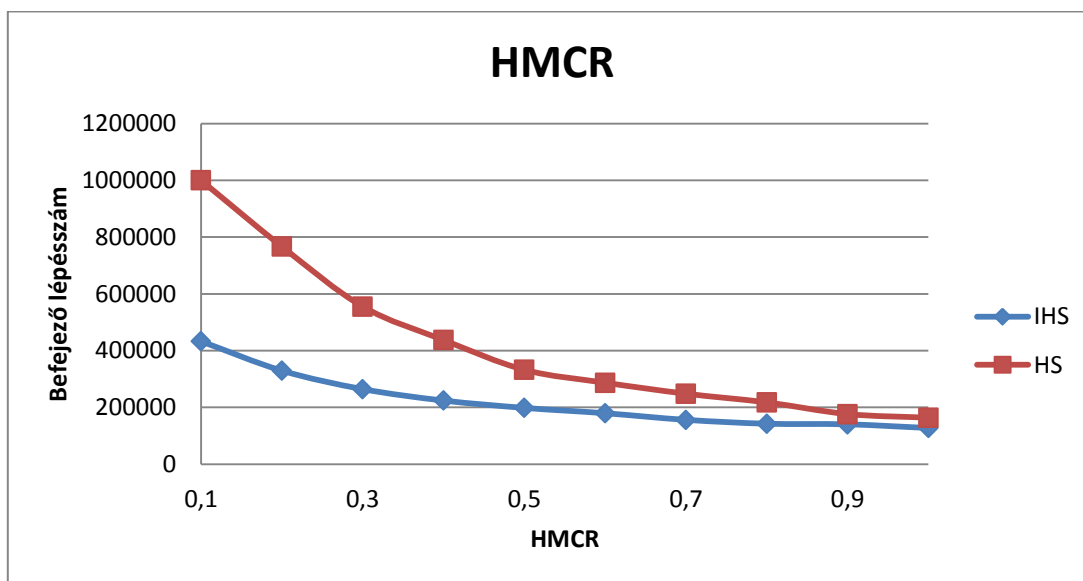
17. ábra: A PAR valószínűség változtatásának eredményei

Az IHS PAR értékének növelése csak kisebb kezdeti PAR értéknél biztosít előnyt, hiszen kellően nagy PAR valószínűségnél szinte már mindegy mennyire nagy az egyik, illetve a másik mennyire magas értékről indul.

6.3. HMCR

A paraméter vizsgálatánál tulajdonképpen papírforma eredményt kapunk: ha ez a valószínűség túl alacsony értékű, akkor csak kevés dallam kerül kiválasztásra a repertoárból és az algoritmus lassan konvergál. Mivel többelemű memóriát alkalmazunk, így már

biztosítunk egy kellő változatosságot a lehetséges megoldásaink esetében, így csak ritkán van szükség egy teljesen véletlen megoldás szolgáltatása „plusz” változatossági tényezőre.

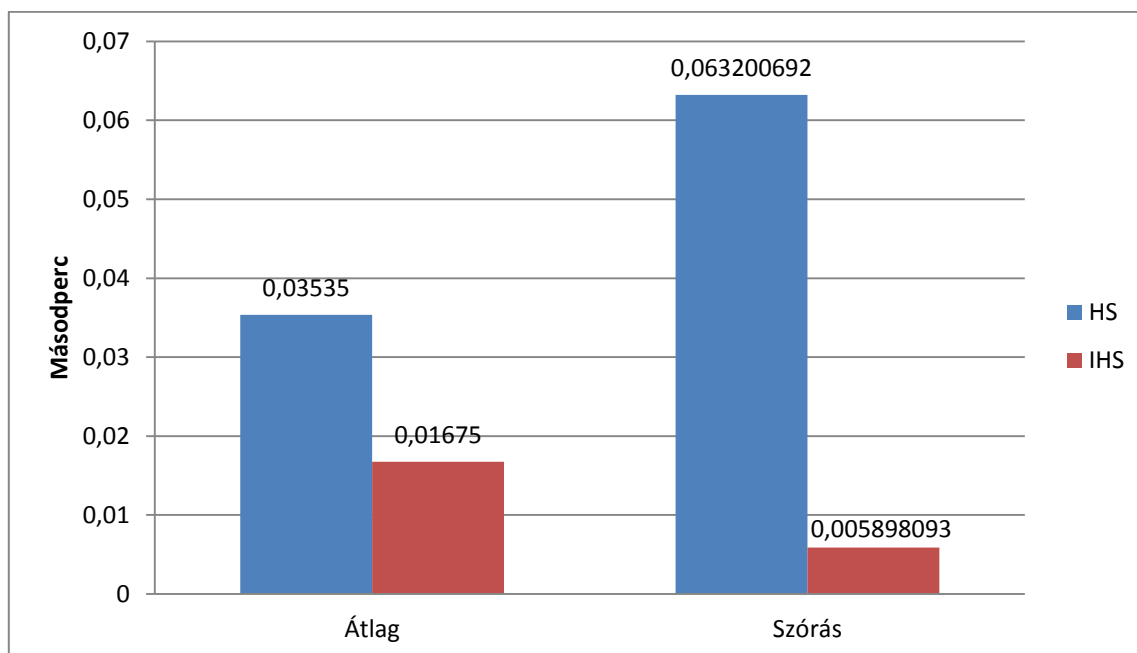


18. ábra: A HMCR valószínűség változtatásának eredményei

A grafikont megvizsgálva itt is megmutatkozik az, hogy az IHS jobban ki tudja használni a többelemű memóriáját, hiszen az ebből való választás valószínűségét növeljük. Tehát az IHS folyamatosan javuló paraméterei (PAR, *bw*) miatt ismét jobban tudja módosítani a memóriát.

6.4. A többelemű memóriájú harmónia kereső algoritmus összefoglalása

Ha az algoritmust a legjobb hatékonyságot eredményező beállításával (HMS, PAR, HMCR) futtatjuk, a futásidejét vizsgálva a következő eredményre jutunk:



19. ábra: A harmónia kereső algoritmus futásideje

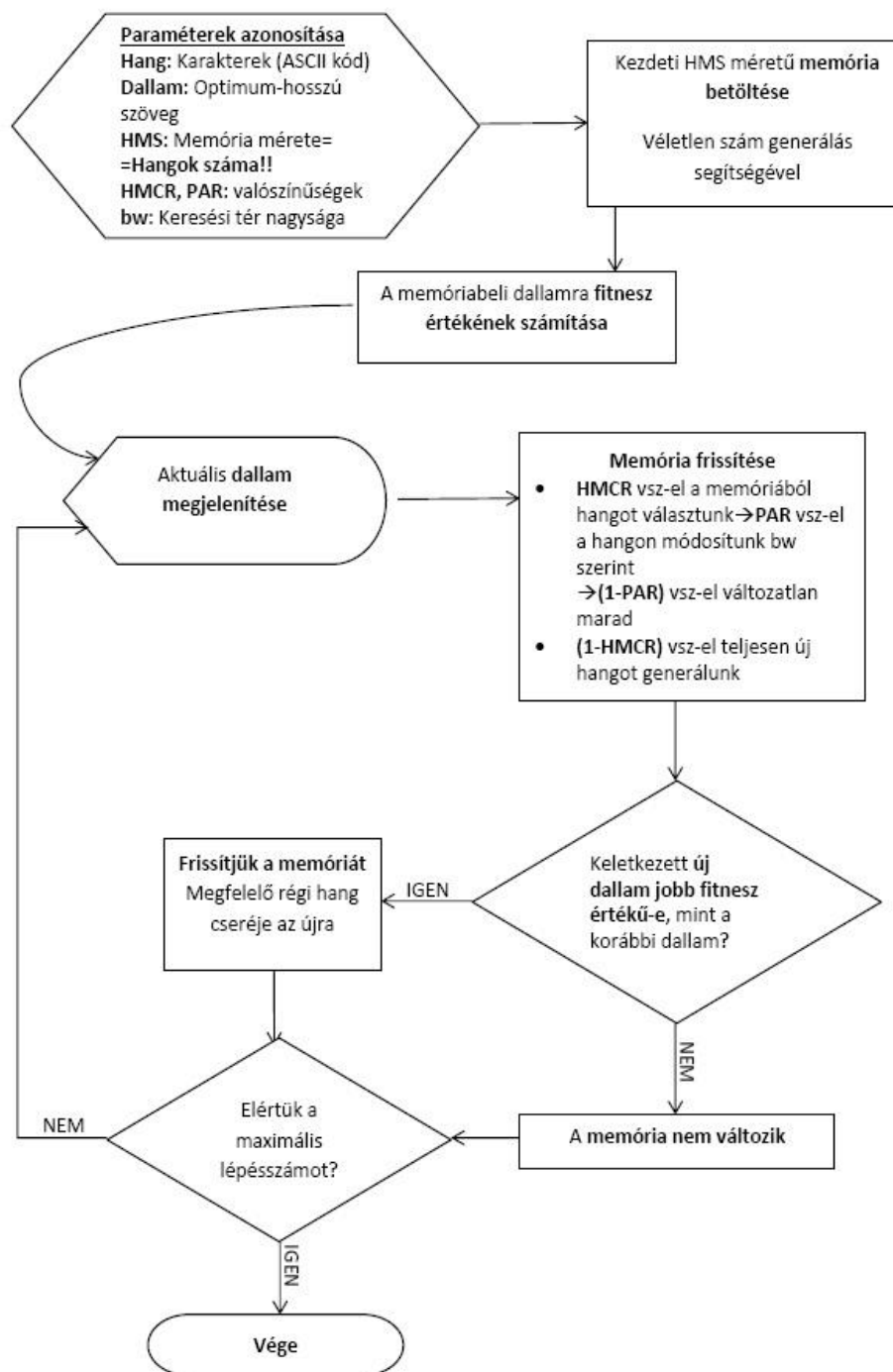
Láthatjuk, hogy az átlagos futásidő nagymértékben javult. Jóval egy másodperc alá került, ez még a genetikai algoritmus legjobb eredményeit (1,213 másodperc: Rulett kerék elv, 1,014 másodperc: gyorsrendezési elitista szelekció) is felülmúlja. Ráadásul az IHS ezt a remek átlagos futásidőt egy igen minimális, szinte elhanyagolható szórással eredményezi.

Ennek oka abban keresendő, hogy a harmónia kereső algoritmus memóriájának mérete csupán 5-10 elemű szemben az eddig megismert metaheurisztikák több ezres populációs méretével. Tehát nem kell olyan sok vektort szimultán kezelni, ez pedig jelentősen javít a futásidőn. Ráadásul a bonyolultabb szelekciós operátorok és keresztezési eljárások helyett alkalmazott egyszerűbb mutációs operátorok sem igényelnek annyi processzoridőt. Így bár a lépésszám magasabb értéket mutat, de a műveletek nem olyan időigényesek.

A másik fontos tanulság a determinisztikusnak tekinthető kapcsolat felfedezése a memória mérete és az optimum megtalálásához szükséges lépésszám között. Ebből látszik, hogy a leghatékonyabb az egyelemű memória alkalmazása lenne.

7. A módosított harmónia keresés

A harmónia keresés algoritmusát nem feltétlen szükséges többelemű memóriára alkalmazni. Tekinthetünk egy egyelemű populációt is, azaz a hangok lesznek az egyes döntési változók, és a dallam veszi át a hangzás helyét. Ekkor a memória mérete (HMS) a dallam hossza. Az algoritmus ebben az esetben is lényegi változtatások nélkül alkalmazható, mivel egy dallam előállításához is ugyan olyan improvizációra van szükség, mintha hangzásokat raknánk össze dallamokból. Csak az egész folyamat „egy szinttel lejjebb” zajlik. Az egyetlen lényegi különbség, hogy csak a dallam megoldásának fitness értékét tudjuk mérni, a hangokét nem, így csak a teljes dallam viszonylatában tudunk dönteni a módosítástól, de ez jelenleg nekünk megfelelő is lesz, mivel a dallam jelenti a lehetséges megoldást, az egyes hangok önmagukban nem határoznak meg fitness értéket (a zeneszerző sem tud mondani semmit a művéről egy hang alapján, de a dallamokon keresztül tudja őket szabályozni). Az így kialakított algoritmus folyamatábrája a következő:



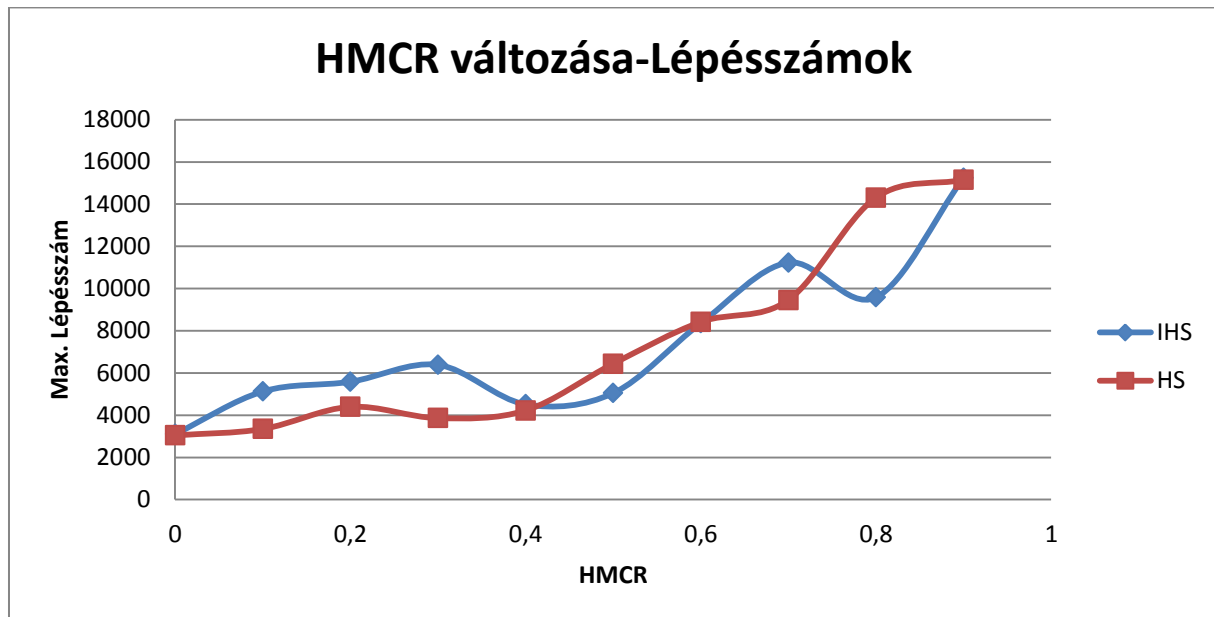
20. ábra: Az egyelemű harmónia kereső algoritmus alkalmazása a dolgozat problémájára

Természetesen a finomhangolt harmónia kereső algoritmus, PAR valószínűség növelésével, sávszélesség (*bw*) csökkentésével, ebben az esetben is alkalmazható (Mellékletek: 36. ábra), így a hatékonyságvizsgálat során az IHS eredményeit is vizsgáljuk.

Mivel a HMS az adott karaktersorozat miatt konstans érték marad, ezért a vizsgált paraméterek csak a HMCR és PAR valószínűségek. A mértékegység továbbra is az optimum megtalálásához szükséges lépésszám, a futásidő ezzel egyenesen arányos.

7.1. HMCR

A módosításunk a metaheurisztikán meghozta a kívánt eredményt: a lépésszám jelentősen lecsökkent, százezresről ezres nagyságrendűre.



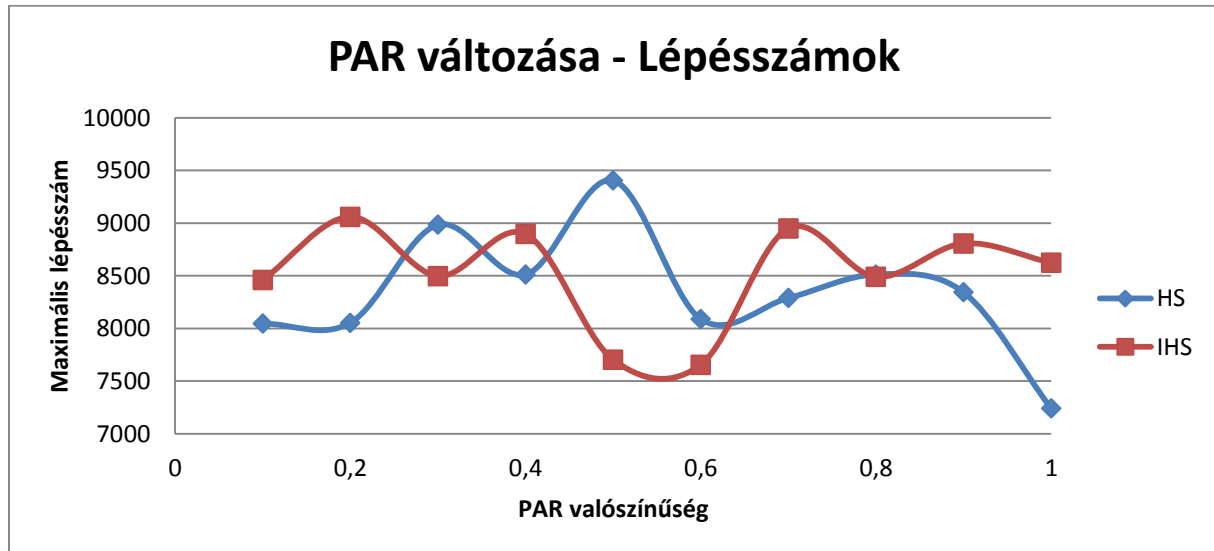
21. ábra: A HMCR valószínűség változtatásának eredményei

A grafikon eredményeit elemezve láthatjuk, hogy jelen esetben a kisebb, 50% alatti valószínűségek jelentenek hatékonyabb futási eredményeket. Ha jobban belegondolunk ez nem is olyan váratlan. Hiszen itt a módosítások hang, azaz karakterszinten történnek. Ekkor, ha a memóriából való választás valószínűségét növeljük, egyúttal nagyobb eséllyel hagyjuk változatlanul az egyetlen megoldásunkat, ezzel lassítva a konvergálás folyamatát. Továbbá ha teljesen új hangot generálunk, akkor nem a teljes megoldás cserélődik le, csak egy hangja. Tehát ez az operátor nagyjából egyenértékűvé vált a hang sávszélesség alapján történő módosításával. Ráadásul a teljesen véletlen karaktergenerálás még jobbnak is tekinthető az előző karakter (hang) alapján történő módosításnál, mivel jelenleg a memória elemeinek korábbi állapota nincs hatással a következő állapotra (a dallamok komplexitása miatt megérte egy korábbi dallamból kiindulni), így nem is igazán éri meg egy ilyen egyenletet alkalmazni. Éppen ezért a nagyobb HMCR valószínűség többet árt, mint használ.

Ha az IHS szempontjából vizsgáljuk az eredményeket, láthatjuk, hogy igazából sokat nem javít az eredeti algoritmuson. Ez tulajdonképpen az előbb elmondottakból következik is, hiszen az IHS pont azon a ponton javít (PAR, sávszélesség), amit eleve nem is éri meg gyakrabban alkalmazni. Viszont ha a HMCR egy kellően alacsony érték (40%-nál kisebb), azaz a memóriából választás megfelelő gyakorisággal történik, akkor már látszik az eredményeken az ott bekövetkezett javulás is.

7.2. PAR

Az előző részben leírtak miatt a PAR valószínűség változtatása nem igazán befolyásolja az algoritmus hatékonyságát. Az egyes PAR értékek mellett az optimum megtalálásához szükséges lépésszám egyenletesen ingadozik:



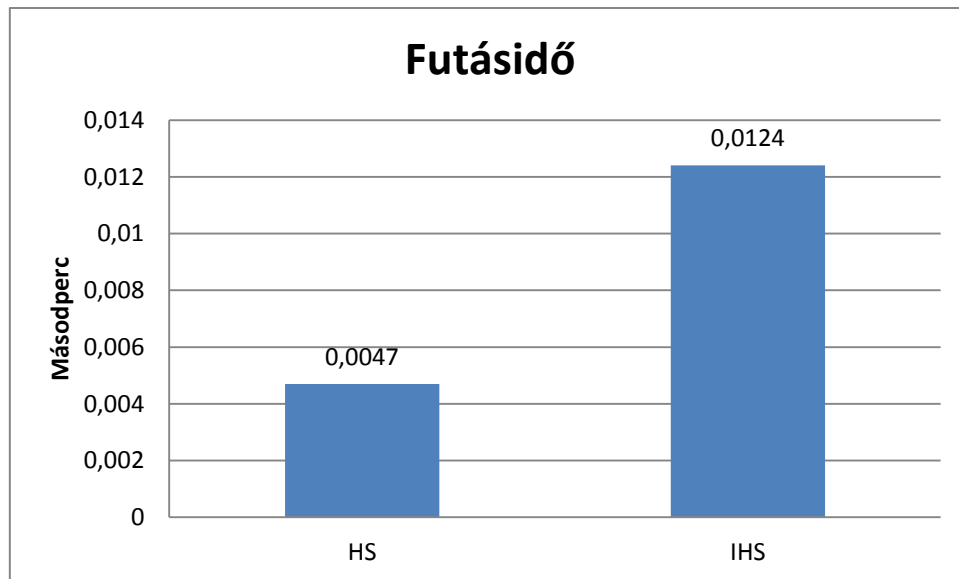
22. ábra: A PAR valószínűség változtatásának eredményei

A vizsgálat során nem eredményezett a PAR valószínűség változtatása jelentős változást az optimum megtalálásához szükséges lépésszámban. A lépésszám egyenletesen ingadozik a különböző PAR valószínűségek esetében. A két ismerv átlagos korrelációs együtthatója (-0,155) is egy abszolút értékében 0,3-nál kisebb eredményt szolgáltat, tehát a PAR valószínűség és az optimum megtalálásához szükséges lépésszám között még sztochasztikus kapcsolat sem feltételezhető.

Az IHS javítása megint minimális, ennek indokai is szintén az előző részben ismertettekhez hasonlóak, azaz az algoritmus azon részén történik javítás, amit kisebb valószínűséggel éri meg használni hatékonyság szempontjából.

7.3. A módosított harmónia keresés alkalmazásának összefoglalása

Ha végezetül a metaheurisztika legjobb beállításait alkalmazva a futásidőt vizsgáljuk, láthatjuk, hogy a sima HS algoritmus átlagos futásidejét sikeresen megjavítottuk (eredeti: 0,03535 másodperc), viszont az IHS értékei nagyjából nem változtak, minimális javulás tapasztalható (eredeti: 0,01675 másodperc):



23. ábra: Az egyelemű harmónia keresés futásideje

Az IHS rosszabb eredményeit az előző részekben már megmagyaráztuk.

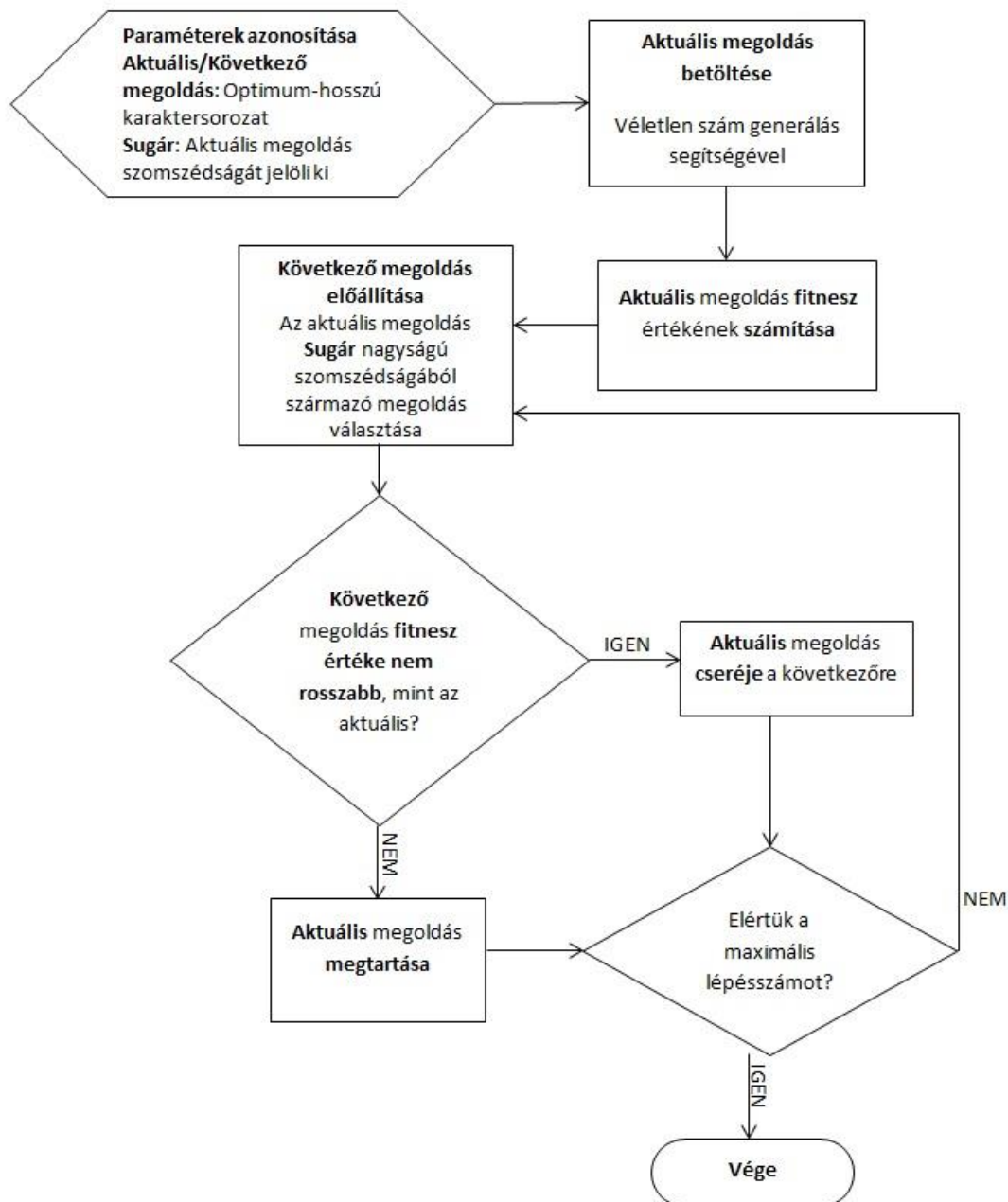
Vegyük észre azonban, hogy ezek a jó eredmények a sima harmónia kereső algoritmussal nagyon alacsony HMCR valószínűség alkalmazásával voltak elérhetők. Sőt, a legjobb eredmények az egyenesen nulla HMCR érték mellett születtek! A két ismerv korrelációs együtthatója is kimagasló értéket (0,940) mutat, azaz a HMCR valószínűség 88,43%-ban befolyásolja az optimum megtalálásához szükséges lépésszámot. Így tehát elmondható, hogy a jelenleg futásidőben leghatékonyabb megoldás egy 0 HMCR valószínűségű sima harmónia keresés. Viszont ilyen paraméterezéssel a harmónia kereső algoritmus egy másik metaheurisztika speciális esete, ami nem más, mint a sztochasztikus hegymászó.

8. Sztochasztikus hegymászó

A sztochasztikus hegymászó a vizsgált metaheurisztikák közül a legegyszerűbb. Egyelemű populációval dolgozik, oly módon, hogy kijelöl egy kezdeti lehetséges megoldást, és ebből kiindulva egy szomszédsági függvény segítségével járja be a keresési teret. Ha talál egy nem rosszabb megoldást az éppen aktuális megoldásnál, akkor lecseréli az aktuális megoldást erre az újra, és a szomszédsági függvényt erre a megoldásra hívja meg. Ezt a folyamatot ismételi a megállási kritérium bekövetkezéséig.

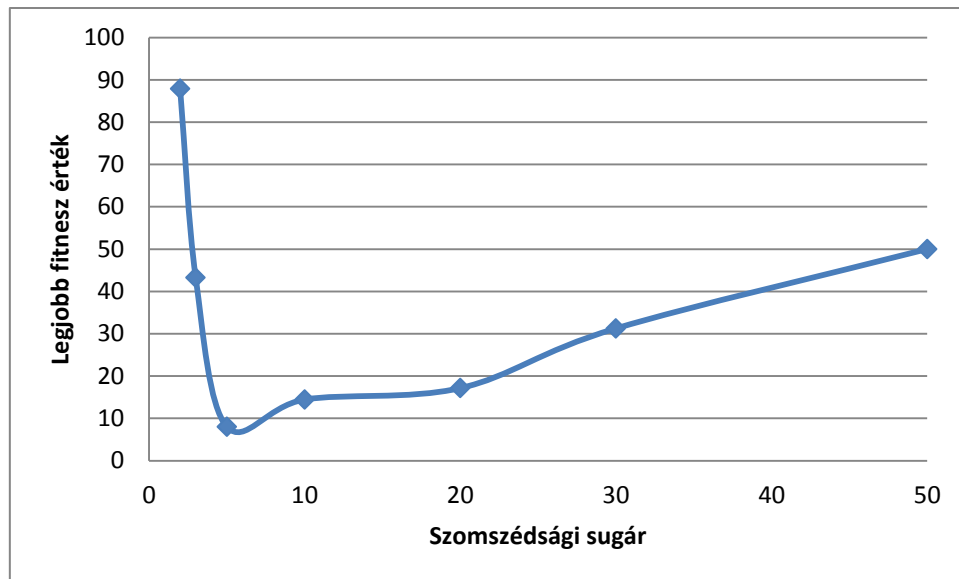
Az algoritmus működését áttekintve láthatjuk, hogy az egyetlen szabályozó eleme ez a szomszédsági függvény, tehát az a metódus, ami alapján az algoritmus bejárja a keresési teret. A metaheurisztikát alapvetően klasszikus matematikai függvényoptimalizálási problémák megoldására fejlesztették ki. Egy ilyen optimalizálási probléma esetében a leghatékonyabb

módszer az aktuális megoldás egy r sugarú környezetéből véletlenszerűen kiválasztani a következő megoldást. Ezt az alapelvet a jelenlegi feladatunkra alkalmazva, a következő folyamatábra szerint működő algoritmust kapjuk:



24. ábra: A sztochasztikus hegymászó alkalmazása a dolgozat problémájára

Az algoritmust többször lefuttatva, és a szomszédsági sugár értékeit változtatva a következő eredményeket kapjuk:



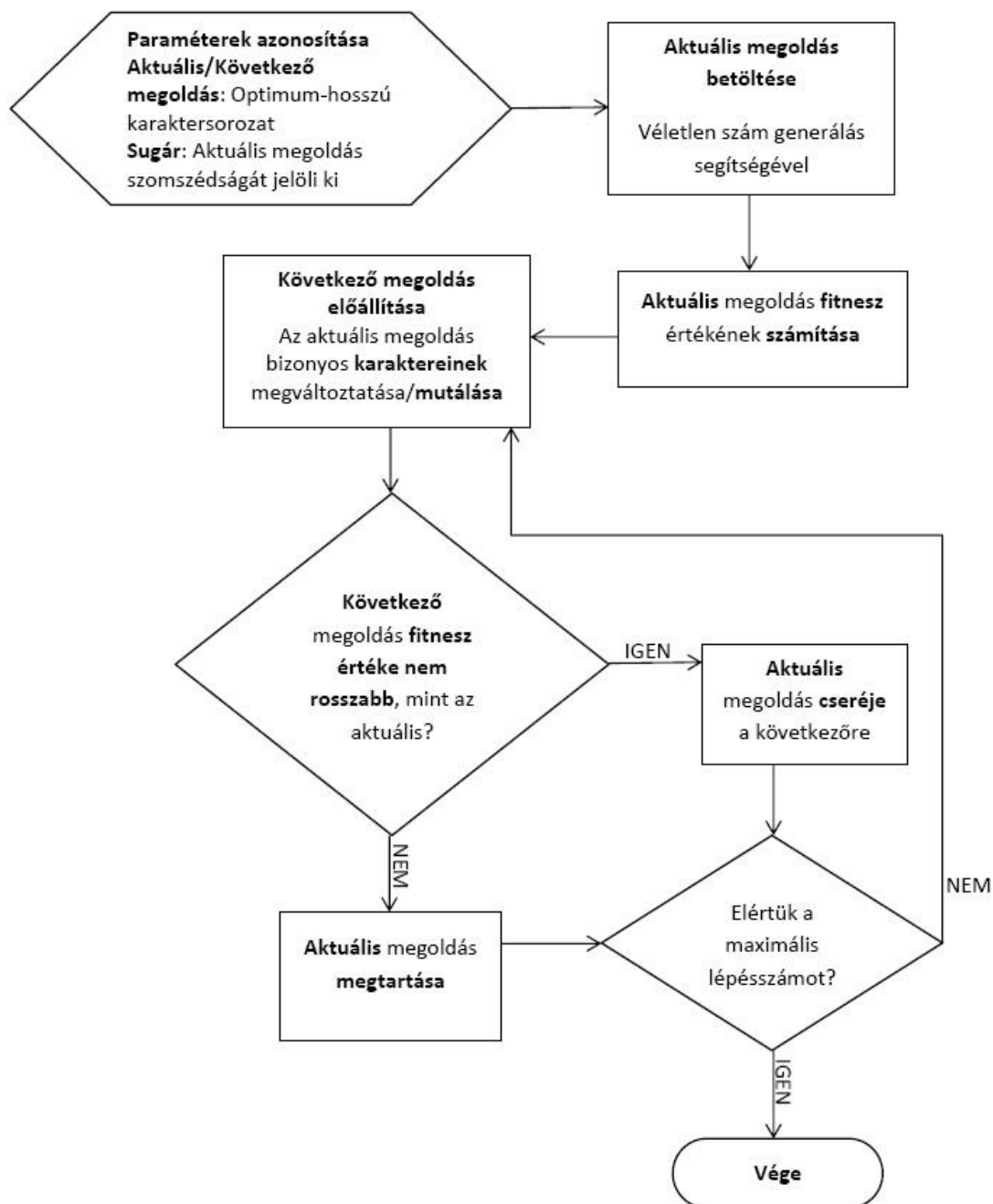
25. ábra: A szomszédsági sugár változtatásának eredményei

Az eredményeket vizsgálva a konklúziónk sajnos az, hogy az algoritmus belátható lépésszámon belül nem képes optimális megoldást találni. A „leghatékonyabb” sugárnak az $r = 5$ érték mutatkozik, de ezt alkalmazva is a keresett értéktől átlagosan csak 8 egység távolságra lévő megoldásokat talál az algoritmus, azaz lokális optimumban ragad.

Azonban ez a kimenetel nem volt váratlan, hiszen a problémánk nem fogható fel klasszikus matematikai függvényoptimalizálásként. Az adott karaktersorozat megtalálásához nem szabad egy egész megoldást elvetni, és egy r sugarú környezetében újat keresni, hiszen lehet, hogy az már majdnem a keresett szövegünk, csupán egy-két karakter nem volt megfelelő, elég lett volna csak azokon módosítani. Mint ahogyan a genetikusan programozási probléma esetében sem szabad egy egész programot elvetni, előbb az egyes szubrutinjainak működését kell vizsgálni.

A fenti tapasztalatokból kiindulva egy új szomszédsági függvényt kell tehát definiálnunk. Egyelemű populáció esetén korábbi metaheurisztikáinkban (HS és IHS) a lehetséges megoldásunk módosítására a mutációs operátort használtuk. Sőt, azt a következtetést is le tudtuk vonni, hogy egy olyan mutációs operátor a legcélravezetőbb, ami a módosításhoz nem veszi figyelembe a korábbi állapotot, hiszen karakterszintű módosítás esetén nincsen jelentősége az előző megoldásnak. Tehát tulajdonképpen az egyelemű populációval dolgozó harmónia kereső algoritmus (1-HMCR) valószínűséggel bekövetkező teljesen véletlen hangot generáló mutációs operátort választjuk szomszédsági függvénynek.

Ebben az esetben a következőképpen változik a sztochasztikus hegymászó algoritmus folyamatábrája:



26. ábra: A módosított sztochasztikus hegymászó folyamatábrája

A folyamatábrát megvizsgálva láthatjuk, hogy tulajdonképpen egy olyan egyelemű populációval dolgozó harmónia kereső algoritmust kaptunk, aminek a HMCR valószínűsége 0, azaz az $1 - \text{HMCR}$ értéke 1, tehát mindig a teljesen új hangot (karaktert) generáló ágba lép bele, és ha az így keletkezett harmónia nem rosszabb, mint az eddigi, akkor frissíti a memóriát.

Egy ilyen algoritmusnak a hatékonyságvizsgálata pedig már megtörtént a harmónia kereső algoritmus esetében. Ráadásul pont azt a következtetést sikerült levonni, hogy az egyelemű

populációval működő harmónia keresés pont a 0 HMCR valószínűségű esetben működött a leghatékonyabban. Futásideje is ennek az algoritmusnak volt a legjobb, 0,0047 másodperc. Adódik tehát a kérdés, hogy valóban a legegyszerűbb metaheurisztika szolgáltatja a leghatékonyabb megoldást a feladatunkra, vagy esetleg ezt is lehet még valamiféleképpen javítani?

Nos, a javítás lehetősége mindenképpen fennáll, mivel a sztochasztikus hegymászó algoritmus szintén egy másik metaheurisztikának, a szimulált hűtésnek egy leegyszerűsített változata.

9. Szimulált hűtés

A szimulált hűtésen alapuló optimalizálási technikák alapjait 1983-ban alkotta meg Kirkpatrick, Gelatt és Vecchi. Optimalizálási algoritmusuk a fémek edzésének folyamatát veszi alapul.

Adott egy test, amit alapállapotba akarunk hozni, ahol az anyagi részecskék jól strukturált kristályrácsba rendeződnek, és a test belső energiája minimális. Először felmelegítjük a testet, hogy az megolvadjon és a részecskék szabadon mozoghassanak, majd lassan lehűtjük, hogy a részecskék az alapállapot szerint elrendeződjenek. Ha az olvasztott anyag nem elég meleg, vagy a hűtést nem megfelelő ütemben végzik, akkor a kristályrácsba rendeződés az alapállapot elérése előtt megakad.

Az algoritmus maga alapvetően a sztochasztikus hegymászból indul ki. Egyelemű populációval dolgozunk, azaz kijelölünk egy kezdeti aktuális megoldást, és egy szomszédsági függvényt követve járjuk be a keresési teret. Ha egy megoldás nem rosszabb az aktuális megoldásnál, akkor kicseréljük az aktuális megoldást erre az új megoldásra.

A szimulált hűtés futása viszont nem áll meg itt. Az algoritmus azt mondja, hogy a keresési tér hatékonyabb, részletesebb bejárása érdekében bizonyos valószínűséggel elfogadhatunk az aktuálisnál rosszabb megoldásokat is, hogy ezekre meghívva a szomszédsági függvényt a keresési tér olyan megoldásait is megtaláljuk, amelyeken a sztochasztikus hegymászó „átugrik”. A rosszabb megoldás elfogadásának valószínűségét a Boltzmann-feltétel határozza meg:

$$e^{\frac{-(\text{fitnesz}(\text{következő}) - \text{fitnesz}(\text{aktuális}))}{T}} > \text{random}(0; 1)$$

Röviden, ha az e -t a megfelelő hatványra emelve nagyobb értéket kapunk, mint egy 0 és 1 közötti véletlen szám, akkor elfogadhatunk az aktuális megoldásnál rosszabb értéket is új

aktuális megoldásnak. A fitness értékek különbsége szabályozza, hogy az új aktuális megoldásnak elfogadott rosszabb megoldás ne legyen elfogadhatatlanul rossz.

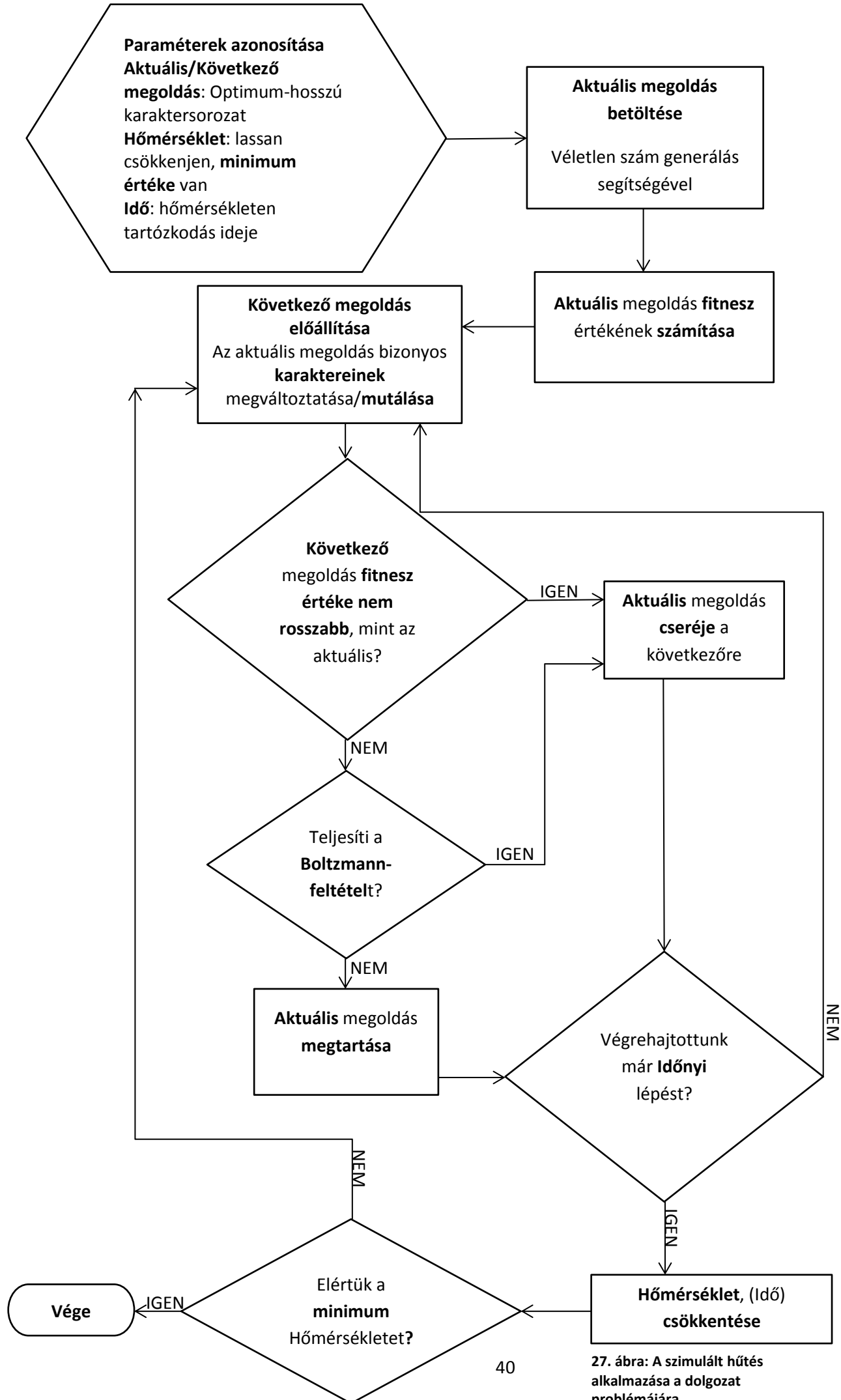
Ezen a ponton lép be a szimulált hűtés legfontosabb paramétere, a T , azaz a hőmérséklet. Ez a hőmérséklet egy előre megadott értékről indul, és szigorúan csökken minden lépésben, amíg el nem éri a minimum hőmérsékletet.

Mivel T folyamatosan csökken, ezért ha jól választottunk kezdő hőmérsékletet, akkor az algoritmus a futás előrehaladtával egyre kisebb valószínűséggel fogad el az aktuálisnál rosszabb megoldást.

Ezen kívül még egy „idő” paramétert is meghatározhatunk az algoritmushoz, ami az egy adott hőmérsékleten tartózkodás idejét (azaz hány ciklusban fusson le az algoritmus azonos hőmérséklettel) jelenti.

A hőmérséklet és az idő együttesen határozzák meg a hűtési ütemtervet, és azzal együtt a rosszabb megoldások elfogadásának valószínűségét. Tehát ezt az ütemtervet kell szabályozni úgy, hogy a leghatékonyabban járjuk be a keresési teret.

A feladatunkra az algoritmust a sztochasztikus hegymászó paraméterezéséből kiindulva alkotjuk meg, tehát szomszédsági függvénynek az egy karaktert teljesen véletlenül módosító mutációs operátort választjuk:



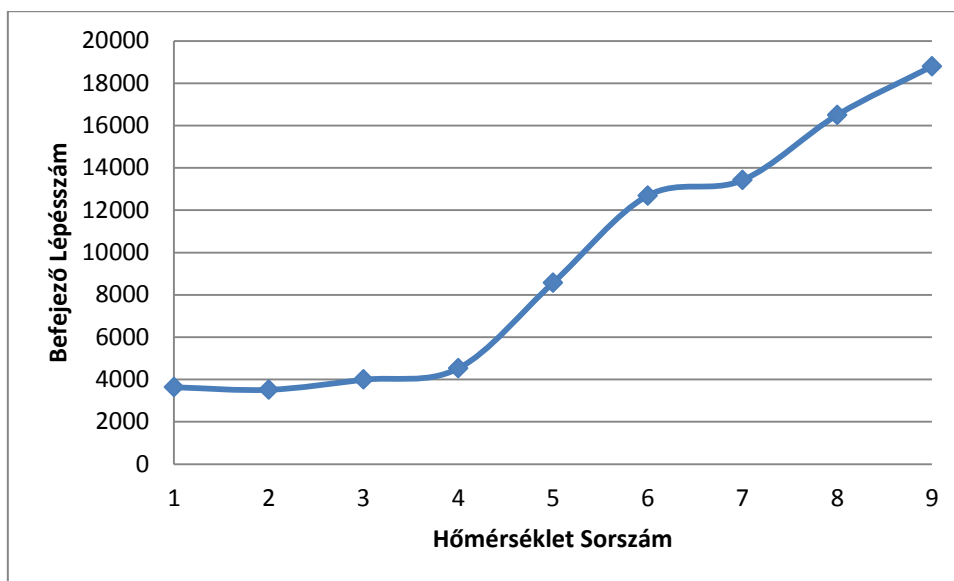
27. ábra: A szimulált hűtés alkalmazása a dolgozat problémájára

Ezen algoritmus hatékonyságvizsgálatát végezzük el a következőkben a hőmérséklet és idő paraméterek változásának függvényében, és próbáljuk megtalálni az ideális hűtési ütemtervet. A hatékonyság mértékegysége a saját paraméterek vizsgálatakor az optimum megtalálásához szükséges lépésszám, a futásidő ezzel egyenesen arányos.

Természetesen az egyes paraméterek hatékonyságvizsgálatát a többi paraméter változatlanul hagyásával, és egy többelemű minta átlagaként végezzük el.

9.1. Kezdeti hőmérséklet

A paraméter értékeit változtatva a következő futási eredményeket kapjuk:



28. ábra: A kezdeti hőmérséklet változtatásának eredményei

A grafikon áttekinthetősége miatt az egyes értékeket sorszámmal jelöljük meg, az ezekhez a sorszámokhoz tartozó konkrét értékeket a következő táblázat tartalmazza:

1. táblázat: A 28. ábra jelöléseinek magyarázata

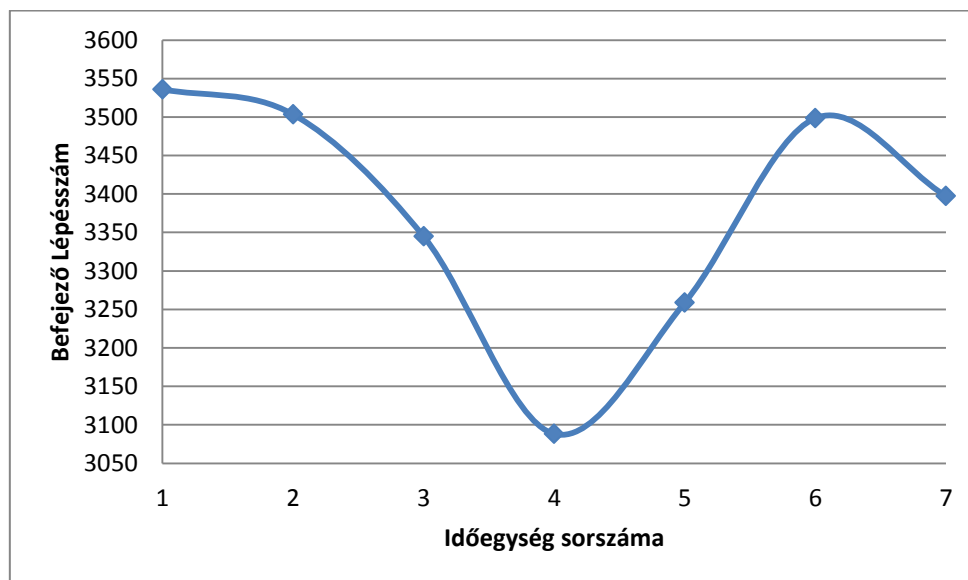
Hőmérséklet Sorszám	Hőmérséklet
9	1000
8	100
7	10
6	5
5	1
4	0,5
3	0,1
2	0,01
1	0,001

Az eredményeket elemezve láthatjuk, hogy a feladatunkhoz nem szükséges túl hosszú hűtési folyamat, a keresési tér (kis-és nagybetűs angol karakterek, valamint alapvető írásjelek) nem

olyan nagy, hogy ezt igényelné. Láthatjuk, hogy az első három kezdeti érték nagyjából azonos hatékonyságot eredményez, ezen kezdeti értékekkel tudjuk elérni a legideálisabb hűtési ütemet.

9.2. Hőmérsékleten tartózkodás ideje

A hőmérsékleten tartózkodás idejét változtatva a következő eredményekhez jutunk:



29. ábra: A hőmérsékleten tartózkodás idejének változtatásának eredményei

A könnyebb áttekinthetőség miatt ismételten sorszámokat alkalmazunk, a konkrét értékeket az alábbi táblázat szemlélteti:

2. táblázat: A 29. ábra jelöléseinek magyarázata

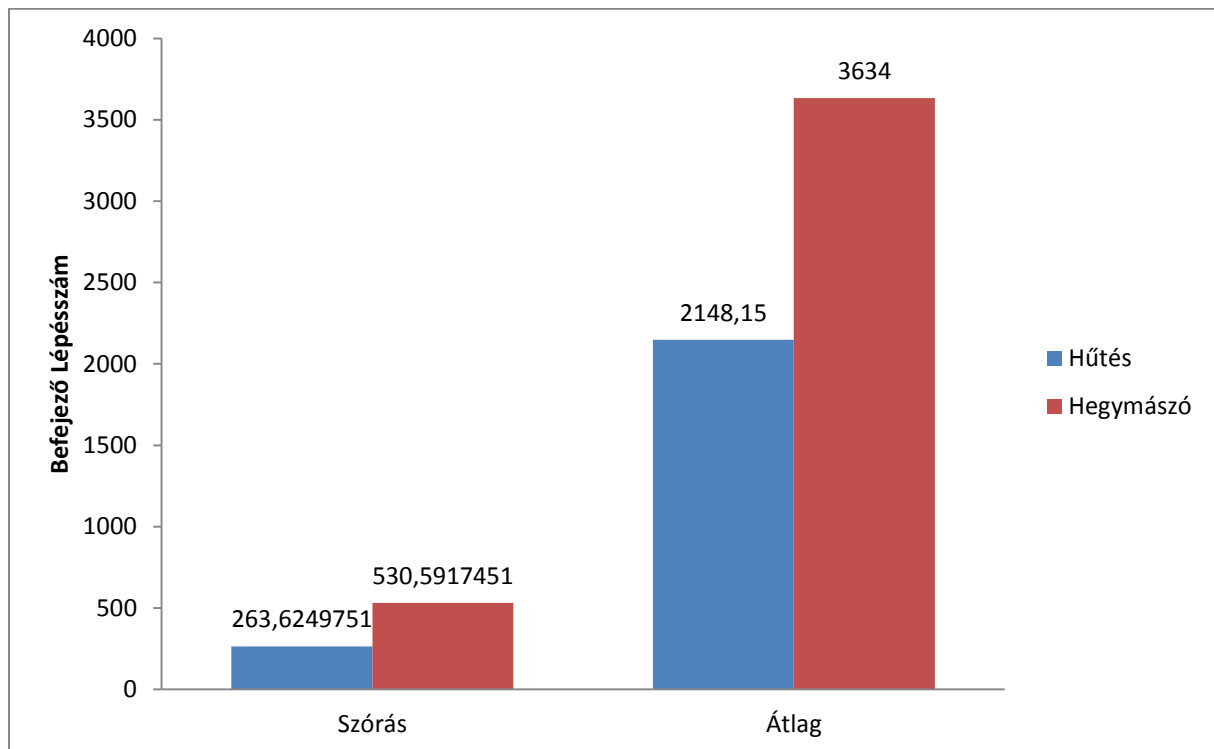
Időegység sorszáma	Időegység
7	1000
6	100
5	50
4	10
3	5
2	3
1	1

Az optimum megtalálásához szükséges lépésszám és a hőmérsékleten tartózkodás ideje között komolyabb kapcsolat nem áll fenn. Többszörös futtatások során csupán az volt megfigyelhető, hogy a 10-es időegység jobb eredményeket hozott a többi értéknél. Ez adja tehát az ideális hűtési ütemet.

10. Sztochasztikus hegymászó és Szimulált hűtés

Miután megtaláltuk az ideális hűtési ütemtervet, alkalmazzuk is, a sztochasztikus hegymászóval való összehasonlításra. A hegymászó eredményeit a 0 HMCR valószínűségű egyelemű populációt használó harmónia keresés eredményeivel azonosnak tekintjük, hiszen a kettő ugyanaz.

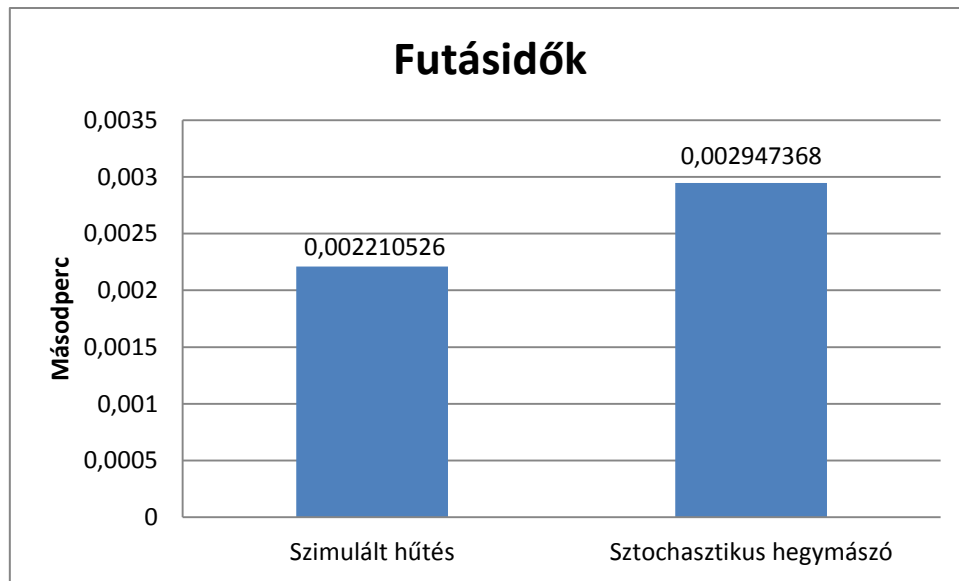
Az ideális beállításokat alkalmazva a szimulált hűtésre a következő eredményre jutunk:



30. ábra: A sztochasztikus hegymászó és szimulált hűtés lépésszámainak összehasonlítása

A hűtés megfelelő paraméterezésével (kellően lassú hűtést eredményező paraméterekkel) és szomszédsági függvénynek a HS-ben alkalmazott karakter (hang) szintű módosítást alkalmazva, megjavítható a Hegymászó lépésszáma, így a futásideje is. Ráadásul a lépésszám értékek szórása is kisebb, mint a Hegymászó esetében. Ez abból adódik, hogy a szimulált hűtés a Boltzmann-feltételnek köszönhetően kiegyensúlyozottabban, nagyobb lefedettséggel képes a keresési tér „feltérképezésére”, így nagyobb megbízhatósággal szolgáltatja az optimális megoldást.

Ezek után vizsgáljuk meg a két algoritmus futásidejét is a legjobb hatékonyságot eredményező beállításokkal:



31. ábra: A sztochasztikus hegymászó és a szimulált hűtés futásideje

A futási idő közel azonos, hiszen hiába javít sokat a lépésszámon a szimulált hűtés, a keresési tér hatékonyabb bejárásához ki kell bővítenie a sztochasztikus hegymászó műveleteit (Boltzmann-feltétel), ami plusz processzoridőt igényel. Viszont a szimulált hűtés futásideje egyáltalán nem rosszabb, sőt egy kicsivel rövidebb is, mint a hegymászó esetében. Továbbá hatékonyság szempontjából a sztochasztikus hegymászó eredményeit „rontja” a nagy szórása, amit már a lépéseknél is megfigyelhettünk.

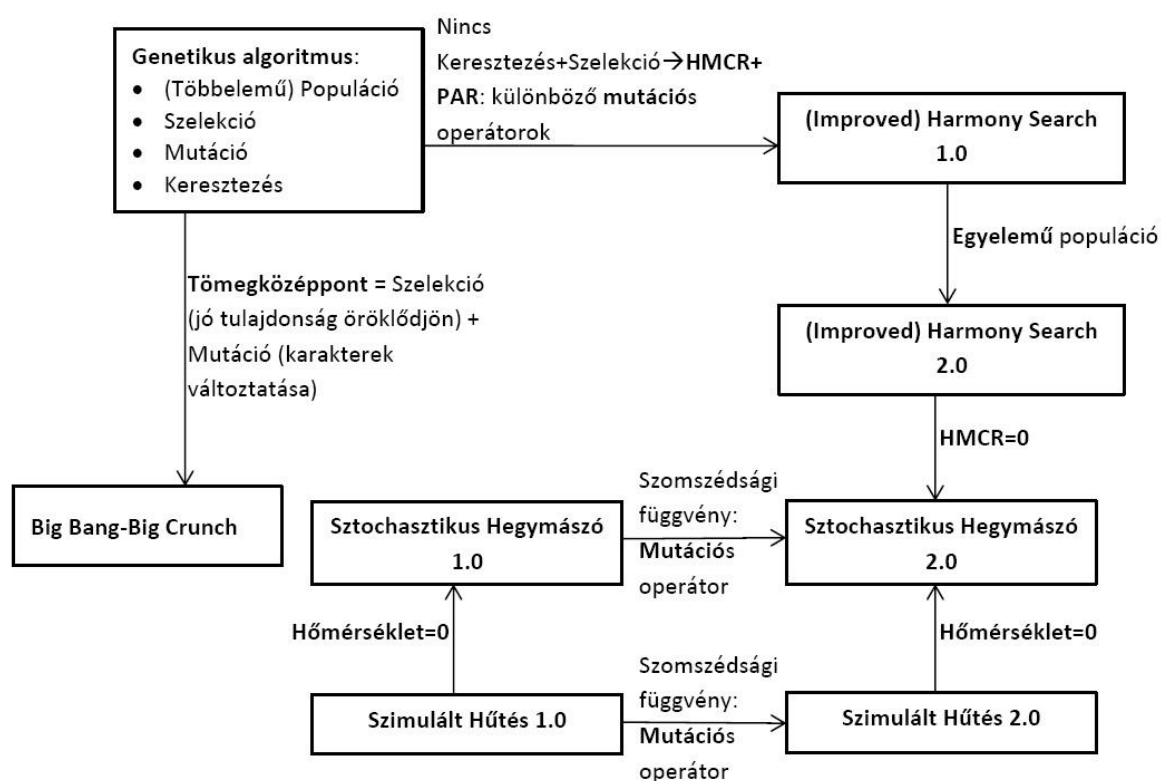
Összegezve, a szimulált hűtés legjobb hatékonyságot eredményező beállításával az algoritmus futásideje az összes eddigi metaheurisztika eredményét felülmúlja.

11. Összegzés

A dolgozatban bemutattuk a választott probléma (adott karaktersorozat előállítása) természetét, jellemzőit, valamint hasonlóságát a genetikus programozási feladattal. Ezután különböző metaheurisztikus megoldásokat alkalmaztunk a feladatra. Ezzel alapvetően nem a feladat megoldása volt a célunk, hanem annak vizsgálata, hogy a választott metaheurisztika miként illeszthető az adott problémánkhoz. A dolgozat tárgya tehát maga az illesztés folyamata, és annak komplexitásának bemutatása volt. A metaheurisztikus algoritmusokat általában bonyolult, NP-nehéz feladatok esetében szokás alkalmazni. A dolgozatban, mi mégis egy szándékoltan egyszerű problémát választottunk, hogy megmutassuk még egy ilyen feladat esetén is komoly hatékonyságvizsgálatokat igényel a megfelelő metaheurisztika kiválasztása. Bemutattuk, hogy ez a metaheurisztikák „meta” jellegéből adódik, azaz abból a tulajdonságukból, hogy különböző heurisztikus keresési eljárásokat próbálnak meg irányítani

általában valamiféle természeti jelenség alapján. Tehát ezen felsőbb, meta szint megfelelő beállítása (egyedek, operátorok, stb. megválasztása) nagymértékben növeli vagy csökkenti az algoritmus hatékonyságát, sőt képes a probléma megoldhatóságára is hatást gyakorolni.

A feladat megoldását először a Koza 1992-es tanulmányában is alkalmazott, és más kutatók által is gyakran használt genetikus algoritmus alkalmazásával oldottuk meg. Ezek után a genetikus algoritmus - mint kiindulási alap - eredményeit próbáltuk meg különböző más metaheurisztikák segítségével javítani. A további metaheurisztikákhoz, mint a genetikus algoritmus jobb hatékonyságot eredményező alternatíváihoz a következő gondolatmeneten keresztül jutottunk el:



32. ábra: A különböző metaheurisztikák kapcsolatai

Az ősrobbanás algoritmus a legkézenfekvőbb alternatívának a genetikus algoritmust tekintve kiindulási alapnak, de ez egy „zsákutcás” megoldásnak bizonyult egy ilyen természetű (matematika nyelvén nehezebben megfogható) feladat esetében. Ezek után a harmónia kereső algoritmust alkalmazva „egyenes út” vezetett el a legjobb hatékonyságot mutató szimulált hűtés felé, ahol a szomszédsági függvényt egy mutációs operátorként definiáltuk (Szimulált hűtés 2.0).

A dolgozat gondolatmenetét végigjárva világossá válik, hogy a bemutatott metaheurisztikák mind egymás módosításai valamilyen értelemben. Ez elég érdekes, tekintve, hogy különböző

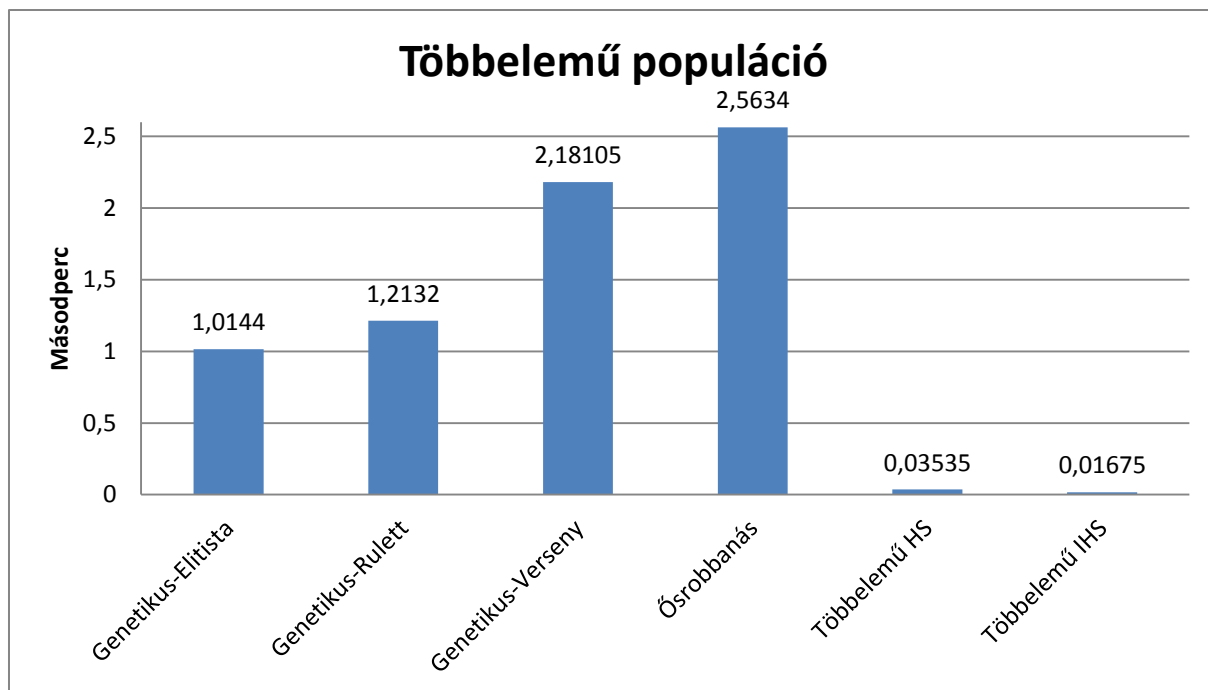
természeti jelenségeket vesznek alapul, sőt a harmónia kereső algoritmus analógiája nem is a természetből veszi a mintát, hanem a zeneszerzés folyamatából. Azonban a metaheurisztikák egy közös jellemzője, hogy a körülöttünk lévő anyagi világból veszik át az alapötleteket, folyamatokat. Ez a valós világ pedig optimalizáció központú, így a metaheurisztikák rokon vonásokat mutathatnak egymással, hiszen az összes természeti jelenség mögött ugyanaz az optimalitásra való törekvés húzódik meg.

Éppen ezért a metaheurisztikák kutatásában a legmeghatározóbb irányvonal jelenleg a különböző algoritmusok módszereinek, operátorainak vegyítése, kombinálása. A dolgozat során is ezt az utat követtük, hiszen a legjobb hatékonyságot mutató szimulált hűtés algoritmus alapvető keresési operátorával (r sugarú szomszédsági függvény) még az optimális megoldás megtalálására sem volt képes. Azonban ha a harmónia kereső algoritmus 1-HMCR valószínűséggel bekövetkező mutációs operátorát alkalmaztuk, akkor futásideje ezen algoritmusnak a legrövidebb. Viszont ha az Ősrobbanás algoritmusában kombináljuk a genetikus algoritmus szelekciós és mutációs operátorát, akkor rontottunk a hatékonyságon egy ilyen (a matematika nyelvén nem jól megfogható) problémát vizsgálva.

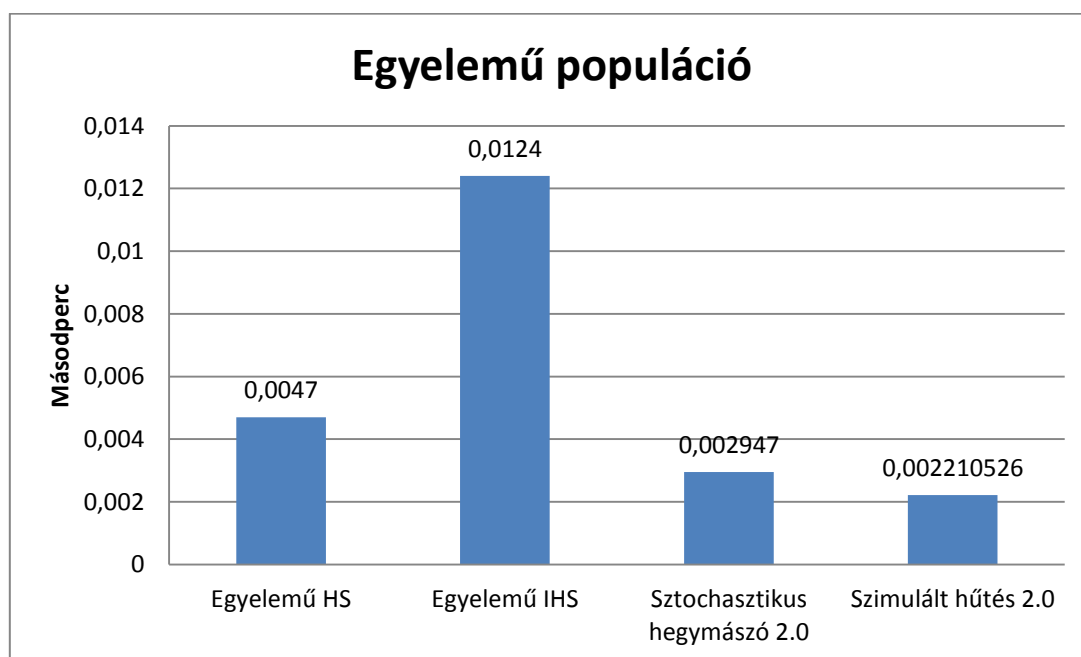
Tehát a metaheurisztikáknak ez a fajta kutatási iránya nagy körütekintést valamint alapos hatékonyságvizsgálatokat igényel a különböző típusú optimalizálási problémák esetében, még egy ilyen, első ránézésre igen egyszerű feladat esetén is.

Éppen ezért a jövőben tervezem hasonló hatékonyságvizsgálatok folytatását más természetű, NP-nehéz problémákon (pl.: kínaipostás-probléma), újabb metaheurisztikákat (hangyaboly algoritmus, méh algoritmus, részecske zajlás) is alkalmazva, kombinálva.

Végezetül egy összefoglaló grafikonnal zárnánk az egyes metaheurisztikák legjobb hatékonyságot eredményező beállításaikkal keletkező futásidejéről (azon metaheurisztikákat, melyek nem találtak optimális megoldást belátható időn belül, nem jelenítjük meg):

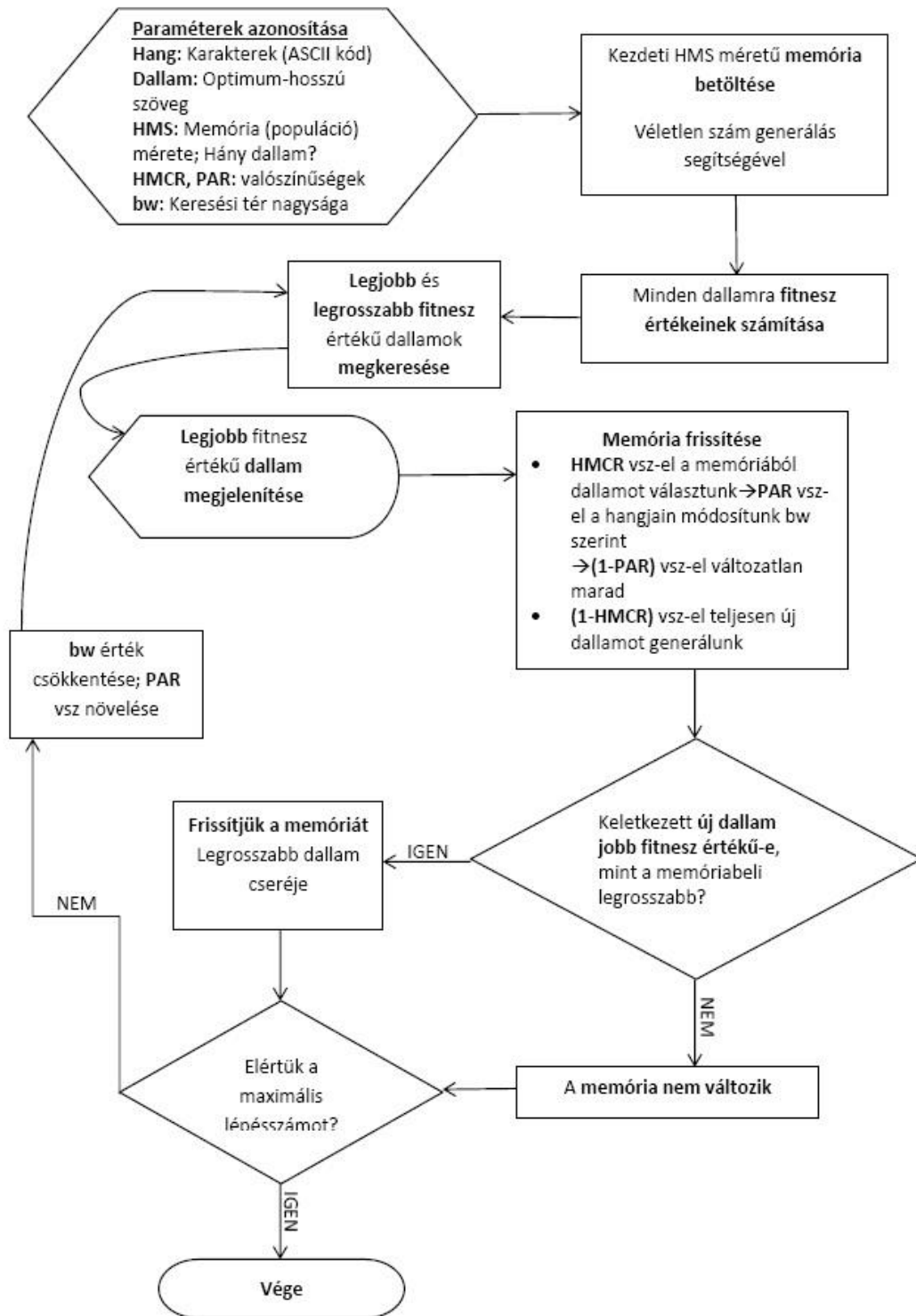


33. ábra: A többelelemű populációt alkalmazó metaheurisztikák futásideje

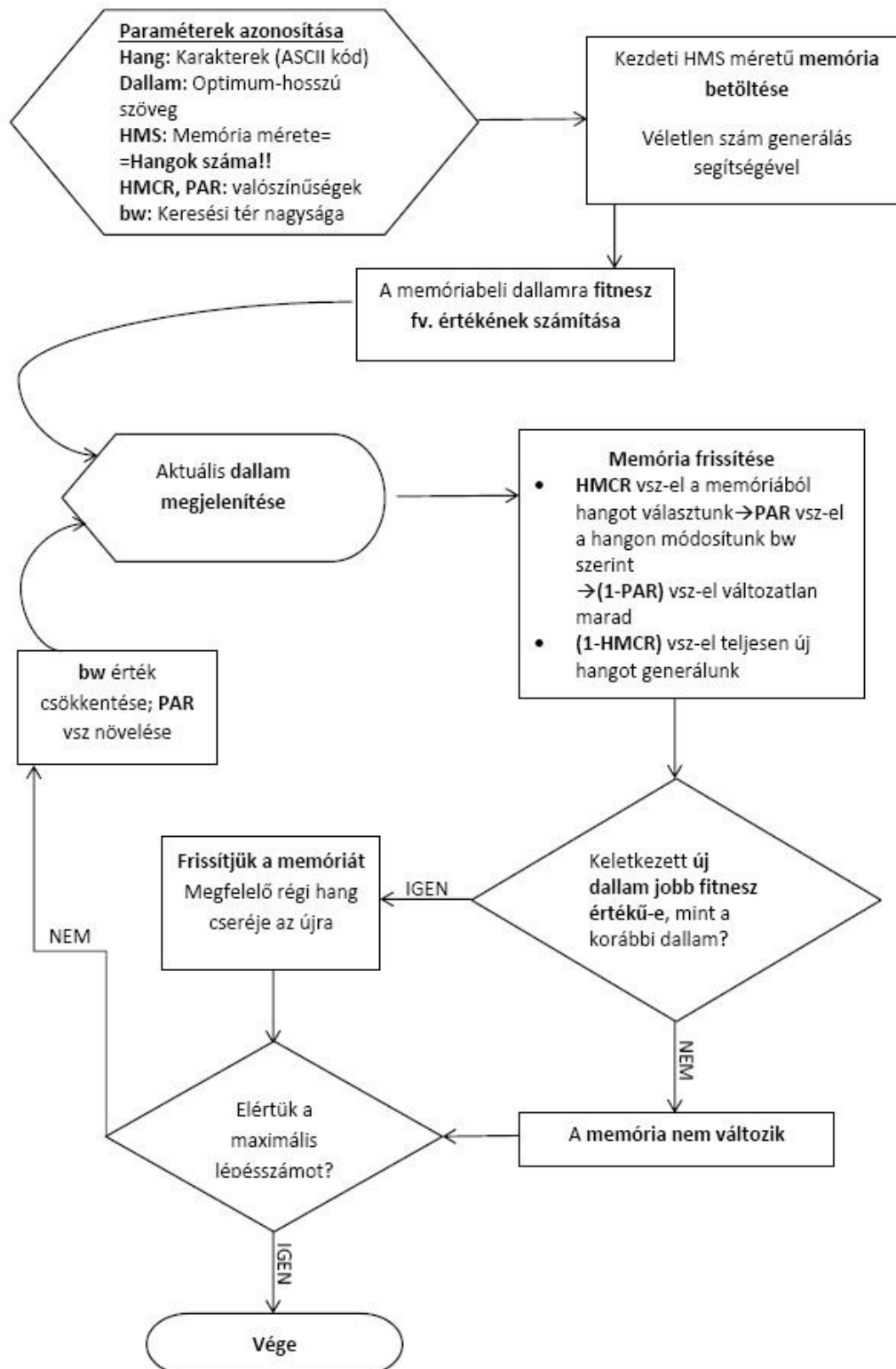


34. ábra: Az egyelemű populációt alkalmazó metaheurisztikák futásideje

Mellékletek



35. ábra: A fejlesztett harmónia kereső algoritmus alkalmazása a dolgozat problémájára



36. ábra: Az egyelemű fejlesztett harmóniakereső algoritmus alkalmazása a dolgozat problémájára

Irodalomjegyzék

Erol, O., K., Eksin, I. (2006) : A new optimization method: Big Bang–BigCrunch, *Advances in Engineering Software*, 37, 106-11.

Goldberg, D. E., (1989) : *Genetic Algorithms in Search, Optimization and Machine Learning*, Boston, Kluwer Academic Publishers

Holland, J. H.,(1975) : *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor

Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., (1983) : Optimization by Simulated Annealing, *Science*, 220 (4598), 671-680.

Koza, J., R. (1992) : *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press

Láng, B. (2010) : A nettó jelenérték maximalizálása az erőforráskorlátos projekt ütemezés során - egy új harmónia kereső heurisztika, doktori értekezés, Pécsi Tudományegyetem

Lee, K. S., Geem, Z. W. (2005) : A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice. *Computer Methods in Applied Mechanics and Engineering*, 194, 3902–3933.

Mahdavi, M., Fesanghary, M., Damangir, E. (2007) : An improved harmony search algorithm for solving optimization problems. *Applied Mathematics and Computation*, 188, 1567-1579.

Osman, I. H., Laporte, G., (1996) : Metaheuristics: A bibliography. *Annals of Operations Research*, 63 (5), 511-623.

Tasnádi, A. (2006) : *Számítástudomány gazdaságinformatikusoknak*, AULA Kiadó

Vofl, S., Martello, S., Osman, I. H., Roucairol, C. (1999) : *Meta-Heuristics – Advances and Trends in Local Search Paradigms for Optimization*. Dordrecht, The Netherlands, Kluwer Academic Publishers

<http://www.nist.gov/>

<http://www.metaheuristics.net/>

<http://www.harmonysearch.info/>