



Metaheurisztikus algoritmusok lineáris regressziós modellek optimalizálására

Tudományos Diákköri Dolgozat

Budapesti Corvinus Egyetem

Gazdálkodástudományi kar

Számítástudományi Tanszék

Konzulens: Dr. Láng Blanka

adjunktus

Kovács László

Alapképzés

III. évfolyam

2014. március 11.

Tartalomjegyzék

A dolgozatban használt jelölések	3
1. Bevezetés.....	4
2. A regressziós modellek szelekciójának problémája	4
3. A hagyományos modellszelekciós eljárások.....	5
4. Metaheurisztikus algoritmusokról általában.....	7
5. Genetic Algorithm for Regressors' Selection (GARS)	11
5.1. Az általános genetikus algoritmus (GA).....	11
5.2. A célfüggvény megválasztása	13
5.3. Az algoritmus rekonstrukciója.....	18
5.4. A GARS célfüggvényének bővítése	24
6. Harmony Search for Regressors' Selection (HSRS).....	28
6.1. Az általános harmónia kereső algoritmus	28
6.2. A HSRS megalkotása	31
6.3. Az Improved HSRS	36
7. Összefoglalás	40
Mellékletek.....	43
Irodalomjegyzék	48

A dolgozatban használt jelölések

n	A lineáris regresszió alapjául szolgáló megfigyelések (minta elemeinek) száma . (Az adatbázis rekordjainak száma.)
m	A lineáris regressziós modellhez rendelkezésre álló magyarázó változók száma.
X_i	A lineáris regressziós modellhez rendelkezésre álló i -edik magyarázó változó vektor. ($i \leq m$)
Y	Az eredményváltozó vektor.
p	A lineáris regressziós modellben felhasznált álló magyarázó változók száma. ($p \leq m$)
$\tilde{X}_i$	A regressziós modellben felhasznált i -edik magyarázó változó vektor. ($i \leq p$)
β_i	Az $\tilde{X}_i$ vektor együtthatója a lineáris regressziós modellben. ($i \leq p$)
β_0	A lineáris regressziós modell tengelymetszete .
ε	A lineáris regressziós modell hibatagja . („véletlen tényező”)
$\hat{\beta}_i$	Az $\tilde{X}_i$ vektor együtthatójának OLS becslése a lineáris regressziós modellben. ($i \leq p$)
$\hat{\beta}_0$	A lineáris regressziós modell tengelymetszetének OLS becslése .
$\hat{Y}$	Az OLS-el előállított lineáris regresszióból becsült eredményváltozó vektor.
y_i	Az eredményváltozó vektor i -edik eleme. ($i \leq n$)
$\bar{y}$	Az eredményváltozó vektor elemeinek átlaga egy n elemű megfigyelés (minta) alapján.
$\hat{y}_i$	A lineáris regresszióból becsült eredményváltozó vektor i -edik eleme. ($i \leq n$)

1. Bevezetés

A dolgozat célja, hogy lineáris regressziós modellekben az eredményváltozót optimálisan magyarázó változók (regresszorok) körét és számát metaheurisztikus algoritmus segítségével meghatározzuk.

A lineáris regressziós modellek változóinak szelekciója egy NP-teljes probléma, a megoldások lehetséges száma m db lehetséges magyarázóváltozó esetén: $2^m - 1$, ha az üres halmazt nem számoljuk. (Minerva – Paterlini, 2002) Azonban, a jelenlegi statisztikai szoftvercsomagokban (pl. SPSS) csak lokális kereső eljárások állnak rendelkezésre a feladat megoldására (Forward és Backward elimináció). Ezen algoritmusok futása során az egyes regressziós modellek mindig csupán egy darab változóban térnek el (egy változót adunk hozzá vagy vonunk el). Tehát, a keresési tér nagy része feltérképezetlen maradhat, és az algoritmus lokális optimum felé konvergálhat.

Így a globális optimumok felkutatásához érdemes metaheurisztikákat alkalmazni a változók szelekciójára. A dolgozatban áttekintjük egy már létező, genetikus algoritmus alapú megoldás (Genetic Algorithm for Regressors' Selection) eredményeit, illetve hiányosságait. Ezek után egy saját, harmónia kereső megoldás (Harmony Search for Regressors' Selection) eredményeit és előnyeit (dupla célfüggvények kezelése, processzoridő felhasználás) mutatom be a genetikus algoritmushoz képest.

2. A regressziós modellek szelekciójának problémája

Legyen $X = \{X_1, X_2, \dots, X_m\}$ az m darab lehetséges magyarázó változó halmaza (egy n elemű mintában), és Y pedig legyen az eredményváltozó egy többváltozós lineáris regressziós modellben. Tegyük fel, hogy a regressziós modell, ami magyarázza az eredményváltozót, és a magyarázó változók közötti kapcsolatot a következő:

$$Y = \beta_0 + \beta_1 \tilde{X}_1 + \beta_2 \tilde{X}_2 + \dots + \beta_p \tilde{X}_p + \varepsilon.$$

Ahol, $\tilde{X} \subseteq X$ a $p \leq m$ darab kiválasztott magyarázó változó halmaza, és $\beta = \{\beta_1, \beta_2, \dots, \beta_p\}$ a magyarázó változók lineáris együtthatóinak a halmaza, ε pedig a hibateg. Ha a β halmazt a legkisebb négyzetek módszerével (Ordinary Least Squares method, **OLS**), $\hat{\beta} = \{\hat{\beta}_1, \hat{\beta}_2, \dots, \hat{\beta}_p\}$ halmazként megbecsüljük, akkor a következő feladat azon magyarázóváltozók megtalálása, amelyek a $\tilde{X} \subseteq X$ halmazt alkotják. Ezt a halmazt oly módon

kell meghatározni, hogy az előállított (megbecsült) $\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1\tilde{X}_1 + \hat{\beta}_2\tilde{X}_2 + \dots + \hat{\beta}_p\tilde{X}_p$ regressziós modell optimális legyen, olyan – később ismertetett – mutatószámok ($\bar{R}^2$, AIC , BIC , stb.) szerint, amik a modell magyarázóerejét vagy egységnyi mintaelemre jutó hibáját mérik.

Ezen kívül pedig az is fontos szempont, hogy az optimálisnak ítélt modellben csak szignifikáns változók szerepeljenek. Azaz, olyan változó ne szerepeljen a modellben, aminek az együtthatója (β_i) közel 0, tehát nem lehet egyértelműen eldönteni, hogy változása (ceteris paribus) növeli vagy csökkenti-e az eredményváltozó értékét. Ezt hipotézisvizsgálatok segítségével tudjuk ellenőrizni (parciális t és F próbák), együtthatónként (β_i) külön-külön. A vizsgálat során a nullhipotézis tartalmazza azt az állítást, hogy az adott magyarázó változó ($\tilde{X}_i$) együtthatója (β_i) nem szignifikáns, míg az alternatív hipotézis szerint a változó együtthatója szignifikáns. Tehát, a nullhipotézis elutasítását vizsgáljuk egy adott α szignifikancia szinten. Ha bármelyik magyarázó változó együtthatója nem szignifikánsnak bizonyul, nem tartjuk optimálisnak az adott modellt ilyen szempontból. (Hunyadi – Vita, 2006)

Amikor a vizsgált algoritmus célfüggvénye mind a két értékelési szempontot egyszerre figyelembe veszi (mutatószámok, illetve szignifikancia szerinti optimalizálás), akkor „dupla” célfüggvényként hivatkozunk rá a továbbiakban.

3. A hagyományos modellszelekciós eljárások

A regressziós modellek változóinak szelekciója tehát egy NP-teljes probléma, a megoldások lehetséges száma m db lehetséges magyarázóváltozó esetén $2^m - 1$, ha az üres halmazt nem számoljuk.

Azonban a különböző statisztikai szoftverekben (pl.: SPSS) csak olyan lokális keresési eljárások állnak rendelkezésünkre, mint a Forward vagy a Backward elimináció, melyek hipotézisvizsgálatok (parciális t vagy F próbák) sorozatával igyekeznek meghatározni a magyarázó változók optimális halmazát. Egy adott lépésben vagy folyamatosan szűkítjük a modellt a legkisebb szignifikanciájú változók elhagyásával (Backward), vagy a legnagyobb szignifikanciájú változó bevonásával (Forward). Az algoritmusok addig folytatják ezeket a műveleteket, amíg egy adott szignifikancia szint mellett (pl.: $\alpha = 5\%$) minden változó szignifikáns a modellben. Ugyan akkor a statisztikai szoftverekben rendszerint van lehetőség eltérő szignifikancia szintet beállítani a magyarázó változók modellbe történő bevonására,

illetve a modellből való kihagyásra. Például, az SPSS nevű szoftverben az alapértelmezett beállítás szerint a bevonási szint 5%, a kihagyási szignifikancia szint pedig 10%. Ezen beállítások mellett a Forward szelekció futása az 1. ábrán látható.

Model Summary				
Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	,975 ^a	,951	,951	1,8465371
2	,979 ^b	,958	,957	1,7265580
3	,981 ^c	,962	,960	1,6509723
4	,988 ^d	,976	,975	1,3209435
5	,990 ^e	,980	,979	1,1965530
6	,991 ^f	,982	,981	1,1456385
7	,992 ^g	,983	,982	1,1189287
8	,992 ^h	,984	,983	1,0925061

a. Predictors: (Constant), X1
b. Predictors: (Constant), X1, X8
c. Predictors: (Constant), X1, X8, X6
d. Predictors: (Constant), X1, X8, X6, X3
e. Predictors: (Constant), X1, X8, X6, X3, X5
f. Predictors: (Constant), X1, X8, X6, X3, X5, X11
g. Predictors: (Constant), X1, X8, X6, X3, X5, X11, X15
h. Predictors: (Constant), X1, X8, X6, X3, X5, X11, X15, X12

1. ábra: A Forward szelekció futása SPSS-ben, képernyőkép

Ezek után a Backward eliminációt is lefuttattuk a programban. Ekkor, a szignifikancia szintet egységesen (bevonási és elhagyási) 5%-ra állítjuk, mivel azon metaheurisztikus algoritmusokban is 5%-os szignifikancia szinten vizsgáljuk a magyarázó változókat, amelyek célfüggvényében a szignifikanciát is vizsgáljuk. A később bemutatott 5. és 7. táblázatokban a Backward és Forward algoritmusok eredményei ezen futtatásokból származnak.

Ha áttekintjük az 1. ábrát, ezen algoritmusok futása során az egyes regressziós modellek (az adott változóhalmazból álló lehetséges megoldások) mindig csupán egy darab változóban különböznek (egy változót adunk hozzá vagy vonunk el). Tehát, a keresési tér nagy része feltérképezetlen maradhat, és az algoritmusunk lokális optimum felé konvergálhat.

Különösen akkor, ha a regressziós modell lehetséges magyarázó változói között jelentős multikollinearitás áll fenn. Ez a jelenség azt jelenti, hogy a magyarázóváltozók redundánsak,

azaz egymást magyarázzák. Ekkor előfordulhat az, hogy X_1 változó, ha önmagában áll a regressziós modellben, az optimumhoz közelebb álló értéket ad valamely később ismertett mutatószámra ($\bar{R}^2$, AIC , BIC , stb.), mint ha X_2 változó állna önmagában a modellben. Ám, ha mind a kettő szerepel a modellben, akkor pl. a Backward algoritmus a hipotézisvizsgálat eredményeképpen mégis inkább X_2 -t preferálja, és X_1 -et kihagyja a modellből, tehát nem szignifikáns változó lesz. Ebben az esetben pedig már nem tárhatna fel a szelekciós eljárás olyan, az optimumhoz közelebb álló, megoldásokat, amik X_1 -et szerepeltetik. Ekkor érdemes az összes szóba jöhető regressziós modellt megvizsgálni, amit a metaheurisztikák igyekeznek biztosítani. (Kovács, 2008)

Éppen ezért lehet létjogosultsága egy ilyen feladatra metaheurisztikus megoldásokat illeszteni. Ezek az algoritmusok törekszenek a lehető legnagyobb részét feltérképezni egy adott keresési térnek, ami esetünkben az összes lehetséges regressziós modell megvizsgálását jelenti. Alapvetően megvizsgáljuk, hogyan is illeszthető egy adott metaheurisztika analógiája a regresszió optimalizálási feladatára, illetve, hogy ez az illesztés a metaheurisztikák milyen paraméterezése mellett optimális. A fő cél viszont a feladatot legjobban megoldó metaheurisztika megtalálása, aminek a processzoridő felhasználása is a minimálishoz legjobban közelítő.

4. 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
- gráfszínezés
- diszkrét esetű ütemezési problémák

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

Az NP-nehézség és a problémák alapvetően bonyolult matematikai modellezhetőségéből adódóan az ilyen feladatok megoldására a kutatók rendszerint heurisztikus algoritmusokat alkalmaznak. Az utóbbi 20 évben egy új fejezet nyílt a heurisztikus megoldások történetében. A kombinatorikus optimalizálási NP-nehéz 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 többeleműek. A többelemű 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 hármat alkalmazó genetikai 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 2. ábrával lehet szemléltetni:



2. ábra: Kétszintű heurisztikák

Forrás: Láng (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 és munkatársai, 1999, 8. o.)

„A metaheurisztika formálisan definiálható egy iteratív generációs eljárásként, 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, 1996, 513-514. o.)

„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 (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.”

Az első vizsgált metaheurisztika a kutatók által legismertebb, és leggyakrabban használt, a **genetikus algoritmus**. Nem meglepő módon ezt a rendkívül népszerű algoritmust már adoptálták is erre a feladatra. (Minerva – Paterlini, 2010) Ezt a létező megoldást tekintjük a vizsgálat kiindulópontjának, mivel az olasz kutatók által használt egyik adatbázist sikerült megtalálni, így jó összehasonlítási alapnak ígérkezett.

A következőekben tárgyalt metaheurisztikus algoritmusok futtatási környezete:

- Operációs rendszer: Windows 7 Ultimate 64 bit
- Fejlesztői környezet: Visual C# 2010 Pro
- CPU: Dual Core Intel Core i5, 2133 MHz
- RAM: 4GB DDR3 SDRAM

5. Genetic Algorithm for Regressors' Selection (GARS)

5.1. Az általános genetikus algoritmus (GA)

A genetikus algoritmust 1975-ben Holland fejlesztette ki, majd 1989-ben Goldberg továbbfejlesztette. (Goldberg, 1989) Az algoritmus alapgondolata a biológiai genetikán és Darwin „természetes kiválasztódás” elvén alapul. Az egyedek az egyes megoldások, melyeket az algoritmus 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. (A később részletesen is tárgyalt GARS algoritmus bináris kódolást használ az egyedek reprezentációjára: ha az adatbázis i -edik változó szerepel a modellben, akkor az m (lehetséges magyarázó változók száma) elemű bitsorozat i -edik tagja a regressziós modell része, akkor a bit az 1 értéket vesz fel, ha nem része, akkor 0-át.)

A három szint (populáció, egyed, gének) megválasztása után az algoritmus minden lépésben egy új populációt hoz létre a célfüggvény, azaz fitness értékek alapján (a fitness elnevezés alkalmazása a „célfüggvény érték” kifejezés helyett 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 $\varphi * c$ db egyedet mindig áthelyezi az új populációba, ahol c 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 vagy egy előre rögzített génnél elvágjuk. A vágás előtti géntulajdonságokat örökli az új egyed az egyik szülőtől, a vágás utániakat a másik szülőtől.

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. De olyan megoldás is elképzelhető, ahol egy kellően kis valószínűséggel akár az összes gén módosulhat.

5.2. A célfüggvény megválasztása

A hivatkozott kutatók (Minerva – Paterlini, 2010), genetikus algoritmusukban háromféle célfüggvény (fitnesz) értéket használtak fel az algoritmus eredményeképpen előálló megoldások értékelésére. A mutatók ugyan azon elvre épülnek: egyfelől „jutalmazza” a modell egységnyi hibájának csökkenését, másrészt büntetik, ha ehhez túl sok, fölösleges magyarázó változót (regresszort) használunk el.

Az első az Akaike Információs Kritérium/Akaike Information Criterion (*AIC*), melyet a cikkükben a következőképpen értelmeznek:

$$AIC = n * \ln(s_e^2) + 2 * p \quad (1)$$

A képletben n a megfigyelések száma (minta elemszáma), p az adott modellben lévő magyarázó változók darabszáma, s_e^2 pedig a korrigált reziduális variancia:

$$s_e^2 = \frac{SSE}{n - p - 1} \quad (2)$$

$$SSE = \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (3)$$

Ahol $\hat{y}_i$ a regressziós modellből becsült eredményváltozó értéke, míg y az eredményváltozó valódi értéke.

A második célfüggvény, ugyan azon jelöléseket használva a Bayes Információs Kritérium/ Bayesian Information Criterion (*BIC*), a cikk értelmezése szerint:

$$BIC = (n - p) * \ln\left(\frac{n * s_e^2}{n - p}\right) + p * \ln\left(n * \frac{s^2 - s_e^2}{p}\right) \quad (4)$$

$s^2 = \frac{\sum_{i=1}^n (y_i - \bar{y})^2}{n - 1}$, azaz az eredményváltozó teljes korrigált tapasztalati varianciája az n db megfigyelés alapján.

A harmadik lehetséges célfüggvény pedig a Schwarz Információs Kritérium/Schwarz Information Criterion (*SIC*). Hasonló jelölésekkel a cikk értelmezésében:

$$SIC = \ln(s_e^2) + \frac{p}{n} * \ln(n) \quad (5)$$

Mindhárom célfüggvényt minimalizálni kell az optimális változóhalmaz megtalálásához, hiszen alapvetően az egy tagra jutó modellből származó hibát (*SSE*) mérik.

Ezen kívül a szerzők a korrigált R^2 mutatót ($\bar{R}^2$) is számolják, az alábbi képletek szerint. A korábban bevezetett jelölések változatlanok.

$$SST = \sum_{i=1}^n (y_i - \bar{y})^2 \quad (6)$$

$$R^2 = 1 - \frac{SSE}{SST} \quad (7)$$

$$\bar{R}^2 = 1 - \frac{n-1}{n-p-1} * (1 - R^2) \quad (8)$$

Azonban a számítás kizárólag jelzésértékkel történik, célfüggvényként nem használják fel az algoritmusban. Egy modell akkor közelíti meg jobban az optimumot, ha a különböző információs kritériumok értéke minél kisebb, illetve, ha az $\bar{R}^2$ értéke minél nagyobb. Ez a mutató az információs kritériumokkal ellentétben nem az egységnyi elkövetett – modellből származó – hibát minimalizálja, hanem az eredményváltozó négyzetes szóródását (*SST*) kísérli meg minél inkább a regressziós modell segítségével magyarázni. Tehát, azt nézzük meg, hogy az *SST* mekkora része NEM származik a modell négyzetes hibájából (*SSE*). A fölösleges változók bevonásának büntetésére szolgál a korrekciós tag.

A modellek jobb ellenőrizhetősége érdekében a cikk írói az $n = 252$ elemű adatbázist megbontották 3 részhalmazra. Az első részhalmazt, a *tanuló halmazt* (az adatok első 40%-át, azaz a 1-101. elemeket) a regressziós együtthatók becslésére használják fel. A második halmazt, a *validációs halmazt* (második 40%, 102-202. elemek) az *AIC*, *BIC* és *SIC* értékek számítására tartják fenn, míg az elemek utolsó 20%-át, a *teszt halmazt* (203-252. elemek) az $\bar{R}^2$ mutató meghatározáshoz használják el.

Ugyan ezen elveket követve próbáljuk rekonstruálni először azt a regressziós modellt, ami az adatbázis mind a 16 változóját tartalmazza, tehát: $p = 16$. Hiszen, ez még a különböző kereső algoritmusok (GARS, Backward, Forward) futásától független eredmény. Azonban az implementációnk, és a cikk eredményei lényegesen eltértek. Végül, arra a következtetésre jutottunk, hogy sajnos nem lehet teljes egészében rekonstruálni a cikkben közölt számításokat. Ennek okai a következők.

1. Az egyik adathalmazt nem sikerült felkutatni, csak részben, és a változók értelmezéséhez nem sikerült dokumentációt találni.
2. A mutatók értéke jelentősen függ az adatok rendezési sorrendjétől. Például lehetséges az adatokat egy fordított sorrendbe rendezni, tehát a részhalmazokat egy fordított sorrendre definiálni. Azaz, az együtthatók meghatározásához használt „első” 40% halmaza most a 152-252. elemeket jelenti, míg az információs kritériumok számításához való „második” 40% az 51-151. elemeket és az $\bar{R}^2$ számításhoz használt „utolsó” 20% pedig az 1-50. elemeket reprezentálja. Az eredmények AIC és $\bar{R}^2$ értékére az 1. táblázatban láthatók.

1. táblázat: Az eltérő halmazbontások különböző eredményei

	Eredeti sorrend	Fordított sorrend
AIC (1) képlet	43,862	-0,753
$\bar{R}^2$ (8) képlet	0,961	0,951

3. A használt adatbázis verzió még adatrögzítési hibákat tartalmazott, amit a cikk szerzői lehetséges, hogy már korrigáltak, de az is lehetséges, hogy nem. (Johnson, 1996)

Ezek után utánanéztem a különböző információs kritériumok értelmezésének a szakirodalomban, és eltérő értelmezéseket is találtam rájuk. Ezen értelmezéseket Hajdu Ottó: Statisztikai adatok ökonometria elemzése (Hajdu, 2013) c. egyetemi jegyzete alapján foglaljuk össze. Az eddigiekkel megegyező jelöléseket használva.

Tehát, Akaike Információs Kritériuma/Akaike Information Criterion (AIC):

$$AIC = \ln\left(\frac{SSE}{n}\right) + \frac{2 * p}{n} \quad (9)$$

A Schwarz- Bayes Információs Kritérium/Bayesian-Schwarz Information Criterion (*SBC*):

$$SBC = \ln\left(\frac{SSE}{n}\right) + \frac{2p}{n} * \ln(\sqrt{n}) \quad (10)$$

A Hannan-Quinn Információs Kritérium/Hannan-Quinn Information Criterion (*HQC*):

$$HQC = \ln\left(\frac{SSE}{n}\right) + \frac{2p}{n} * \ln(\ln(n)) \quad (11)$$

Ezen információs kritériumokat a fordított logikájú részhalmazok szerinti bontásra alkalmazva (ugyanis ezzel a logikával kaptam $\bar{R}^2$ -re olyan eredményt, ami alig tér el a cikkben közölt eredménytől) kaptam a cikk íróival megegyező nagyságrendű eredményeket, ahol az eltérés már magyarázható a fentebb részletezett esetleges eltérő adatbázis verziókkal. Az alkalmazott részhalmazra bontási logika, és az így kapott eredmények a 2. illetve a 3. táblázatban szerepelnek.

2. táblázat: A "fordított" részhalmazokra bontás

Részhalmazok	Elemek
első 40%	152-252.
második 40%	51-151.
utolsó 20%	1-50.

3. táblázat: A különböző információs kritérium képletek eredményei

Hajdu formulák szerint:	Értékek	Cikkben	Értéke
AIC (9) képlet	1,031	AIC (1) képlet	1,042
Schwarz-Bayes (10) képlet	1,446	SIC (5) képlet	1,456
Hannan-Quinn (11) képlet	1,199	BIC (4) képlet	1,941

Egy ökonometria szakértővel való konzultáció során kiderült, hogy az információs kritériumok eltérő képletei tulajdonképpen pont a kezelhető nagyságrend problémájából származnak. Ha egy számítási módszer szerint az egyes modellek jóságát mérő értékek között csupán kis eltérés állapítható meg, akkor a büntetőtag megfelelő változtatásával más nagyságrendűvé tehetők az eredmények, így feltűnőbb lehet a különbség két modell közt, és nem csupán tizedes jegyeken múlik a döntés.

A részhalmazok megválasztásának sorrendje is tetszőleges lehet, ha a különböző modellek összehasonlításakor következetesen járunk el. Azaz, ha ugyan azok a részhalmazaink, mintáink az egész kutatás, modellkeresés során.

Ezek után, végül úgy döntöttünk, hogy a birtokomban lévő adatbázis verzió szerinti eredeti sorrend szerint választom ki a 40-40-20%-os bontását az adatoknak. Tehát a vizsgálatunkban a halmazok a 4. táblázat szerint alakulnak.

4. táblázat: A dolgozat során alkalmazott részhalmazokra bontási logika

Részhalmazok	Elemek
első 40% (tanuló)	1-101.
második 40% (validációs)	102-202.
utolsó 20% (teszt)	203-252.

A továbbiakban ezzel a halmazbontással, és a cikk eredményeitől kevésbé eltérő, Hajdu-féle formulákkal dolgozunk tovább a célfüggvények (információs kritériumok) számításakor.

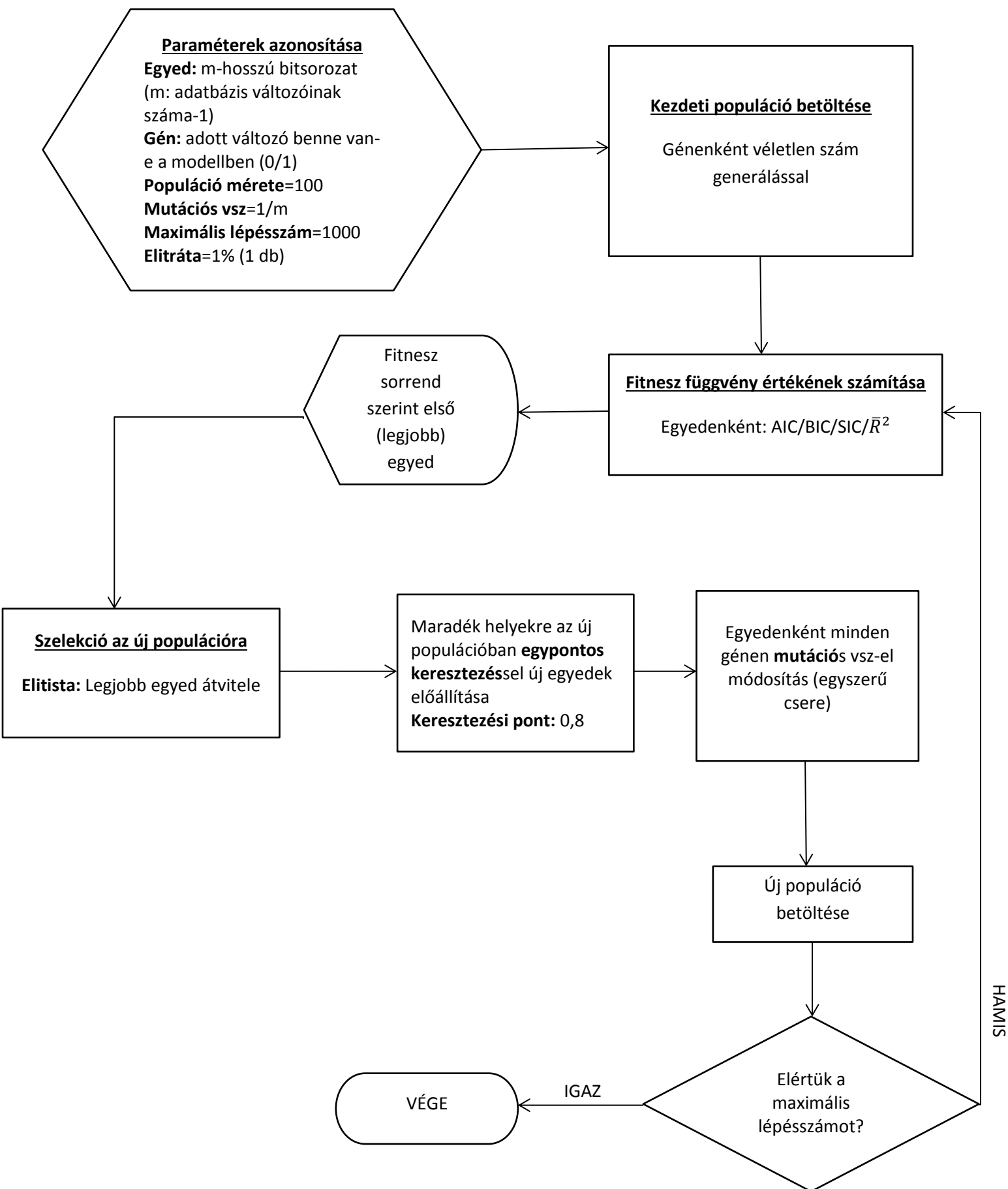
Azonban, az $\bar{R}^2$ mutatót a tanuló mintából (első 40%) számítom ki két okból is. Az egyik ok az, hogy a validációs, illetve teszt mintákon nem érvényes az $SST = SSE + SSR$ összefüggés, így a korrigálatlan R^2 mutatót csak a már megismert módon, a (7) képlet alapján lenne érdemes számítani.

Tehát azt nézzük meg, hogy az eredményváltozó szórásnégyzetének (varianciájának) mekkora része NEM származik a modell négyzetes hibájából (SSE). Ez adja a modell magyarázóerejét. Azonban itt az sem teljesen egyértelmű, hogy az SST számításakor a tanuló minta vagy a validációs minta átlagától vett négyzetes eltérést tekintjük SST-nek, azaz a szórásnégyzet számlálójának.

Továbbá, a másik ok az, hogy modelleket optimalizáló Backward vagy Forward lokális kereső algoritmus a tanuló mintából számított $\bar{R}^2$ -re is optimalizálhat annak köszönhetően, hogy elsősorban a nem szignifikáns változókat szűri ki a modellekből, lépésről lépésre, és törekszik arra, hogy az összes szignifikáns változót a modellben hagyja. Így a metaheurisztikák a Backward és Forward eljárásokkal való könnyebb összevetése érdekében az $\bar{R}^2$ -et a tanuló mintára számítjuk.

5.3. Az algoritmus rekonstrukciója

Az előző részben megállapított módon kezelve a modelleket, és az információs kritériumaikat (fitness függvényeiket) a cikk alapján az 1. folyamatábrán szereplő algoritmus rekonstruálható.

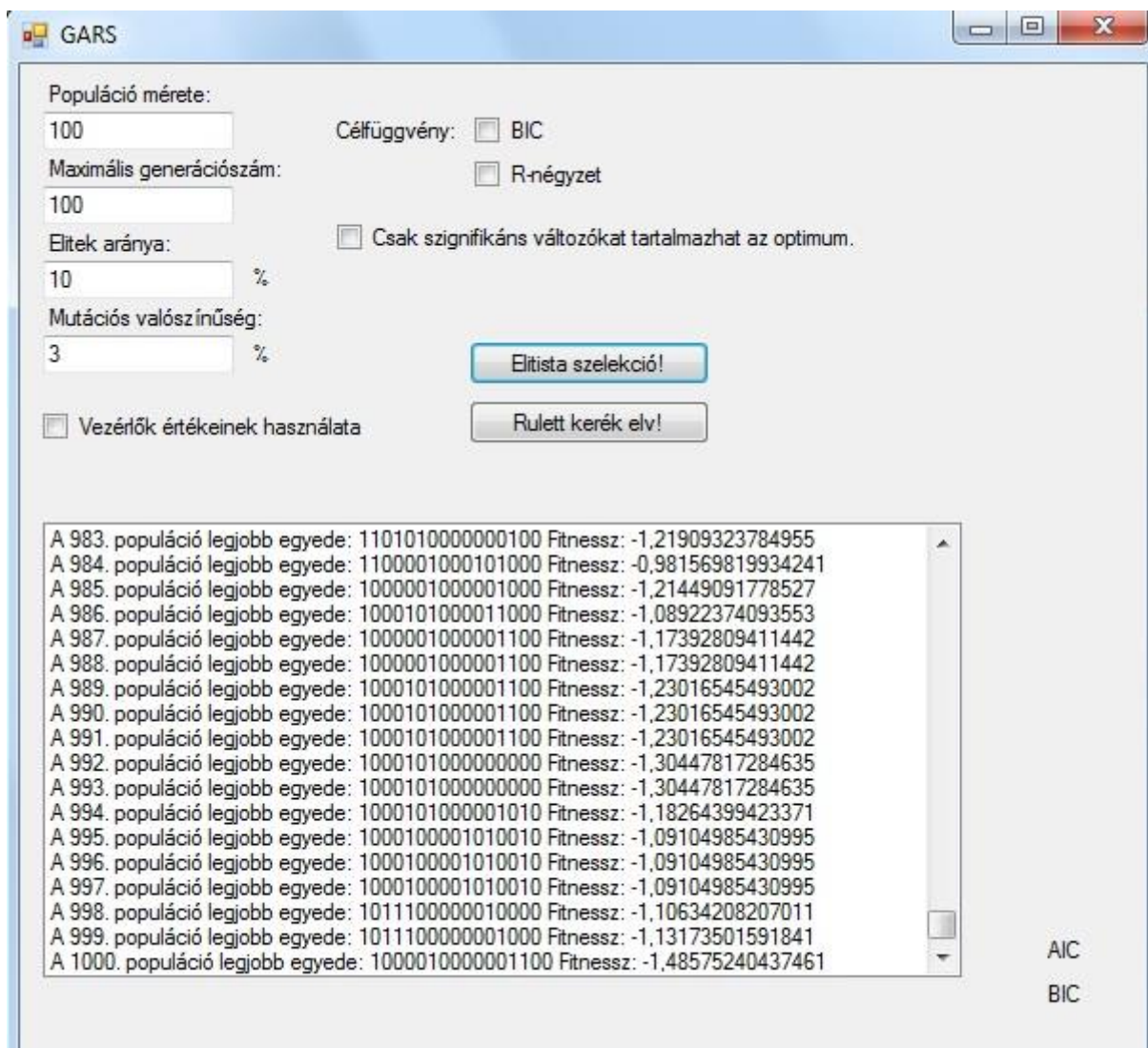


1. folyamatábra: A GARS algoritmus folyamatábrája, saját szerkesztés

Viszont az így megalkotott algoritmus egyáltalán nem konvergál egy optimumhoz. A 3. ábrán látható, hogy a 995. populációban a legjobb célfüggvény (AIC) értéke -1,09. Ellenben, pl. már

a 992. populáció legjobb egyede is egy, az optimumhoz jobban közelítő, -1,3 értékű célfüggvénnyel bír (minimalizálás), mégis az újabb populációkban (995.) rosszabb megoldást kapunk, tehát a konvergencia egy adott lépésszám után felbomlik.

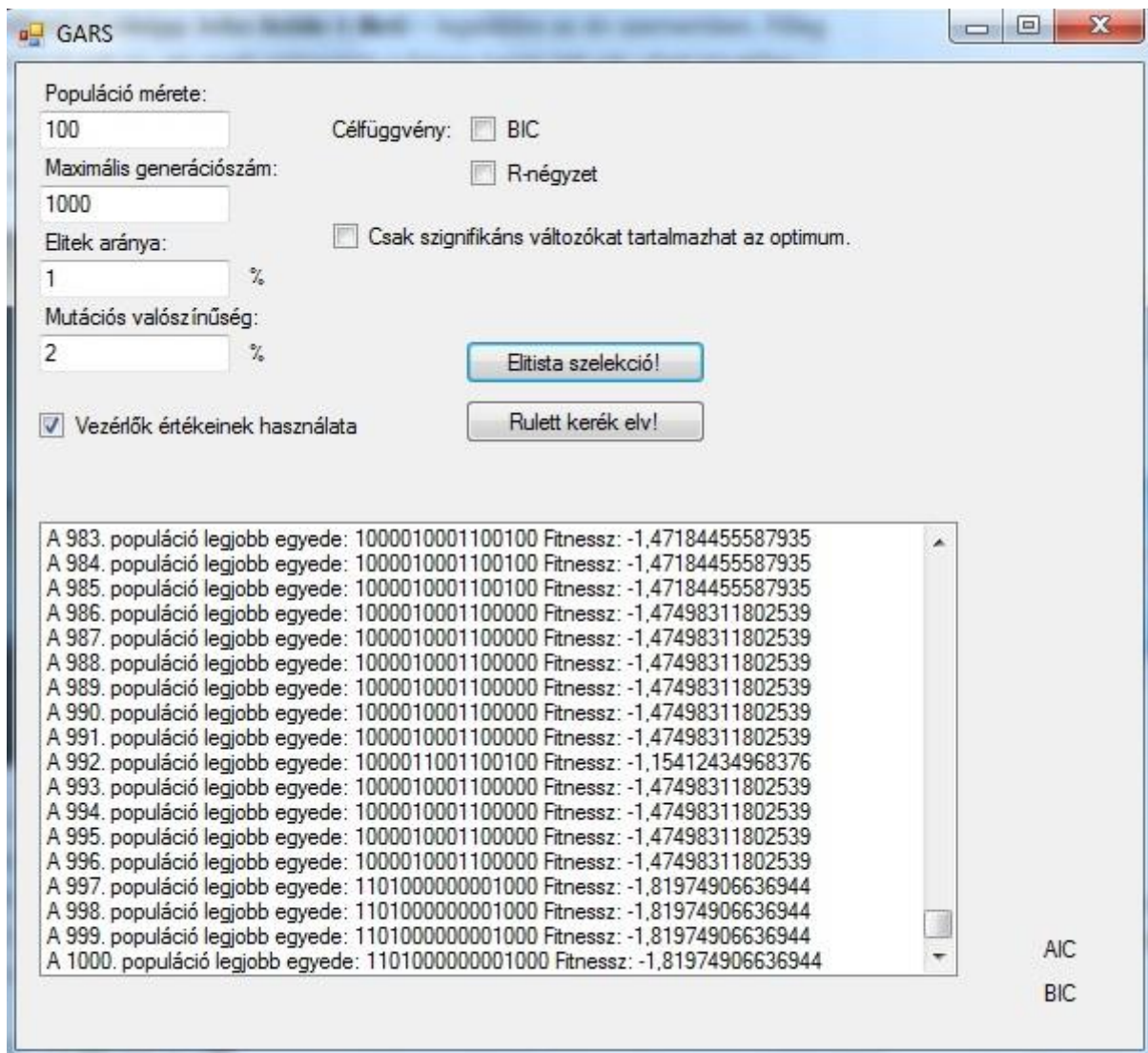
Az alapesetben az *AIC* mutató a célfüggvény a készített programban. A GARS algoritmus esetében minden eredmény 30 futtatás átlagaként vagy ugyan ennyi futtatásból levont következtetésként értelmezendő, ahogyan a cikkben is történik. Az ábrákon ilyen sorozatos futtatások legtipikusabb eredménye látható, szemléltetési céllal.



3. ábra: A GARS első futtatása

A legvalószínűbb probléma a mutációs valószínűség környékén kereshető. Ugyanis a cikkből nem derül ki egyértelműen, hogy a mutációs valószínűség azt jelenti-e, amit most feltételeztünk, hogy minden génen ekkor valószínűséggel módosítunk, vagy inkább azt jelenti, hogy egy egyed megváltoztatásának a valószínűsége ennyi. De ezen a problémán könnyen

segíthetünk, ha a mutációs valószínűséget lejjebb vesszük az $1/16=6,25\%$ -os értéknél, mivel kellően kis valószínűségnél pont az fog történni, hogy egy egyed csak egyszer módosul. Azonban ez a változtatás még önmagában nem hozza meg a kívánt eredményt, a konvergencia továbbra sem stabil (4. ábra). Tehát, még mindig előfordul, hogy a későbbi populációk rosszabb megoldásokat szolgáltatnak, mint a régebbiek, tehát a lépések során nem a globális optimum felé haladunk.



4. ábra: A mutációs valószínűség módosítása

Ennek további oka pedig, véleményem szerint, az elitrálásban található meg, hiszen a 100 elemű populációból csupán az egyetlen legjobb elem kerül tovább az új populációba. Így az egyes populációk egyedei nem kapnak kellő mennyiségű „íránymutatást” a szelekciós operátortól, hogy pontosan melyik, „milyen jellegzetességekkel rendelkező” megoldásokat kellene az evolúciónak előnyben részesíteni. Ezt csupán egyetlen egyed alapján megállapítani

túl nehéz. Így az elitrátát is megemeljük, 10%-ra, hogy megfelelő mennyiségű pozitív visszacsatolást kapjanak az egyes populációk. Ekkor pedig már az 5. ábrán látható, hogy egy teljesen stabil konvergenciát kapunk, egészen jó átlagos futásidővel (30 futtatásból 1,484 másodperc).

The screenshot shows the GARS software window with the following settings:

- Populáció mérete: 100
- Maximális generációs szám: 50
- Elitek aránya: 10 %
- Mutációs valószínűség: 3 %
- Célfüggvény: ☐ BIC, ☐ R-négyzet
- ☐ Csak szignifikáns változókat tartalmazhat az optimum.
- ☒ Vezérlők értékeinek használata
- Buttons: "Elitista szelekció!", "Rulett kerék elv!"

The results list shows the best individual for each population (A 27 to A 44) with their fitness values. The fitness values are consistent across all populations, indicating stable convergence.

Populáció	Legjobb egyede	Fitnessz
A 27.	1100000000000000	-1,9254982594034
A 28.	1100000000000000	-1,9254982594034
A 29.	1000000000000000	-2,01332773181054
A 30.	1000000000000000	-2,01332773181054
A 31.	1000000000000000	-2,01332773181054
A 32.	1000000000000000	-2,01332773181054
A 33.	1000000000000000	-2,01332773181054
A 34.	1000000000000000	-2,01332773181054
A 35.	1000000000000000	-2,01332773181054
A 36.	1000000000000000	-2,01332773181054
A 37.	1000000000000000	-2,01332773181054
A 38.	1000000000000000	-2,01332773181054
A 39.	1000000000000000	-2,01332773181054
A 40.	1000000000000000	-2,01332773181054
A 41.	1000000000000000	-2,01332773181054
A 42.	1000000000000000	-2,01332773181054
A 43.	1000000000000000	-2,01332773181054
A 44.	1000000000000000	-2,01332773181054

On the right side, there are labels for "AIC" and "BIC".

5. ábra: Az elitráta módosítása

Viszont az értéket tovább emelni nem szabad, a konvergencia már 20%-os elitráta esetén sem lesz stabil!

Ezek a beállítások a többi lehetséges célfüggvény eredményénél is stabil konvergenciát eredményeznek, hasonló átlagos futásidővel. Az eredmények a Backward és Forward algoritmus által optimálisnak tartott modell mutatószámainál jobb megoldást mutatnak (5. táblázat).

5. táblázat: A GARS futási eredményeinek összevetése a szelekciós eljárásokkal

	AIC (9) képlet	BIC (10) képlet	$\bar{R}^2$ (8) képlet
GARS (rekonstruált)	-2,013 (Változók: 1)	-1,987 (Változók: 1)	0,9829 (Változók: 1, 2, 3, 5, 6, 8, 11, 12, 15)
Backward	0,058 (Változók: 1, 3, 5, 6, 8, 12, 15)	0,239 (Változók: 1, 3, 5, 6, 8, 12, 15)	0,9822 (Változók: 1, 3, 5, 6, 8, 12, 15)
Forward	0,085 (Változók: 1, 3, 5, 6, 8, 11, 12, 15)	0,292 (Változók: 1, 3, 5, 6, 8, 11, 12, 15)	0,9827 (Változók: 1, 3, 5, 6, 8, 11, 12, 15)

Azonban akad az algoritmusnak még egy kritikus pontja. Ez pedig a rögzített keresztezési pont, az egyed 0,8-ik génjénél. A cikk ezt a választást nem magyarázza. Egy ilyen megoldás indokolt lehet például, ha a feladat például egy értelmes karaktersorozat előállítása, amiben vannak értelmes szavak is. Ekkor a karaktersorozatban a szóhatárok jó fix keresztezési pontok lesznek, mivel ekkor kisebb eséllyel rontunk el kialakuló szavakat egy új populációban, ezzel gyorsíthatjuk az optimumhoz történő konvergenciát.

Ellenben itt teljesen esetleges, hogy melyik változó bevonása/eltávolítása javítja a modell magyarázóerejét, csökkenti átlagos hibáját. Nincsenek olyan kialakuló minták (bitsoportok) az egyedekben, amiket elronthatunk. Így módosítjuk a keresztezési stratégiát úgy, hogy véletlenszerűen válasszon keresztezési pontot, ahol elvágja az egyedet. Ekkor azt figyelhetjük meg, hogy szignifikánsan ($\alpha = 5\%$) megnő a 20 lépésen belül kialakuló stabil konvergenciák aránya a 30 darab futtatásban (6. táblázat). Részletes számítások az 1. számú mellékletben.

6. táblázat: A Fix és Vegyes keresztezési pontok összehasonlítása

	Fix pont	Véletlen pont
Stabil konvergenciák aránya 20 lépésen belül	40%	63%

A szelekciós stratégia megváltoztatásával is lehet kísérletezni, ám az eltérő stratégiák (elitista, rulett kerék, stb.) alkalmazásai a munkám során közömbösnek bizonyultak a hatékonyság szempontjából. Ez ismételten abból adódhat, hogy az egyedekben nincs semmiféle szabályosság (mint a kialakuló szavak esetében). Ekkor, például a rulett kerék elvnek is szükséges a mutációs operátor, míg a karaktersorozatok előállítása során nem volt az. (Kovács, 2013) Hiszen, tisztán a keresztezésből – hiába kapja meg az átlagosnál jobb egyedeket – nem tudja megtalálni az optimumot, abban az esetben, ha csak egy darab teljesen véletlenszerűen kiválasztott változót kell bevenni, vagy kihagyni a modellből. Ez a jelenség pedig a jelenlegi feladatnál egy elég gyakori eset, a szabályosságok hiánya miatt.

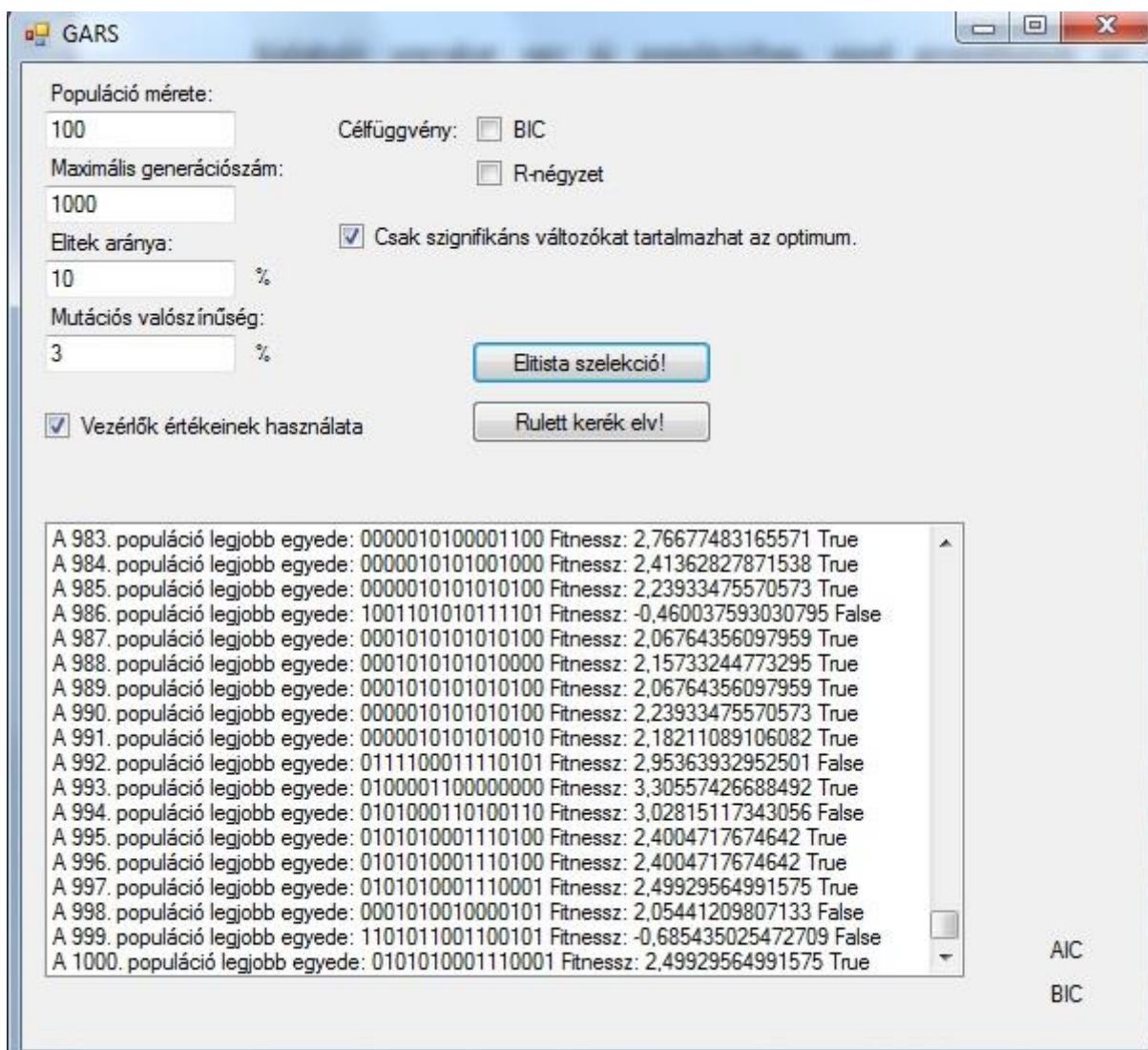
5.4. A GARS célfüggvényének bővítése

További vizsgálódások után azt lehet még felfedezni, hogy az algoritmusnak jelenlegi állapotában egy nagy hátránya van a Backward eljáráshoz hasonló lokális keresési eljárásokkal szemben: nem veszi figyelembe a célfüggvénye, hogy a modellben van-e nem szignifikáns változó, azaz olyan ismérv, aminek az együtthatója (β_i) közel 0, valamint nem is lehet egyértelműen eldönteni, hogy változása (ceteris paribus) növeli vagy csökkenti-e az eredményváltozó értékét. A Backward (vagy a Forward) eljárásnak pedig a kilépési feltétele pont az, hogy már nem maradt nem szignifikáns változó a modellben. Ezzel szemben a GARS olyan megoldásokat is talál, ami lehet, hogy egy mutatószám szerint optimális, de van benne nem szignifikáns változó is ($\alpha = 5\%$ szignifikancia szinten). Például az 5. táblázatban az $\bar{R}^2$ -re optimális modell.

Továbbá, azért is érdemes elvégezni ezt a bővítést, mert a Backward és Forward eljárások során az egyes lépések modelljei csupán egy változóban térnek el. Ekkor előfordulhat az (jelentős multikollinearitás esetén), hogy X_1 változó, ha önmagában áll a modellben, jobban csökkenti AIC-ot, vagy növeli $\bar{R}^2$ -et, mint ha X_2 változó állna önmagában a modellben. Ám, ha mind a kettő szerepel a modellben, akkor pl. a Backward algoritmus a hipotézisvizsgálat eredményeképpen mégis inkább X_2 -t preferálja, és X_1 -et kihagyja a modellből, tehát nem szignifikáns változó lesz. Ebben az esetben pedig már nem tarthatna fel a szelekciós eljárás olyan, az optimumhoz közelebb álló megoldásokat, amik X_1 -et szerepeltetik, és a többi változó is szignifikáns benne. (Kovács, 2008) Tehát, jelentős multikollinearitás esetén a Backward vagy a Forward szelekció sem tudja biztosítani, hogy tisztán a „legszigorúbb” változók szerepeljenek a modellben. Ezért is érdemes a metaheurisztikákat egy második

célfüggvénnyel bővíteni, ami azt vizsgálja, hogy csak szignifikáns változók kerültek-e a modellbe.

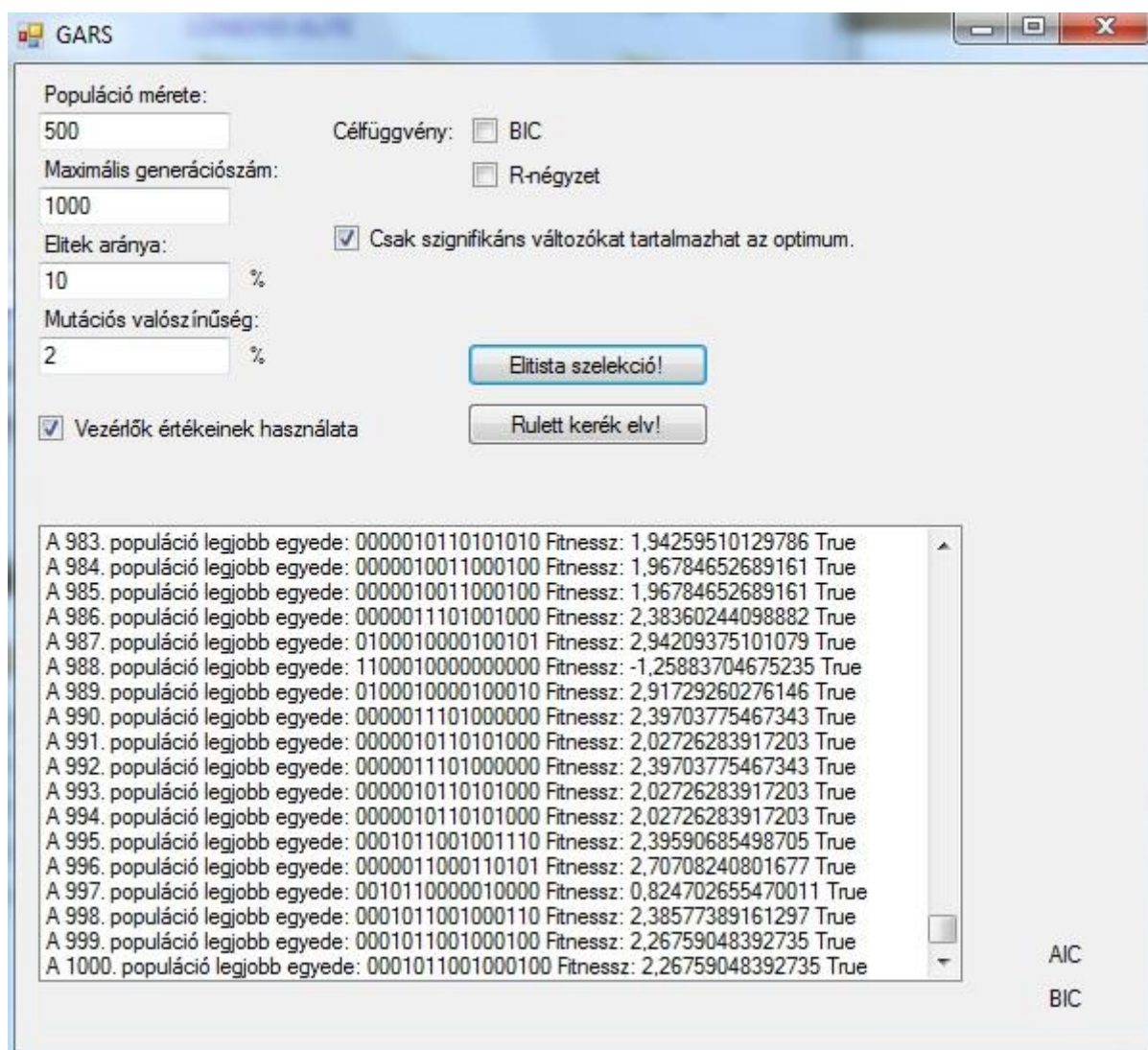
Ha ezt a hiányosságot megkíséreljük kijavítani a GARS-ban (a legjobb beállítások mellett), akkor viszont ismételen nem kapunk stabil konvergenciát (6. ábra). Az algoritmus egyes lépéseiben a legjobb egyed célfüggvény értékei egy igaz/hamis (true/false) jelzővel is bővülnek a programban. Az igaz érték (true) azt jelenti, hogy a megoldás (legjobb egyed) csak szignifikáns változókat tartalmaz, míg ha az érték hamis (false), akkor a megoldás modellje tartalmaz legalább egy nem szignifikáns változót is ($\alpha = 5\%$ szignifikancia szinten). Ez a bővítés már túlmutat a cikk algoritmusán, az nem foglalkozott egy ilyen dupla célfüggvénnyel.



6. ábra: A GARS futása dupla célfüggvény alkalmazása esetén

Ennek az oka az, hogy egy populáción belül a csak szignifikáns változókat tartalmazó egyedek aránya nagyon kicsi, nagyobb populációk (pl.: 2000 egyed) esetében is 1-2%, maximum. Így az új populációk nagyon kevés „jó példát” kapnak, ez kellően megnehezíti a konvergenciát, ahogyan az 1%-os elitrátánál már tapasztaltuk korábban. Ráadásul, akár egyetlen egy génen végrehajtott mutáció is teljesen elronthatja a következő generációba átengedett optimumhoz közelítő egyedeket, ezek után pedig kezdődhet a hosszú keresési folyamat előlről, mintha újra visszatérnénk a véletlenszerűen generált populációhoz.

A mutációs valószínűség nullára állítása, tehát az operátor elhagyása sem javít a konvergencián, mivel ekkor a véletlenszerűen felfedezett megoldásokat teljesen kivesszük a keresésünkből, viszont a probléma, a változók ki- és bevonásának esetlegessége nagyon is igényli az ilyen megoldásokat is, ahogyan a fix keresztezési pont esetében láttuk is. Nagyobb méretű populációk és nagyobb maximális lépésszám javít valamit a helyzeten, egyre hosszabb időtartamokra alakulnak ki jó irányú konvergenciák, de előbb-utóbb felbomlanak. Továbbá a futásidő a 3 percet meghaladja, már egy 500 elemű populáció esetén is, ami túl magas processzoridő, az eddigi átlagos futásidővel (kb. 1,5 másodperc) összehasonlítva. Lásd 7. ábra.



7. ábra: A GARS algoritmus futása dupla célfüggvény esetén, nagyobb méretű populációval

Tehát, ha ilyen „dupla” célfüggvénnyel dolgozunk, akkor a genetikus algoritmus szimultán vektorkezelése kifejezetten káros. Olyan metaheurisztika lenne ekkor a célravezető, mint például a harmónia kereső algoritmus (HS), amelynek csak abban az esetben frissül a populációja (több elemű memóriája), ha biztosan jobb az új megoldás az eddigi legrosszabbnál.

Továbbá a HS algoritmus operátorai sokkal nagyobb hangsúlyt fektetnek a véletlen alapú módosításokra, hiszen tulajdonképpen mutációs operátorok, csupán az „agresszivitásukban” térnek el (módosít vagy teljesen új egyedet generál). Ezen kívül általában a genetikus algoritmusénál sokkal kisebb méretű memóriával (populációval, repertoárral) a leghatékonyabb. Így joggal remélhetjük, hogy ha HS algoritmust alkalmazunk a feladat

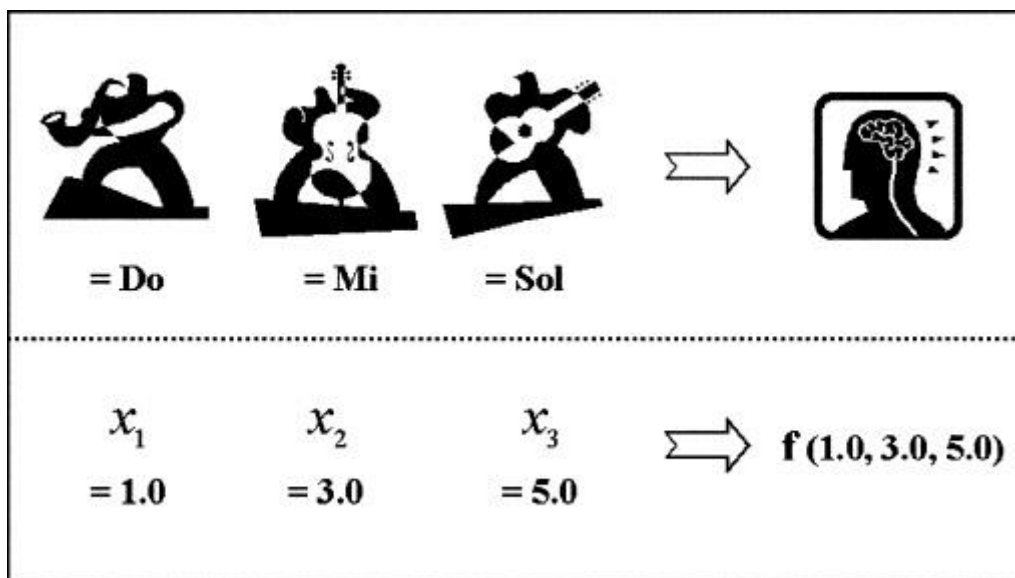
megoldására, akkor hatékonyabb megoldást kapunk, mind a dupla célfüggvény, mind a futásidő terén.

6. Harmony Search for Regressors' Selection (HSRS)

6.1. Az általános 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.

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 ötletszerűen lejátszik 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 (8. ábra).

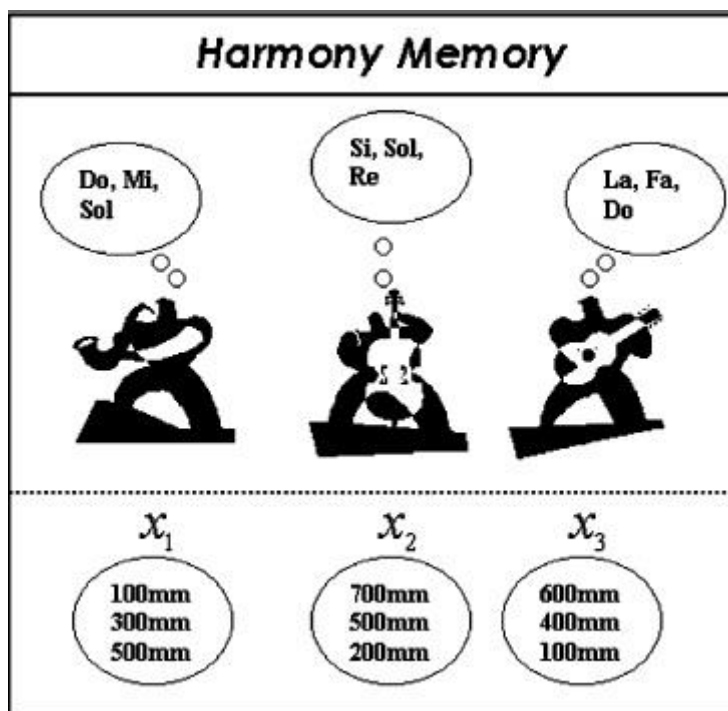


8. ábra: Az improvizáció zenei analógiája

Forrás: Lee – 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). Lásd 9. ábra.



9. ábra: A harmónia keresés memóriája

Forrás: Lee – 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 $1-HMCR$. 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 $1-PAR$ valószínűséggel hagyjuk változatlanul.

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

$$x_{új} = x_{régi} + bw * r \quad (12)$$

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 genetikusan 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”. Ráadásul, pont ez az egyszerűsített szelekció teszi a harmónia keresést ígéretessé a dupla célfüggvény kezelésében.

A keresztezés operátora teljesen kiesett az algoritmusból, ám helyette kétféle mutációs operátort kapunk: a hangmagasság változtatása bizonyos géneket módosít, ez felel meg a klasszikus (genetikusan is alkalmazott) mutációnak, míg a teljesen új dallam generálása egy eldurvult, nagyon agresszív mutációt jelent. További érdekesség a

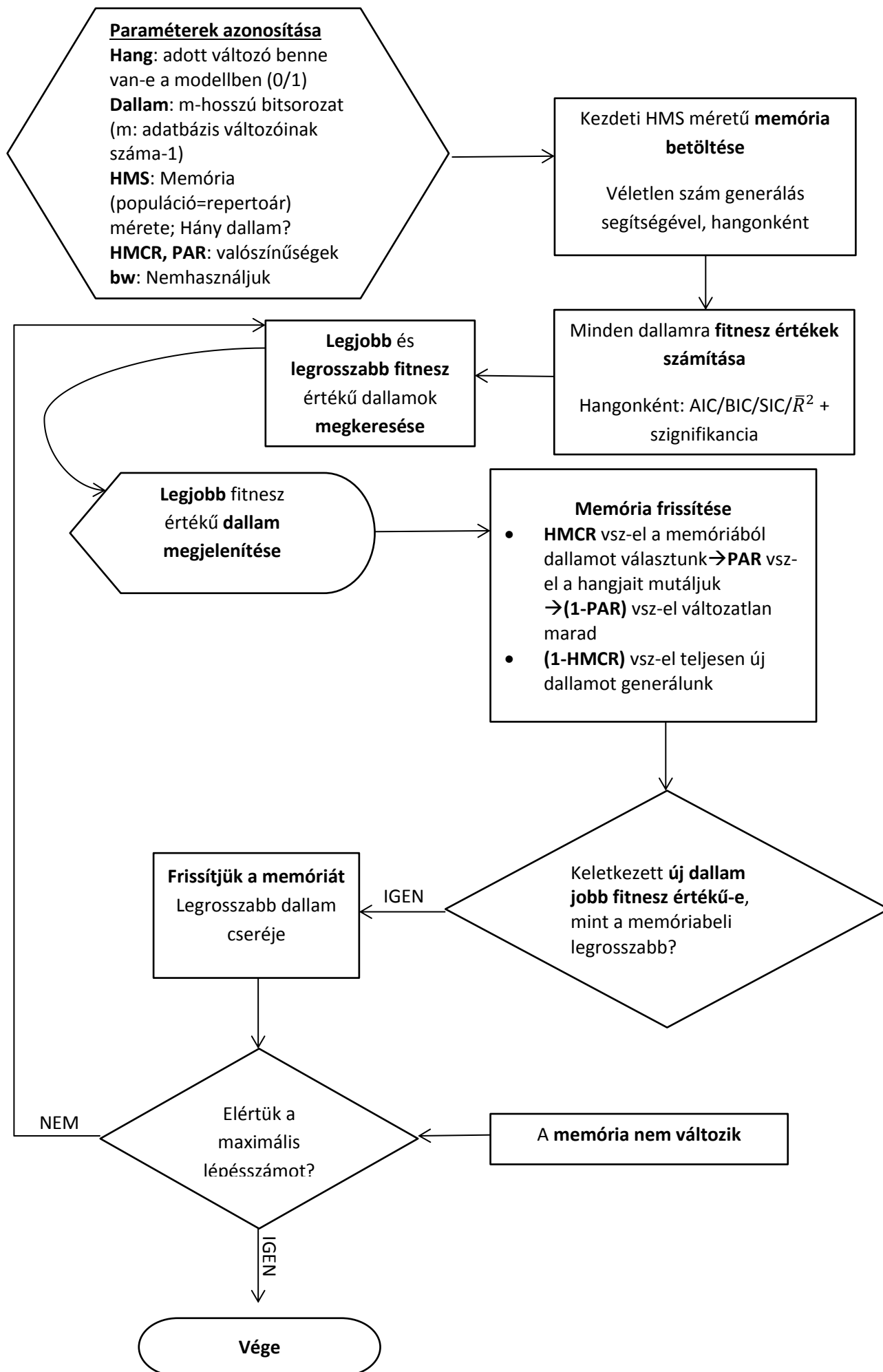
harmóniakereső algoritmusban, hogy ezek az operátorok nem sorban egyszerre, hanem bizonyos valószínűség szerint következnek be, azaz váltakoznak az egyes ciklusokban.

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.

6.2. A HSRS megalkotása

Amikor a harmónia kereső algoritmust a (regressziós) modellszekciós feladatra alkalmazzuk, akkor az egyedeket, azaz az algoritmus dallamait szintén a bináris rendszer szerint tudjuk reprezentálni. Ha az adathalmaz adott változója a modellben van, akkor a megfelelő hang értéke 1, egyébként (ha nincs benne) 0. Egy ilyen reprezentáció az algoritmus azon pontján módosít, amikor a memória egy adott dallamát módosítjuk. Mivel a hangok a génekhez hasonlóan csupán kétféle értéket vehetnek fel, a sáv szélesség (*bw*) paraméterre egyáltalán semmi szükség. Hiszen ekkor tulajdonképpen a legegyszerűbb mutáció történik: 0-ról 1-re vagy 1-ről 0-ra módosul a kiválasztott dallam egyik hangja. Ugyan is a probléma megfogalmazásából adódóan a gének tört értékeket nem vehetnek fel, tehát az általános algoritmus szerinti (12) képlet alkalmazása fölösleges bonyolítása a módosítási folyamatnak, és nem vezetne más helyes dallamok felfedezésére. Így ez az operátort tulajdonképpen a GARS mutációs operátoraként fogjuk fel.

Ezeket a módosításokat alkalmazva a **HSRS (Harmony Search for Regression Selection)** algoritmus folyamatábrája a 2. folyamatábra szerint alakul.



2. folyamatábra: A HSRS algoritmus folyamatábrája, saját szerkesztés

A megfelelő paraméterezéssel (2. számú melléklet) a HSRS algoritmus képes hatékonyan megtalálni azon modelleket, melyek valamely információs kritériumra (AIC, BIS, SIC, $\bar{R}^2$) optimálisak, és csupán szignifikáns változókat tartalmaznak, tehát minden változóról el lehet dönteni egyértelműen a modellben, hogy növeli vagy csökkenti az eredményváltozó értékét. Azaz, sikerrel kezeli a dupla célfüggvényt is. Harminc futtatásból csupán kétszer konvergált nem megfelelő, lokális optimumhoz (10. ábra).

The screenshot displays the HSRS algorithm interface. On the left, there are input fields for parameters: HMS (set to 5), HMCR (set to 70%), PAR (set to 70%), and Maximális lépésszám (set to 8000). Below these, there are checkboxes for the objective function: BIC (checked) and R-négyszet (unchecked). The main area on the right shows a list of 30 search results. Each result line contains the number of steps (8000), the current optimum, a binary string, the fitness value, and a boolean status. Most results show a fitness of -2,01332773181054 and a status of True, indicating convergence to the global optimum. Two results show a different fitness value (-0,996600316816196) and a status of True, indicating convergence to a local optimum. At the bottom, there is a checkbox labeled 'Csak szignifikáns változókat tartalmazhat az optimum.' which is checked, and a button labeled 'Harmony Search!'.

10. ábra: A HSRS algoritmus futása

Az alapesetben – a GARS-hoz hasonlóan – az AIC mutató a célfüggvény a programban. A többi célfüggvény (természetesen itt is figyeljük a modellben lévő változók szignifikanciáját) esetén is stabil konvergenciát mutat a HSRS.

7. táblázat: A HSRS futási eredményeinek összevetése a szelekciós eljárásokkal

	AIC (9) képlet	BIC (10) képlet	$\bar{R}^2$ (8) képlet
HSRS	-2,013 (Változók: 1)	-1,987 (Változók: 1)	0,9822 (Változók: 1, 3, 5, 6, 8, 12, 15)
Backward	0,058 (Változók: 1, 3, 5, 6, 8, 12, 15)	0,239 (Változók: 1, 3, 5, 6, 8, 12, 15)	0,9822 (Változók: 1, 3, 5, 6, 8, 12, 15)
Forward	0,085 (Változók: 1, 3, 5, 6, 8, 11, 12, 15)	0,292 (Változók: 1, 3, 5, 6, 8, 11, 12, 15)	0,9827 (Változók: 1, 3, 5, 6, 8, 11, 12, 15)

$\bar{R}^2$ mutató esetén az optimum megegyezik a Backward eljárás optimumával (7. táblázat), hiszen a Backward algoritmus a nem szignifikáns változók szűrése mellett a tanuló mintára számított $\bar{R}^2$ mutatóra is optimalizálhat, noha nem ez az elsődleges célfüggvénye.

Ugyan is az $\bar{R}^2$ mutató az eredményváltozó varianciáját (négyzetes szóródását) kísérli meg minél inkább magyarázni, ehhez pedig több magyarázó változó bevonását engedi meg a büntetőtag. Így a mutatónak kifejezetten kedvez az, hogy a Backward (és a Forward) algoritmus lehetőleg az összes szignifikáns változót a modellben akarja tartani. Ezzel ellentétben, az információs kritériumok esetében, amiket már egy külön, új validációs mintára számolunk, a szelekciós eljárások nem képesek optimális megoldást találni. Mivel az információs kritériumok az egy elemre/rekorda jutó modellhibát $\left(\frac{SSE}{n}\right)$ minimalizálják. Ekkor az újabb változók már nem javítanak annyit, hogy a büntetőtag a regressziós modellben meghagyja őket. Tehát, ebben az esetben már „árt”, hogy a Backward (Forward) szelekció sok változót hagy meg a modellben. Az eredményeket alátámasztja a változók korrelációs mátrixa (4. számú melléklet), illetve az Y és az X_1 változó közötti parciális korrelációs együtttható értéke (-0,989) is.

A GARS algoritmus pedig a szimultán vektorkezelése miatt, a dupla célfüggvényeket volt képtelen kezelni. Ugyan – akkor a modellszelekciós mutatószámokra jobban optimalizált a Backward és Forward algoritmusoknál.

A HSRS az egyetlen a fenti algoritmusok közül, amely mindkét feladatot, a szignifikáns változók szűrését és (validációs mintára számított) információs kritériumok szerinti optimalizálást is hatékonyan megoldja. Ezt alapvetően annak köszönheti, hogy az új hangot előállító operátorai nagy hangsúlyt helyeznek a véletlenszerűségekre. Hiszen az egyes modellekben nem gyakoriak a kialakuló minták, teljesen esetleges, hogy a modell előrejelző képességén az adatbázis melyik változójának bevonása/elhagyása javít. Ugyanis a változók sorrendje egy adatbázisban, teljes mértékben a tervezőn múlik, nem feltétlenül csoportosulnak egy helyre a „jó” változók. Így a véletlen mutációs eljárások (módosítás, teljesen új hang generálása) hatékonyabban tudják feltérképezni a keresési teret, mint a főleg már kialakulóban lévő mintákat vegyítő keresztezési operátorok, amik a GARS domináns műveletei (hiszen a mutáció csupán egy kis valószínűségű módosító szerepkört tölt be az algoritmusban).

Ezen túl pedig a HSRS azon tulajdonsága is az előnyére válik, hogy csak abban az esetben frissül a repertoárja (memóriája, populációja), ha biztosan jobb az új megoldás az eddigi legrosszabbnál. Azaz, nem cserél le egy szignifikáns megoldást nem szignifikánsra (ami a GARS esetében megtörténhet egy rosszul bekövetkező mutáció miatt). Hiszen a csak szignifikáns változókat tartalmazó dallamok (egyedek) aránya kezdeti memóriákban nagyon kicsi (legfeljebb 1-2%). Ebből kifolyólag pedig azokra, amiket már megtaláltunk vigyázni kell, és a keletkező új hangot csak akkor szabad beengedni, ha a biztosan nem ront a memória szignifikáns dallamainak arányán. ennek a feltételnek pedig a HSRS memóriafrissítési kritériuma tökéletesen meg is felel.

Egyetlen egy probléma van az algoritmussal, ez pedig a futásidő. Harminc futtatás alapján az átlagos futásidő 2,571 másodperc. Ez az idő kb. 1 másodperccel magasabb processzoridőt eredményez, mint a GARS átlagos futási ideje. Noha a GARS ezt az eredményt akkor hozza, amikor csak az információs kritériumokra optimalizál, szignifikanciára nem. Viszont felveti a kérdést, hogy lehetséges-e az elhasznált processzoridőt minimalizálni. Ha szimplán a maximális lépésszám csökkentésével próbálkozunk (ceteris paribus, hiszen a többi paramétert már optimalizáltuk), akkor sikerül leSORÍTANI ugyan az átlagos futásidőt 1,278 másodpercre, de 30 futtatásból a nem megfelelő, lokális optimumhoz való konvergálások aránya megnő (11. ábra).

HMS:

HMCr:
 %
PAR:
 %
Maximális lépésszám:

☐ BIC
☐ R-négyszet

☒ Csak szignifikáns változókat tartalmazhat az optimum.

Hamony Search!

4000 lépésből az optimum: 0001010101110000 Fitness: 2,07320057473101 True
4000 lépésből az optimum: 1100110101100000 Fitness: 0,024675904100779 True
4000 lépésből az optimum: 1000000000000000 Fitness: -2,01332773181054 True
4000 lépésből az optimum: 1000000000000000 Fitness: -2,01332773181054 True
4000 lépésből az optimum: 0101010101101010 Fitness: 2,01927905125662 True
4000 lépésből az optimum: 1000000000000000 Fitness: -2,01332773181054 True
4000 lépésből az optimum: 1000000000000000 Fitness: -2,01332773181054 True
4000 lépésből az optimum: 1000000000000000 Fitness: -2,01332773181054 True
4000 lépésből az optimum: 1000000000000000 Fitness: -2,01332773181054 True
4000 lépésből az optimum: 0001010010110100 Fitness: 1,88776910230664 True
4000 lépésből az optimum: 1100010000000000 Fitness: -1,25883704675235 True
4000 lépésből az optimum: 1000000000000000 Fitness: -2,01332773181054 True
4000 lépésből az optimum: 1000010101010000 Fitness: 0,0462330642775795 True
4000 lépésből az optimum: 0000010011001100 Fitness: 1,94775542715673 True
4000 lépésből az optimum: 1000000000000000 Fitness: -2,01332773181054 True
4000 lépésből az optimum: 0001010110110010 Fitness: 1,77025515370515 True
4000 lépésből az optimum: 1000000000000000 Fitness: -2,01332773181054 True
4000 lépésből az optimum: 1011010100010000 Fitness: -0,243213081626092 True
4000 lépésből az optimum: 1000000000000000 Fitness: -2,01332773181054 True
4000 lépésből az optimum: 1000000000000000 Fitness: -2,01332773181054 True
4000 lépésből az optimum: 0010110010000100 Fitness: 0,632044696965806 True
4000 lépésből az optimum: 0010110010000100 Fitness: 0,632044696965806 True
4000 lépésből az optimum: 1000000000000000 Fitness: -2,01332773181054 True
4000 lépésből az optimum: 1000000000000000 Fitness: -2,01332773181054 True
4000 lépésből az optimum: 1000000000000000 Fitness: -2,01332773181054 True

11. ábra: A HSRS futása, ha a maximális lépésszám 4000

Tehát, egy hatékonyabb megoldást kell találni az algoritmus processzoridejének minimalizálására. Ezt pedig a fejlesztetett harmónia kereső algoritmus (Improved Harmony Search, IHS) alkalmazásával el is érhetjük.

6.3. Az Improved HSRS

A továbbfejlesztés az eredeti algoritmusban abban rejlik, hogy a futás során a PAR (azaz egy hang módosításra vonatkozó) valószínűséget folyamatosan növeljük, míg a módosítások hatókörét, intenzitását, tehát a sáv szélességet (*bw*) folyamatosan csökkentjük. A fejlesztés célja tehát az, hogy az algoritmus futásának elején minél agresszívebb legyen a kiválasztott dallam módosítása: inkább az új egyed generálását, és a nagy tartományban történő módosítást támogatja a korai szakaszban ez a fejlesztett algoritmus. (Madhavi és munkatársai, 2007)

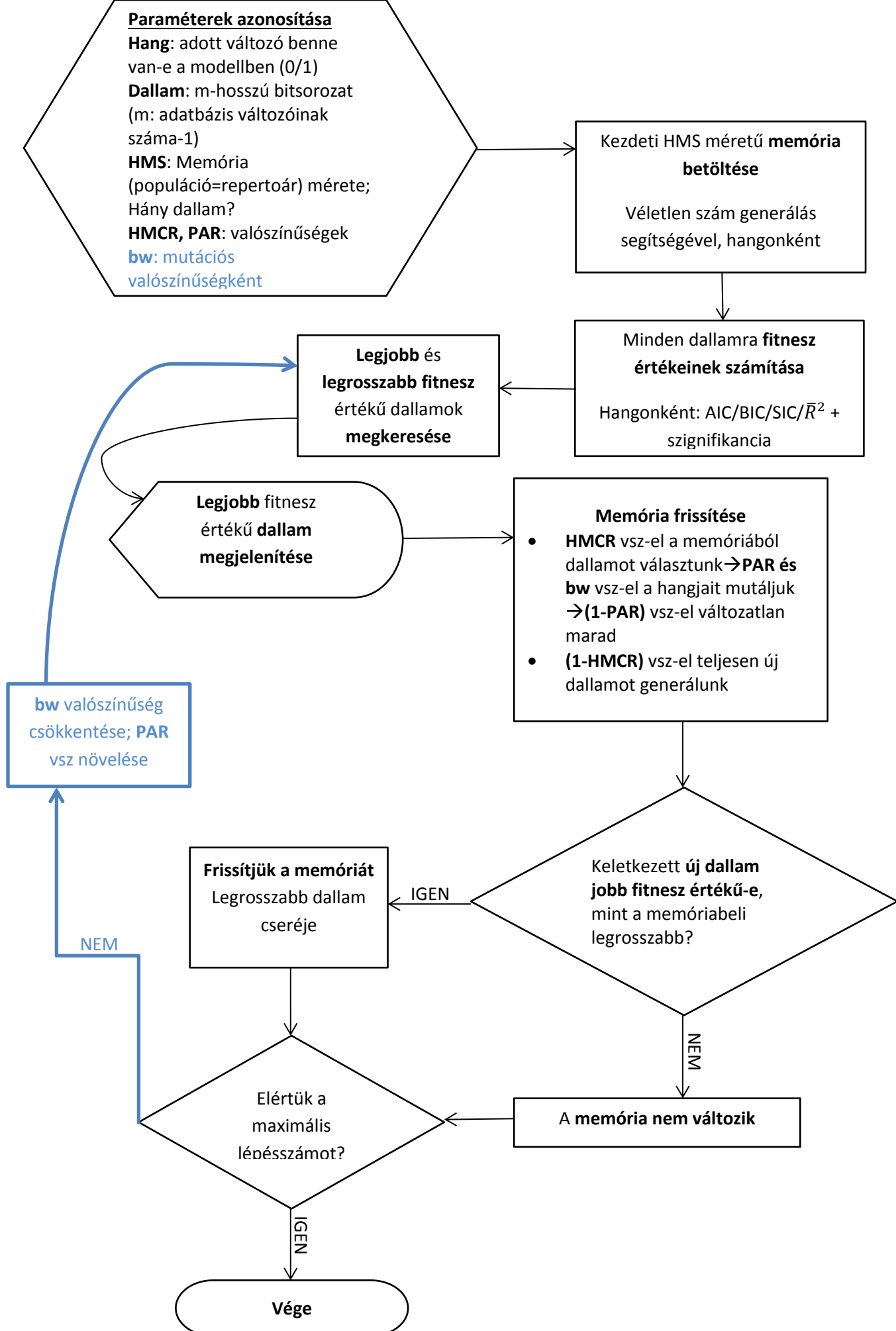
Ez a logika a mi problémánkhoz is kellően illeszkedik, ugyanis a kezdeti repertoárokban (populációkban) a csak szignifikáns változókat tartalmazó dallamok (modellek) aránya rendkívül kicsi (1-2%). Így a futtatás elején nagy szükség van az ilyen tulajdonságú dallamok minél „agresszívebb” keresésére. Mivel a memóriafrissítési kritérium nem engedi, hogy szignifikáns változókat tartalmazó megoldások nem szignifikánsakra cserélődjének, így ezt nyugodtan meg is tehetjük. Az algoritmus futásának utolsó lépéseiben pedig csupán

finomhangolni kell az egyes megoldásokat, így főleg a már létező megoldások kis intenzitású módosítására kell koncentrálni.

Az egyetlen problémánk, hogy a dallamok bináris reprezentációja miatt a bw érték nem csökkenthető, hiszen az alap HSRS-ben nem is értelmeztük ezt a paramétert.

Azonban, ha a GARS alapján úgy értelmezzük, mint egy mutációs valószínűséget, azaz egy hang megváltoztatásának valószínűségéként, akkor már minden további nélkül csökkenthető folyamatosan az algoritmus futása során. Indulhatunk egy kellően magas értékről (pl. 50-60%), amit a futás után lecsökkentünk 1-2%-ra. Azaz az algoritmus futásának elején, még ha egy létező dallamot is módosítok, a változtatások több hangot is érinteni fognak, tehát kellően változatos megoldások születnek. Míg végső stádiumában már legfeljebb 1 hang fog változni a módosítandó dallamon, pont amennyi a finomhangoláshoz szükséges.

A fenti elvek szerint keletkezett algoritmus szerkezete a 3. folyamatábra szerint alakul (a HSRS-hez képest módosítást kék színnel jelezzük).



3. folyamatábra: Az Improved HSRS algoritmus folyamatábrája, saját szerkesztés

A tanulság tehát az, hogy az Improved HSRS futtatásakor érdemes több induló mutációs valószínűséget is kipróbálni, mivel az optimális modell változóinak száma, és egymáshoz képesti viszonya befolyásolni tudja a szükséges módosítások intenzitásának mértékét.

7. Összefoglalás

A dolgozat során áttekintettük a lineáris regressziós modellek szelekciójának problémáját, és annak hagyományos algoritmikus megoldásait (Backward és Forward szelekció). Megállapítottuk, hogy ezen eljárások alapvetően lokális keresések, hiszen részmegoldásaik regressziós modelljei csupán egy változóban térnek el, így lokális optimum felé konvergálhatnak. Különösen, ha a lehetséges magyarázó változók között jelentős multikollinearitás áll fenn. Ráadásul, a szelekciós eljárások célfüggvénye elsősorban csak a szignifikáns változók megkeresésére összpontosít, és a modell jóságát mérő valamely mutatószám ($\bar{R}^2$, AIC , BIC , stb.) szerinti optimalizálást csak közvetve veszi figyelembe.

Ezen okok miatt megkíséreltünk globális kereső eljárásokat, metaheurisztikákat illeszteni a modellszelekciós probléma megoldására. Elsőként egy már létező megoldást, a GARS algoritmust (Minerva – Paterlini, 2010) rekonstruáltuk, és megállapítottuk, hogy a Backward és Forward szelekciónál hatékonyabban optimalizál a regresszió illeszkedését mérő mutatószámokra ($\bar{R}^2$, AIC , BIC , stb.). Azonban, célfüggvénye nem veszi figyelembe, hogy az optimális regressziós modellben csak szignifikáns magyarázó változók szerepeljenek.

Amikor megkíséreljük bővíteni az algoritmus célfüggvényét egy ilyen „dupla” célfüggvénnyé (ami a mutatószámokra és a magyarázó változók szignifikanciájára is optimalizál), akkor nem képes már stabil konvergenciát mutatni az optimális megoldás felé. Ennek fő oka a GARS szimultán vektorkezelése. Az egy populáción belül a csak szignifikáns változókat tartalmazó egyedek aránya nagyon kicsi, és az új populáció kialakításakor a szelekció következtében ezek a „jó tulajdonságú” egyedek is módosulhatnak, így nem biztosított az algoritmus konvergenciája.

Ezek alapján megállapítottuk, hogy olyan metaheurisztika lenne ekkor a célravezető, mint például a harmónia kereső algoritmus (HS), amelynek csak abban az esetben frissül a populációja (több elemű memóriája), ha biztosan jobb az új megoldás az eddigi legrosszabbnál. Ennek ismeretében megalkottuk a HSRS algoritmust, ami sikerrel optimalizált a dupla célfüggvényre is, egyedülként a dolgozatban vizsgált algoritmusok közül.

A HSRS futásidejét is sikerült javítani az Improved HSRS algoritmus implementálásával, ahol az általános algoritmus sáv szélesség paraméterét (bw) egy mutációs valószínűségként értelmeztük.

Fontos megállapítás, hogy ezen valószínűség megfelelő megválasztása nem független az optimum jellegétől, így még érdemes lehet olyan metaheurisztikákat keresni, amik képesek a futásidőt általánosan javítani. Különösen érdekes lehet megnézni, hogy egyelemű populációval rendelkező metaheurisztikák milyen hatékonysággal oldják meg a problémát. Az aktuális megoldások cseréjére alkalmazott szabályai miatt különösen ígéretesnek tartjuk a dupla célfüggvény kezelésében a tabukereső, illetve a szimulált hűtés metaheurisztikákat. Ezen kívül a GARS algoritmus fejlesztése is célravezető lehet egy olyan irányba, hogy a mutációs valószínűségét folyamatosan csökkentjük egy nagy értékről az algoritmus futása során, hogy az Improved HSRS-hez hasonlóan hatékonyan tudja kezelni a dupla célfüggvényt.

A dolgozatot a tárgyalt algoritmusok futási eredményeinek összefoglalásával zárjuk (8. táblázat).

8. táblázat: A dolgozat algoritmusainak futási eredményei

	AIC (9) képlet	BIC (10) képlet	$\bar{R}^2$ (8) képlet	Átlagos futásidő (másodperc)
Improved HSRS	-2,013 (Változók: 1)	-1,987 (Változók: 1)	0,9822 (Változók: 1, 3, 5, 6, 8, 12, 15)	1,278
HSRS	-2,013 (Változók: 1)	-1,987 (Változók: 1)	0,9822 (Változók: 1, 3, 5, 6, 8, 12, 15)	2,571
GARS (rekonstruált)	-2,013 (Változók: 1)	-1,987 (Változók: 1)	0,9829 (Változók: 1, 2, 3, 5, 6, 8, 11, 12, 15)	1,484
Backward	0,058 (Változók: 1, 3, 5, 6, 8, 12, 15)	0,239 (Változók: 1, 3, 5, 6, 8, 12, 15)	0,9822 (Változók: 1, 3, 5, 6, 8, 12, 15)	1,011
Forward	0,085 (Változók: 1, 3, 5, 6, 8, 11, 12, 15)	0,292 (Változók: 1, 3, 5, 6, 8, 11, 12, 15)	0,9827 (Változók: 1, 3, 5, 6, 8, 11, 12, 15)	0,940

Mellékletek

1. számú melléklet: A Fix és Véletlen keresztezési pontok esetén a 20 lépésen belüli stabil konvergenciák arányának elemzése

Minél nagyobb az arány, annál hatékonyabb az algoritmus.

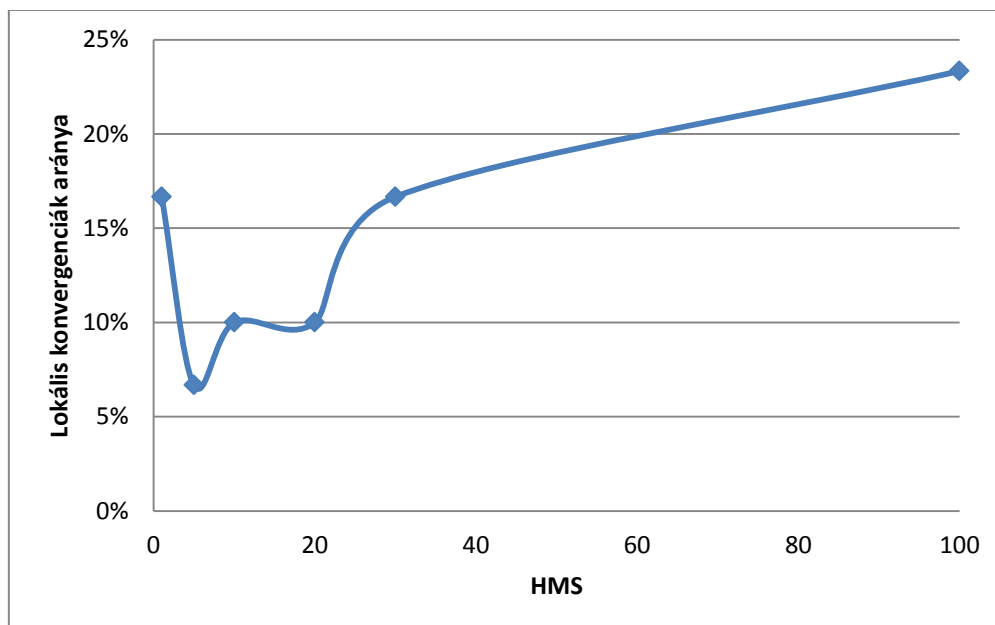
	Fix	Véletlen		H_0	$P(\text{Fix})=P(\text{Véletlen})$					
n	30	30		H_1	$P(\text{Fix}) < P(\text{Véletlen})$					
k	12	19								
p	40%	63%								
Próba	bal									
Elvi eltérés	0			p átlag	0,516666667					
Eltérés	-23%			q átlag	0,483333333					
				stand hiba	0,129027703					
				z	-1,808397173					
				alsó c	-1,644853627					

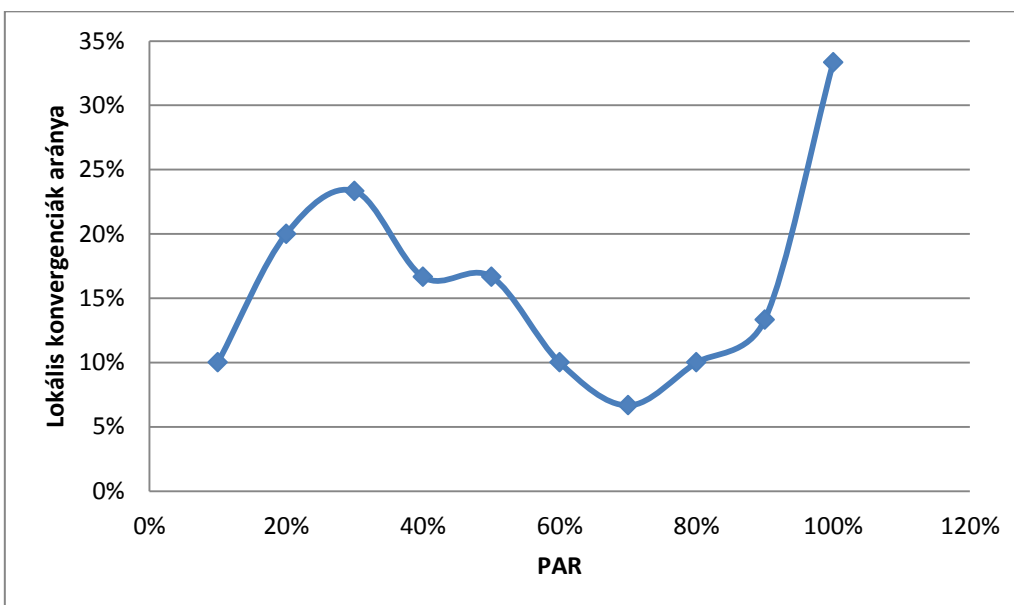
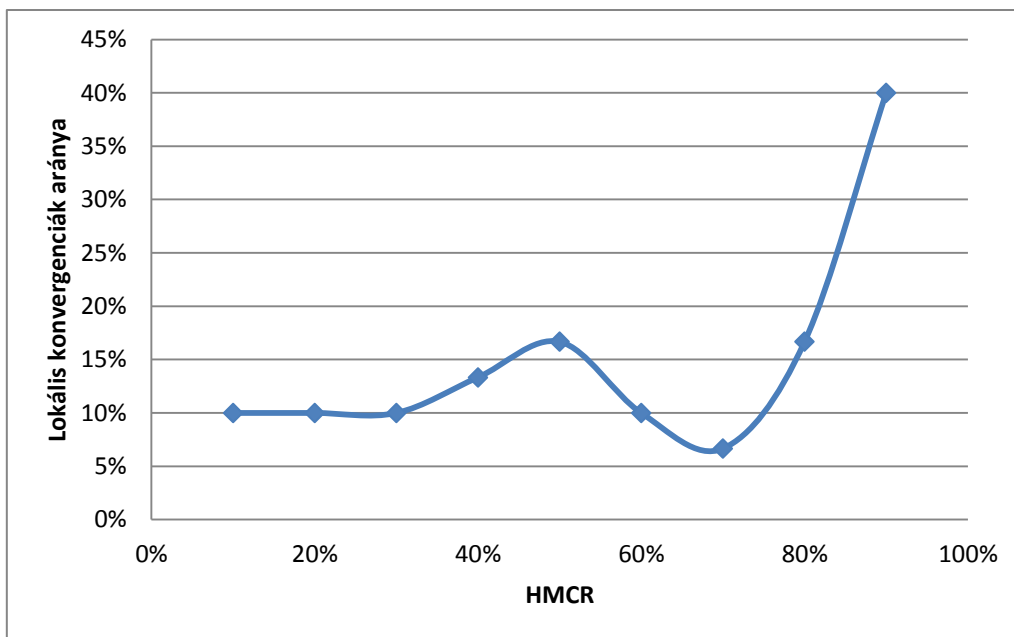
Nullhipotézis elvetve-->Véletlen kereszteződési pont esetén nagyobb a 20 lépésen belül kialakuló stabil konvergenciák aránya.

2. számú melléklet: A HSRS algoritmus paraméterezése

Jobb paraméterezést a lokális, nem stabil konvergenciák arányával mérjük 30 futtatásból.

Minél kisebb ezeknek az aránya, annál hatékonyabb az algoritmus.





3. számú melléklet: Az Improved HSRS algoritmus hatékonysága 4000-es maximális lépésszám esetén

A 4000 lépésből már kialakult stabil konvergenciák arányát vizsgáljuk. Amelyik algoritmusnál nagyobb ez az arány, az a hatékonyabb.

	HS	IHS		H_0	$P(HS)=P(IHS)$					
n	25	25		H_1	$P(HS)<P(IHS)$					
k	14	23								
p	56%	92%								
Próba	bal									
Elvi eltérés	0			p átlag	0,74					
Eltérés	-36%			q átlag	0,26					
				stand hiba	0,124064499					
				z	-2,901716461					
				alsó c	-1,644853627					
Nullhipotézis elvetve-->IHS alkalmazása esetén nagyobb a 4000 lépésből már kialakult stabil konvergenciák aránya.										

4. számú melléklet: Az adatbázis változóinak korrelációmátrixa

	Y	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13	X14	X15	X16
Y	1	- 0,9988	0,2489	0,6209	0,1806	0,6802	0,1189	0,4370	0,6520	0,7792	0,6216	0,6304	0,5119	0,3922	0,5117	0,2981	0,3195
X1	- 0,9988	1	0,2483	0,6227	0,1853	0,6804	0,1267	0,4387	0,6551	0,7786	0,6240	0,6325	0,5151	0,3970	0,5153	0,3045	0,3239
X2	0,2489	- 0,2483	1	0,0119	0,0843	0,0340	0,1598	0,0927	0,0976	0,1339	0,0753	0,1044	0,0257	0,0620	0,0662	0,0380	0,0339
X3	0,6209	- 0,6227	- 0,0119	1	0,5902	0,8838	0,8453	0,8119	0,8897	0,9040	0,9496	0,8674	0,8781	0,7589	0,7689	0,5469	0,7688
X4	0,1806	- 0,1853	- 0,0843	0,5902	1	0,1561	0,6371	0,4074	0,3450	0,3761	0,5041	0,3532	0,5798	0,4617	0,2886	0,2623	0,4393
X5	0,6802	- 0,6804	- 0,0340	0,8838	0,1561	1	0,6537	0,7453	0,8956	0,8934	0,8705	0,8583	0,7438	0,6596	0,7718	0,5061	0,6696
X6	0,1189	- 0,1267	- 0,1598	0,8453	0,6371	0,6537	1	0,7363	0,6819	0,6154	0,7824	0,6740	0,7627	0,7018	0,6448	0,5065	0,7656
X7	0,4370	- 0,4387	- 0,0927	0,8119	0,4074	0,7453	0,7363	1	0,7700	0,7247	0,7136	0,6609	0,6386	0,5414	0,7142	0,5728	0,7307
X8	0,6520	- 0,6551	- 0,0976	0,8897	0,3450	0,8956	0,6819	0,7700	1	0,8816	0,8253	0,7513	0,7290	0,6468	0,7104	0,4525	0,6685
X9	0,7792	- 0,7786	- 0,1339	0,9040	0,3761	0,8934	0,6154	0,7247	0,8816	1	0,8902	0,8372	0,7779	0,6335	0,6759	0,4144	0,6226
X10	0,6216	- 0,6240	- 0,0753	0,9496	0,5041	0,8705	0,7824	0,7136	0,8253	0,8902	1	0,9116	0,8755	0,7242	0,7216	0,4471	0,6874
X11	0,6304	- 0,6325	- 0,1044	0,8674	0,3532	0,8583	0,6740	0,6609	0,7513	0,8372	0,9116	1	0,8209	0,6654	0,7553	0,4434	0,6113
X12	0,5119	- 0,5151	- 0,0257	0,8781	0,5798	0,7438	0,7627	0,6386	0,7290	0,7779	0,8755	0,8209	1	0,7795	0,6685	0,4465	0,6737
X13	0,3922	- 0,3970	- 0,0620	0,7589	0,4617	0,6596	0,7018	0,5414	0,6468	0,6335	0,7242	0,6654	0,7795	1	0,5727	0,5043	0,7602

X14	0,51 17	- 0,5153	0,0662	0,7689	0,2886	0,7718	0,6448	0,7142	0,7104	0,6759	0,7216	0,7553	0,6685	0,5727	1	0,6108	0,6927
X15	0,29 81	- 0,3045	- 0,0380	0,5469	0,2623	0,5061	0,5065	0,5728	0,4525	0,4144	0,4471	0,4434	0,4465	0,5043	0,6108	1	0,5495
X16	0,31 95	- 0,3239	0,0339	0,7688	0,4393	0,6696	0,7656	0,7307	0,6685	0,6226	0,6874	0,6113	0,6737	0,7602	0,6927	0,5495	1

Irodalomjegyzék

- Goldberg, D. E., (1989). Genetic Algorithms in Search, Optimization and Machine Learning. Boston, Kluwer Academic Publishers
- Hajdu, O. (2013). Statisztikai adatok ökonometriai elemzése. Egyetemi jegyzet, Budapesti Corvinus Egyetem
- Holland, J. H., (1975). Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor
- Hunyadi L., Vita L. (2006). Statisztika közgazdászoknak. Központi Statisztikai Hivatal, Budapest
- Johnson R. W., (1996). Fitting Percentage of Body Fat to Simple Body Measurements. Journal of Statistics Education 4(1), 265-266.
- Kovács, L. (2013). Metaheurisztikus algoritmusok hatékonyságvizsgálata adott karaktersorozat előállításában. TDK dolgozat, Budapesti Corvinus Egyetem
- Kovács, P. (2008). A multikollinearitás vizsgálata lineáris regressziós modellekben. Statisztikai szemle 86. évfolyam 1. szám, 38-67.
- 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.
- Minerva, T., Paterlini, S. (2002). Evolutionary Approaches for Statistical Modelling. Fogel DB, El-Sharkawi MA, Yao X., Greenwood G, Iba H, Marrow P, Shackleton A, IEEE Press, Piscataway NJ (2002) Proc.of the Fourth Congress on Evolutionary Computation 2, 2023-2028.
- Minerva, T., Paterlini, S. (2010). Regression Model Selection Using Genetic Algorithms. Proceedings of the 11th WSEAS International Conference on RECENT Advances in Neural Networks, Fuzzy Systems & Evolutionary Computing, 19-27.
- Osman, I. H., Laporte, G., (1996). Metaheuristics: A bibliography. Annals of Operations Research, 63 (5), 511-623.
- Sen, A., Srivastava, M., (1990). Regression Analysis: Theory, Methods, and Application. New York, Springer-Verlag

Talbi, E-G. (2009). Metaheuristics : from design to implementation. John Wiley & Sons, Inc., Hoboken, New Jersey

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/>