PÁRHUZAMOS GENETIKUS ALGORITMUSOK HIBAKEZELÉSI STRATÉGIÁI

ERROR HANDLING STRATEGIES FOR THE PARALLEL GENETIC ALGORITHM

Hatwágner Ferenc Miklós, Horváth András***

ABSTRACT

Genetic algorithms are widely used to solve optimisation problems, e.g. to find the optimal shape of flow domains. These kinds of problems often need high computational power, so parallel implementations of the softwares are essential. Fortunately, it is easy to make a genetic algorithm parallel, using the master-slave architecture. This way the tasks can be shared out between the available computers.

Several types of errors can arise during the optimisation process: computer hardware errors or errors of the modelling software come forward with a considerable probability during a heavy optimisation. We examine some error handling strategies in our paper that can be used in a master-slave style parallel genetic algorithm. The efficiency of these strategies is estimated on an analytic manner and it is measured on an empirical way as well. We present the best obtainable results as a function of the number of processors, the probability of errors and the error handling strategy with the needs of the engineering practice end in view.

Using our results the selection of the best error handling strategy is possible in case of practical optimisation problems.

Keywords: optimisation, genetic algorithm, parallel programming, error handling.

ÖSSZEFOGLALÓ

A genetikus algoritmusokat széles körben használják műszaki optimalizációs problémák megoldásában, mint pl. áramlási tartományok ideális alakjának meghatározása. Az ilyen feladatok gyakran nagy számításigényűek, ezért a párhuzamosítás elengedhetetlen követelmény. Szerencsére a genetikus algoritmusok természetes módon párhuzamosíthatóak a mester-szolga („master-slave”) modell alapján és így a feladatok szétoszthatók a rendelkezésre álló számítógépek között. E folyamat azonban hibákkal terhelt: a számítógépek hardverhibája vagy a modellező szoftver problémái számottevő valószínűséggel jelentkeznek egy nagyobb optimalizálási feladat megoldása esetén.

Cikkünkben megvizsgálunk néhány, a mester-szolga modell segítségével párhuzamosított genetikus algoritmusoknál használható hibakezelési módszert. Analitikus módon is megbecsüljük, és empirikus úton is ellenőrizzük, milyen hatékonyságúak az egyes hibakezelési stratégiák. A mérnöki gyakorlat igényeit szem előtt tartva egy adott futásidő után elérhető legjobb eredményeket mutatjuk be a processzorok száma, a hiba valószínűsége és a hibakezelési módszer függvényében.

Eredményeink ismeretében kiválasztható a legjobb hibakezelési stratégiája a gyakorlatban felmerülő optimalizációs feladatokhoz.

Kulcsszavak: optimalizálás, genetikus algoritmus, párhuzamos programozás, hibakezelés.

1. BEVEZETÉS

A tudományos kutatás és a mérnöki gyakorlat sokszor vet fel olyan problémákat, melynek során azt keressük, hogy egy rendszer bizonyos paramétereit változtatva mikor kapjuk a –bizonyos szempontból– legjobb esetet. Ezek a változó paraméterek lehetnek pl. egy termék geometriai adatai, egy áramköri elem elektromos jellemzői vagy más tervezési változói, melyek kihatnak a termék végső értékére. Ezektől az úgynevezett „tervezési változóktól” való függés sokszor nem egyszerűen számolható, esetleg egy komplett végeselemes vagy áramlástani szimuláció futtatása szükséges egy adott paraméterekkel rendelkező változat értékének megállapításához. (Például annak kiszámolásához, hogy egy helyen változtatva egy lemez vastagságát, milyen kihatással lesz az a teljes, összetett rendszer teherbírására.)

Matematikailag az ilyen problémák sokváltozós, nemlineáris optimalizációs feladatot jelentenek: a változók a rendszer előbb említett tervezési változói, az optimalizálandó célfüggvény pedig valamilyen fontos jellemző (pl. a maximális terhelés vagy egy áramlástani paraméter eltérésnek négyzete egy előírt értéktől). Ezeket a nemlineáris optimalizálási problémákat igen nehéz megoldani, ha a változók száma meghaladja a 4-5-öt, és egy-egy „függvény-kiértékelés” valójában egy komplett modell lefuttatása, ami 10 perctől kezdve akár több órán át is tarthat. (Lásd pl. [1], [2], [3].)

** doktorandusz, Széchenyi István Egyetem, Informatika Tanszék*
*** témavezető, Széchenyi István Egyetem, Fizika és Kémia Tanszék*

Az ilyen jellegű problémákra a genetikus algoritmusok alkalmazása számos esetben mutatkozott sikeresnek: ezek sok változó, a célfüggvény deriváltjának rendelkezésre nem állása és a célfüggvényben mutatkozó sok lokális minimum, oszcilláció és zaj esetén is jó eredményt adnak, még akkor is, ha viszonylag kevés célfüggvény-kiértékelésre van módunk.

2. A GENETIKUS ALGORITMUSOK ALAPGONDOLATA

A genetikus algoritmusok alapgondolatát a természettől, az „alkalmasabb" egyedek nagyobb valószínűséggel történő szaporodásának tényéből (természetes kiválasztódás) és a mutációk által bekövetkező véletlenszerű változatok kialakulásából vették a kutatók. Az ilyenfajta algoritmusoknak számtalan változata létezik, melyekről jó összefoglalót olvashatunk pl. [4], [5]-ben. Az alapfogalmak azonban mindegyik változatban azonosak:

- a tervezési változók listáját „génlánc"-nak hívjuk,
- „egyed"-nek nevezünk egy adott génlánccal leírható modellt,
- „populáció"-nak hívjuk az egyedek egy halmazát,
- „kiértékelés"-nek nevezzük azt a számolást, mely meghatározza, a célfüggvény értékét, azaz hogy egy adott génlánchoz tartozó modell mennyire teljesíti elvárásainkat.

E fogalmakat használva egy tipikus genetikus optimalizáció sémája a következő:

1. Generálunk egy 20-100 elemű populációt
2. Kiértékeljük a populáció egyedeit.
3. Egy átmeneti populációt hozunk létre, melybe a „jobb" egyedek nagyobb valószínűséggel kerülnek bele.
4. Az átmeneti populáció egyedeiből választott véletlenszerű párokat „keresztezzük", azaz véletlenszerűen kiválasztott helyen elvágjuk génláncaikat és a darabokat kicseréljük, az eredményt beletesszük az új generációt jelentő populációba.
5. Az új populációban néhány véletlenszerű egyed génláncán véletlenszerű változást (mutáció) hajtunk végre.
6. Ha még van időnk, visszatérünk a 2-es lépésre.

Látható, hogy ez a módszer nem garantálja az optimum megtalálását, de azt sok változó esetén nem is lehet biztosítani semmilyen módszerrel sem, mégis ez a séma az „életrevalóbb" egyedek génlánc-darabkáinak elterjedését és a véletlenül sikeres mutációk megőrződését biztosítva egyre jobb egyedekből álló populációkat „tenyészt" ki. (Ez a folyamat hasonlít a bizonyos tulajdonságokat irányítottan kiemelő állat- vagy növénynemesítési módszerekre.)

A számolási ciklusoknak összetett célfüggvények esetén általában a rendelkezésre álló idő szab határt és ennek leteltekor a populáció legjobb egyedéről a probléma ismeretében a terület szakértője tudja eldönteni, hogy elég jó-e az.

3. A GENETIKUS ALGORITMUSOK PÁRHUZAMOSÍTÁSA

Gyakorlati problémák esetén a fenti séma messze legidőigényesebb része a 2. lépés, hisz ez több tucat, különböző paraméterű modell végigszámolását jelenti, ami sok óráig vagy akár pár napig is tarthat egy processzoron, pedig ez még nem is a teljes optimalizációs folyamat.

Szerencsére nincs akadálya annak, hogy egy populáció egyedeit egyidejűleg értékeljük ki, és ezzel töredékére csökkentsük a futási időt, ha több számítógép is a rendelkezésünkre áll. Ha több processzorunk van (akár különböző számítógépekben), mint a populáció mérete, akár egyetlen szimuláció elvégzéséhez szükséges idő alatt valamennyi egyedet kiértékelhetjük.

A genetikus algoritmusok többféleképpen is párhuzamosíthatóak; erről jó áttekintést találunk pl. a [6] cikkben. Mindezek közül talán a legegyszerűbben megvalósítható a „mester-szolga" (master-slave) modellen alapuló megoldás [7], [8], [9]. Ebben az esetben a „mester" valósítja meg a genetikus algoritmust, a genetikus operátorokat (kereszteződés, mutáció), és csak az egyedek kiértékelésének erőforrás-igényes feladata hárul a „szolgákra". Így akár egy intézet különböző helységeiben levő egyedi gépei is befoghatók a munkára a hálózaton keresztül, ún. „számítógép-klaszterbe" szervezve őket. Ez a megoldás széles körben elterjedt, mert így az egyes munkaállomások egyébként kihasználatlanul álló kapacitása is munkára fogható, így kis költséggel nagy számítási kapacitás vonható be az optimalizálásba. (Ugyanilyen elosztott rendszer általában nem hatékony egy-egy kiértékelés önmagában való felgyorsítására, mert egy feladat megoldása sokszor nem párhuzamosítható hatékonyan.)

4. A HIBÁK HATÁSA, A PROBLÉMA FELVETÉSE

A mester-szolga séma magától értetődő, de a gyakorlat azt mutatja, hogy az ilyenkor fellépő, hosszú ideig tartó futtatások esetében számolni kell különféle hibák megjelenésével. Ezek egy része hardveres okokra vezethető vissza: a klaszterben lévő számítógépek valamelyike lefagyhat pl. túlmelegedés miatt, vagy leállíthatják, ha nincs felügyeletünk alatt. Egy véletlenül kihúzott hálózati csatlakozó, esetleg áramszünet szintén gondot okozhat. A problémák másik része szoftveres természetű: a szimuláció elvégzését gyakran harmadik féltől származó szoftverek (pl. ANSYS, FLUENT, SYSNOISE) végzik ([10], [11], [12]), mert olyan bonyolult műveletekre van szükség, mint pl. különféle térbeli hálók elkészítése, végeselem módszerek alkalmazása (FEM), vagy áramlástani problémák megoldása (CFD). Előfordul, hogy ezek a szoftverek váratlanul leállnak, vagy nem várt eredményt produkálnak. Ennek pl. az lehet az oka, hogy

a megoldandó feladat az adott feltételek mellett, az adott paraméter-értékek segítségével nem oldható meg, vagy nem értelmezhető.

Az ilyen esetekben a mesterre vagy valamilyen keretrendszerre (pl. MPI, PVM, Globus) hárul a feladat, hogy a hibákat alkalmas módon kezelje. Amennyiben a hiba természetének ismeretében van értelme újból megkísérelni a célfüggvény kiértékelését, jellemzően vagy az azonos processzoron való ismételt futtatással, vagy a feladat más processzorra történő átvitelével szoktak próbálkozni (lásd pl. [13]-ban).

Kérdés azonban, megéri-e várni, hátha az újabb próbálkozás sikeres lesz, vagy inkább hagyjuk figyelmen kívül és vegyük igen rossz értékűnek a sikertelen számításokban részt vevő egyedeket. Ha várunk, az a futásidőt növelheti, miközben a már sikeresen végzett processzorok tétlenül várakoznak, ha „eldobjuk" a rossz számításban részt vevő egyedet, akkor pedig esetleg valójában jó modellt hagyunk figyelmen kívül, aminek csak nem volt szerencséje és épp nála következett be valamilyen fatális esemény, mondjuk a kiértékelést végző számítógépet érintő áramszünet.

Cikkünkben az e téren végzett kutatásaink eredményét mutatjuk be: a populációméret, a processzorok száma és a hibák bekövetkezésének valószínűsége függvényében több lehetséges hibakezelési stratégia összevetését végezzük el. Az eredmények ismeretében a nagy számításidejű optimalizációk esetén kiválasztható a követendő hibakezelési módszer.

5. A PROBLÉMA MEGHATÁROZÁSA

Legyen N_p a populáció mérete és N_c a rendelkezésünkre álló processzorok száma. A célfüggvény kiértékelések során p_e valószínűséggel következik be valamilyen hiba. Az egyszerűség kedvéért feltételezzük, hogy minden célfüggvény kiértékelése pontosan ugyanannyi ideig tart az azonos számítási teljesítményű processzorokon és ehhez képest a mester-szolga gépek közti kommunikáció és a genetikus operátorok végrehajtási ideje elhanyagolható.

A hibakezelési stratégiákat oly módon hasonlítjuk össze, hogy adott futásidő után megvizsgáljuk a különféle hibakezelési módszerek használatával kapott legjobb egyed célfüggvényének értékét. Az nyilvánvaló, hogy a hibák valamilyen módon rontani fogják az eredményeket.

Amennyiben nem lép fel hiba *($p_e = 0$),*

$$U_0 = \mathrm{ceil}\left(\frac{N_p}{N_c}\right) \tag{1}$$

„kiértékelési körre" van szükség a teljes populáció kiértékeléséhez. (Itt „ceil" a következő egészre felkerekítő függvény.)

Ha eközben hiba lép fel, és a hibás egyedek ismételt kiértékelése mellett döntünk, akkor nem veszítünk információt, de a számítási körök száma nő, ami együtt jár a futásidő növekedésével. Ellenben ha a hibás értékek figyelmen kívül hagyását választjuk, akkor a futásidő nem változik, viszont leromlik a legjobb egyed célfüggvény értéke, mert hibás kiértékelési eredményt vittünk a populációba.

Háromféle hibakezelési módszert fogunk megvizsgálni:

- újraszámoló módszer,
- eldobó módszer,
- hibrid módszer.

Újraszámoló módszer

Kezdetben megpróbáljuk kiértékelni az N_p egyedet U_0 számítási körben. Amennyiben valamelyik egyed kiértékelése közben hiba lépett volna fel, akkor azt megkíséreljük ismételten kiértékeltetni valamelyik szolgával.

Ebben az esetben adott számú generáció után az eredmény megegyezik a hibamentes esetben kapottal, azonban az optimalizáció lelassul, mert a számítási körök átlagos száma *(U)* nagyobb lesz U_0-nál.

Másképp fogalmazva, a szoftver csak kevesebb generáció adatait tudja kiértékelni adott idő alatt. Ez azért különösen fontos, mert az optimalizációk elvégzésére általában meghatározott (kevés) idő áll rendelkezésre.

Eldobó módszer

Megkíséreljük az N_p egyed kiértékelését U_0 számítási körben. Amennyiben egy szolga hibát jelez vagy nem válaszol, akkor a mester a megfelelő egyed célfüggvény értékét „nagyon rosszra" állítja. (Lásd pl. [14]) Pl. ha a célfüggvény minimalizálása a cél, akkor a végtelen jó választás lehet a „nagyon rossz" értékének.

A mester ilyen viselkedésének az lesz a következménye, hogy a megfelelő „rátermettségi érték" is nagyon alacsony lesz, tehát az egyednek rendkívül lecsökken az esélye arra, hogy továbbadja genetikai információját.

Néhány egyed kiértékelésének elmulasztása természetszerűleg rontja az adott számú generáció után elérhető eredményt, ugyanakkor ez a módszer nem lassítja le az optimalizációt. Ez egy versenyképes módszer lehet, mert a legjobb egyed célfüggvény értékének romlása minimális, mivel általában egy populációban több hasonló egyed is van, ugyanakkor jelentős futásidő takarítható meg, mivel a kiértékelési körök száma marad U_0.

A genetikus algoritmusok biológiai analógiái alapján ez a stratégia annak felel, meg, hogy bizonyos valószínűséggel egyébként életképes egyedek is meghalhatnak valamilyen baleset folytán, pl. egy villámcsapás következtében. Az állatok esetéből is tudjuk, hogy egy-egy, még oly kiváló egyed elvesztése sem érinti jelentősen a teljes populáció fejlődését, mert még ha el is pusztul a legerősebb vagy legokosabb, a populációban találhatók hasonló génkészletű, majdnem ugyanolyan jó egyedek és így összességében nem romlik annyira a teljes populáció, mint először gondolnánk.

Hibrid módszer

Ez a módszer az előző kettő egyfajta keverékeként fogható fel. Amennyiben a hibás kiértékelés megismétlése

nem jár a futásidő növekedésével (újabb számítási kör megkezdésével), mert van olyan processzorunk, ami különben tétlenül várakozna, akkor a mester az adott generációhoz tartozó utolsó számítási körében utasítja valamelyik tétlen szolgát a számítás megismétlésére. Más esetben a mester az eldobó módszernek megfelelően jár el.

Például, ha egy klaszter hat szolga számítógépet tartalmaz *($N_c = 6$)*, akkor hét számítási körre van szükség egy $N_p = 40$ egyedből álló populáció kiértékeléséhez, de az utolsó számítási körben két tétlen szolga is marad. Amennyiben az első hat számítási körben legfeljebb két hiba történt, akkor a célfüggvény kiértékelések az egyébként tétlenül várakozó processzorokon megismételhetőek a futásidő növekedése nélkül. Több hiba esetén azonban nem mindegyik egyed ismételt kiértékelésével próbálkozunk, hanem az eldobó módszerhez hasonlóan azokat nagyon rossz értékű egyednek vesszük.

Nyilvánvaló, hogy ha $N_p mod N_c = 0$, akkor ez a módszer pontosan az eldobóval egyezik meg. Minden más esetben azonban kevesebb célfüggvény kiértékelés vész el, a futásidő azonban változatlan marad. Tehát azonos futásidőt követően (U_0 kiértékelési kör) ugyanolyan, vagy jobb eredményeket fogunk kapni, mint az eldobó módszer esetében.

6. A MÓDSZEREK ÖSSZEHASONLÍTÁSA

Az előző három módszer összehasonlítására az optimumszámításban standardnak számító egyszerű, ismert optimumú célfüggvényekre alkalmaztuk ezeket, mesterségesen beépítve a programba azt, hogy p_e valószínűséggel a kiértékelés helyett egy igen rossznak számító értékkel térünk vissza. Ilyen körülmények közt azt vizsgáltuk, hogy adott számú kiértékelési kör után az egyes módszerek által adott legjobb egyed mennyire közelíti meg az ismert optimum értékét. Az egyszerű, gyorsan kiértékelhető tesztfüggvények használata lehetővé tette, hogy a méréseket minden esetben 25-ször megismételjük, lecsökkentve ezzel a genetikus algoritmusok véletlenszerű jellegéből adódó statisztikai hibát.

A három különböző stratégia eltérő módon érzékeny a bekövetkező hibákra. Ezek hatását, ha lehet, analitikus számításokkal, ha ez nem volt alkalmazható, numerikus kísérletekkel vizsgáltuk.

Az újraszámoló módszer hatékonysága

Az újraszámoló módszer minden számítást elvégez, ami a hiba nélküli esetben megtörténne, a hatékonyság csökkenése abból adódik, hogy az újraszámolás esetleg több kiértékelési kört jelent, azaz ugyanarra az eredményre több időt kell várnunk, tehát $U \geq U_0 = ceil(N_p / N_c)$ lesz. Az alábbiakban megvizsgáljuk, hogyan függ a futásidővel arányos U érték N_p, N_c és p_e paraméterektől.

Két, egymástól jelentősen különböző esetet kell vizsgálni a processzorszám és a populációméret viszonya függvényében: amikor $N_c \geq N_p$ és amikor $N_c < N_p$.

1. eset: $N_c \geq N_p$

Ekkor $U_0 = 1$, azaz minden számítási kör képes egy teljes populációt kiértékelni, ha nem lép fel hiba. Hiba esetén viszont új számítási körre lesz szükség, de az új körben is felléphet hiba, ezért még újabb kör válhat szükségessé egyre kisebb és kisebb számú elemmel.

Legyen p_i annak valószínűsége, hogy pontosan i számítási körre van szükség. Ekkor a kiértékelési körök számának várható értéke nyilván:

$$U\left(N_p, p_e\right) = \frac{\sum_{i=1}^{\infty} i p_i}{\sum_{i=1}^{\infty} p_i} \qquad (2)$$

p_i kiszámítási módját N_p, N_c és p_e függvényében [15] tartalmazza.

Az újraszámoló módszer esetén tehát az időigény U/U_0 szorosára növekszik, de a végeredmény nem változik.

2. eset: $N_c < N_p$.

Ekkor $U_0 > 1$, azaz hibák nélkül is több kiértékelési körre van szükség. Nincs szükség azonban további körökre, ha

- a nem-utolsó körben összesen fellépő hibás egyedek száma legfeljebb $N_c - N_p mod N_c$,
- és nem lép fel hiba az utolsó számítási körben.

Az 1. esethez képest sokkal bonyolultabbal állunk itt szemben, ezért analitikusan nem tudjuk teljesen pontosan megadni U-t, csak egy alsó becslést adunk az alapján, hogy feltételezzük, hogy legfeljebb 1 plusz körre van szükség. Látni fogjuk, hogy p_e életszerű értékei esetén jó eredményeket kapunk ezzel, mert ritkán következik be, hogy épp az utolsó körben hiba lép fel vagy a korábbiakban annyi hiba lépett fel, hogy egy extra kiértékelési kör nem elegendő.

Az biztos, hogy ha N_p osztható N_c-vel, akkor nincs az utolsó körben tétlen processzor, így egy hiba esetén is új kiértékelési kört kell indítani, különben pedig van esély arra, hogy az egyébként feladat nélküli processzorok ellátják a korábbi hibás kiértékelések megismétlését. Így két további alesetet kell megvizsgálnunk:

2/a. eset: $N_p mod N_c = 0$

Ebben az esetben az utolsó számítási körben is dolgozik az összes processzor. Minden egyes generáció kiértékeléséhez U_0 számítási kör szükséges. Amennyiben feltesszük, hogy egy korábbi hiba miatt indított további számítási körben nem lép fel ismét hiba, alsó korlátot adhatunk U értékére a következőképpen:

$$U > U_0 + 1 - (1-p_e)^{Np} \qquad (3)$$

2/b. eset: $N_p mod N_c \neq 0$

Ebben az esetben az utolsó számítási körben nem dolgozik mind az N_p processzor, csupán $N_w = N_p - (U_0 - 1) N_c$. Ebben az esetben is megadható egy alsó korlát U-ra a következőképpen:

$$U > U_0 + 1 - (1-p_e)^{N_w} \qquad (4)$$

U pontosabb értékét a folyamat Monte Carlo szimulációjával határoztuk meg. Az *1. táblázat* a 40 elemű po-

puláció *(N_p=40)* és különböző N_c, p_e értékekre mutatja *U* értékeit.

1. táblázat
U értékek $N_p = 40$ esetén

N_c \ p_e	0,001	0,01	0,05
40	1,040	1,335	1,972
35	2,005	2,053	2,309
20	2,040	2,335	2,972
18	3,004	3,043	3,271
10	4,040	4,335	4,972
9	5,004	5,043	5,276

A táblázat értékeit a (3) és (4) egyenletekben megadott alsó korlátokkal összevetve azt láthatjuk, hogy még a $p_e = 0,05$ esetben is csak 1–2% százalék az eltérés a formulák szerinti alsó korlát és a tényleges érték között.

Figyeljük meg, hogy *U* már p_e alacsony értékei esetén is jelentősen nagyobb lesz U_0-nál. Például az $N_c = N_p$ esetben 0,1% hiba valószínűség mellett átlagosan 4%-kal nő a számítási körök száma, 1% hiba esetén pedig több mint 30%-os az eltérés. Ez azt jelenti, hogy még igen kis valószínűségű hiba is jelentősen csökkenti a hatékonyságot.

Az eldobó és a hibrid módszerek hatékonysága

A fent elmondottak szerint ezek a módszerek nem igényelnek új kiértékelési köröket. Hatékonyságcsökkenésük abból fakad, hogy néha nem az egyedek valódi értékét, hanem egy mesterséges, igen rossznak vett értéket vesznek alapul, ami nyilván megzavarja az algoritmus működését. Ennek hatása azonban csak numerikus kísérletekkel tanulmányozható.

7. TESZT SZÁMÍTÁSOK

A hibakezelési módszerek hatékonyságának tanulmányozása céljából több tesztfuttatást is végeztünk a saját fejlesztésű, [14]-ben ismertetett szoftver segítségével. A futtatások során $N_p = 40$ egyedből álló populációt használtunk és az egyedeket 20 gén alkotta. A mutáció valószínűségét 14%-ra állítottuk. Az optimalizációt 1500 generáció (G) kiértékelését követően állítottuk le. A további beállítások a következők voltak: reciprok rangsorolás, rulett kerék kiválasztás, egypontos keresztezés, kényszerített mutáció (távolság paraméterének értéke 10^{-5}), valamint a hegymászó algoritmus egy speciális változatát alkalmaztuk a lokális szélsőérték pontosabb meghatározására. Az operátorokról részletesebben [4] ír.

Minden számítást 25 alkalommal ismételtünk meg, az ál-véletlenszám generátort különféle kezdőértékekkel inicializálva. Az alábbiakban a futtatások átlagát ábrázoljuk.

A diagramokon a legjobb egyed (B) célfüggvény értékét ábrázoltuk a futásidő *(T_{wall})* függvényében. T_{wall}-t az egyetlen egyed kiértékeléséhez szükséges időegységben mérjük, azaz $T_{wall}=1$ egy kiértékelési körnek felel meg.

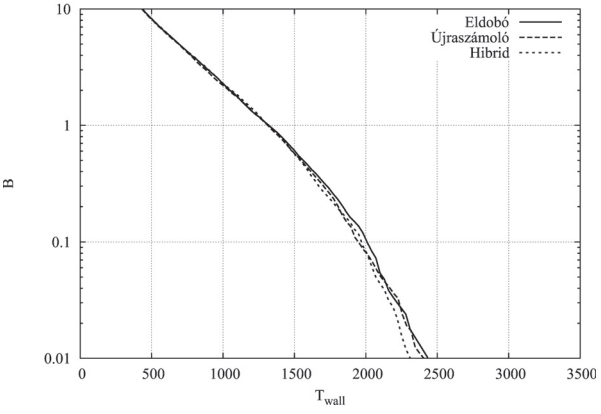
A mérnöki gyakorlat szempontjából legérdekesebb kérdés az, hogy melyik módszerrel lehet a legjobb eredményt elérni rögzített T_{wall} esetén? A 2. pontból tudjuk, hogy ha $N_c \geq N_p$, akkor az eldobó és a hibrid módszer ekvivalens, és jobb, mint az újraszámoló módszer. Nyilvánvaló, hogy a fenti 2/a. esetben is hasonló a helyzet. Az azonban nem magától értetődő, hogy mi történik a 2/b. esetben, tehát akkor, amikor N_p nem többszöröse N_c-nek. Ilyenkor ugyanis néhány processzor munka nélkül marad az utolsó számítási körben, így ezeket fel lehet használni a korábbi számítási körökben keletkezett hibás kiértékelések megismétlésére.

A Rastrigin-függvény vizsgálata

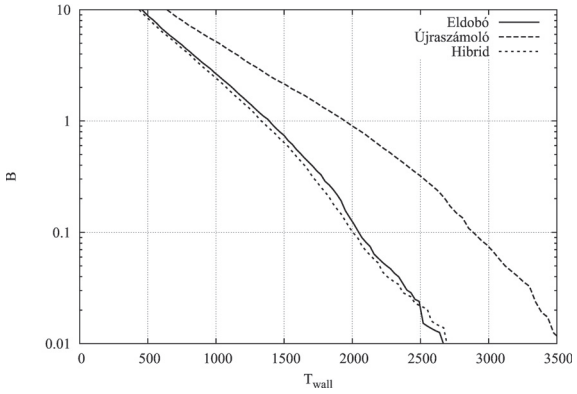
A Rastrigin-függvény széles körben használatos többváltozós optimum-keresési módszerek tesztelésére, mert folytonos, deriválható, de sok lokális minimuma van, és optimuma ismert: ha minden változója 0, akkor a 0 értéket veszi fel, mindenütt máshol pozitívot. Ez a tulajdonság lehetővé teszi, hogy B értékeit logaritmikus skálán ábrázoljuk, olvashatóbbá téve ezzel a grafikonokat. Ezekről a grafikonokról jól látszik az egyes módszerek hatékonysága: mind az adott számítási idő alatt elért eredmény pontossága, mind az adott pontosság eléréséhez szükséges idő könnyen leolvasható.

$$f_{\text{Rastrigin}} = n \cdot 10 + \sum_{i=1}^{n} \left(x_i^2 - 10\cos\left(2\pi x_i\right) \right) \quad -5.12 \leq x_i \leq 5 \tag{5}$$

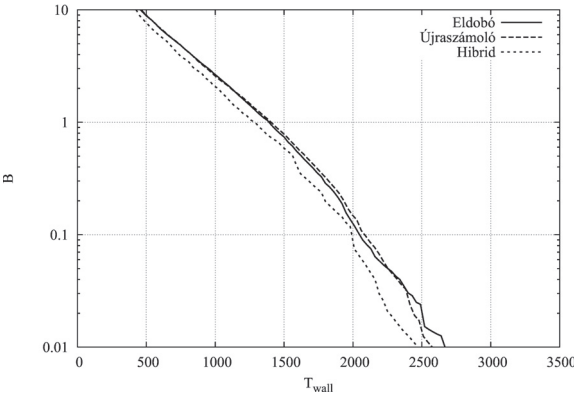
A grafikonok adatai úgy keletkeztek, hogy az újraszámoló módszer esetében egyszer, hibák szimulálása nélkül elvégeztük a futtatásokat és ezt az adatsort jelenítjük meg mindig úgy, hogy a fentiek szerint T_{wall}-t U/U_0-szorosan megnyújtjuk. Az eldobó és hibrid módszereknél viszont nem lehet a viselkedést ilyen egyszerű skálázódással követni: a számítást végző programba beleépítettük, hogy p_e valószínűséggel egy mesterséges, igen rossz értékkel térjenek vissza és minden p_e érték esetén újra kellett futtatni, mert nem volt előre látható, hogyan rontják a hatékonyságot a hibák.



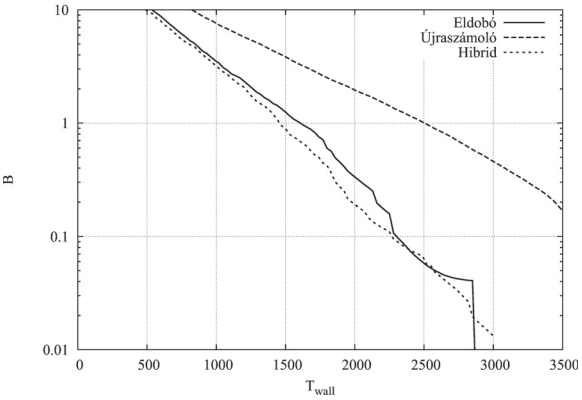
1. ábra. A Rastrigin függvény minimalizálása, G = 1500, p_e = 0,001, N_p = 40, N_c = 35



2. ábra. A Rastrigin függvény minimalizálása, $G = 1500$, $p_e = 0{,}05$, $N_p = 40$, $N_c = 20$



3 ábra. A Rastrigin függvény minimalizálása, $G = 1500$, $p_e = 0{,}05$, $N_p = 40$, $N_c = 39$



4. ábra. A Rastrigin függvény minimalizálása, $G = 1500$, $p_e = 0{,}25$, $N_p = 40$, $N_c = 35$

Az eredmények nyilvánvalóak. A statisztikai hibáktól eltekintve a hibrid módszer soha nem teljesített rosszabbul, mint az eldobó. A [14]-ben foglaltakkal összhangban az eldobó módszer általában jobb, mint az újraszámoló. A szimulációk elvégzése előtt elképzelhetőnek tűnt, hogy az $N_p \bmod N_c \neq 0$ esetben az újraszámoló módszer jobban fog szerepelni, mint az eldobó, az utolsó számítási körben lévő kihasználatlan processzorok miatt. Az eredmények azonban azt mutatják, hogy az eldobó módszer van olyan jó, mint az újraszámoló (és egyértelműen a hibrid módszer a győztes).

A vizsgálatokat más tesztfüggvényekre (pl. a Keane-függvény, lásd [2]) is elvégeztük, de az eredmények jellege ugyanez maradt.

8. KÖVETKEZTETÉS

Az eldobó és hibrid módszerek gyakorlatilag soha nem adnak rosszabb eredményt, mint az újraszámoló. A legjellemzőbb esetekben a hibrid módszer lényegesen jobb eredményeket produkált, mint az eldobó. Ez megfelel a várakozásainknak, hiszen a hibrid módszer lényegében az eldobónak egy kisebb adatvesztést eredményező, továbbfejlesztett változata. A tesztek megmutatták, hogy ha N_p többszöröse N_c-nek, az újraszámoló módszer jelentős hátrányban van a másik kettővel szemben, amelyek ebben az esetben ekvivalensek. Ha N_p nem többszöröse N_c-nek, akkor az újraszámoló és az eldobó módszer közötti különbség nagyon kicsi, de a hibrid módszer jobban szerepel, mint az eldobó.

Az újraszámoló módszer alkalmazása rendkívül rossz hatékonyságú, ezért alkalmazását nem javasoljuk olyan esetekben, amikor a futási idő korlátozott. Az eldobó módszert könnyű megvalósítani, és nem találtunk olyan esetet, amikor az újraszámoló jobban teljesített volna, bár ez elméletileg lehetséges volna N_c és N_p speciális értékei esetén.

A győztes a hibrid módszer, amely kissé összetettebb, mint a másik kettő, de egyértelműen a legjobb eredmények elérését teszi lehetővé.

Végső következtetésként kijelenthetjük, hogy a genetikus algoritmus számítógépes klaszteren mester-szolga módszerrel való párhuzamos megvalósítása esetén a hibrid módszert ajánlott megvalósítani, nevezetesen az utolsó számítási körben egyébként kihasználatlanul álló processzorokat fel kell használni a korábbi számítási körökben hibák miatt ki nem értékelt egyedek ismételt kiértékelésére, de új számítási kört nem célszerű indítani.

SUMMARY

The neglecting and hybrid strategies never produce worse results than re-computing. In the most typical cases the hybrid strategy produced significantly better results than neglecting.

The result lives up to our expectations, because the hybrid strategy is an advanced version of the neglecting strategy, resulting less data loss. The tests showed that if N_p is a multiple of N_c the re-computing strategy is disadvantaged. (The other two strategies are equivalent in this case.) If N_p is not a multiple of N_c then the difference between re-computing and neglecting is very small, but hybrid strategy works better than neglecting.

The efficiency of re-computing is very bad, so we do not recommend using it in case of time-critical applications.

It is easy to implement the neglecting strategy, and we did not find a case when re-compute would have performed better, but theoretically it is possible with special N_c and N_p values.

The winner is the hybrid strategy which is a bit more complicated than the other two, but it makes possible to get the best results.

Our final conclusion is that the recommended error handling strategy for master-slave kind of parallel genetic algorithms is the hybrid one. It means that in the last computational round the idle processors should be used to re-calculate the objective function value of the erroneous individuals. It does not worth to start a new computational round for that reason.

HIVATKOZÁSOK JEGYZÉKE

[1] HORVÁTH A., HORVÁTH Z. Optimal shape design of diesel intake ports with evolutionary algorithm, Proceedings of 5th European conference on numerical mathematics and advanced applications (ENUMATH 2003) Edited by Feistauer, M. et al., Springer Verlag, 2004, pp. 459–470.

[2] MARCO-BLASZKA N., DÉSIDÉRI J. Numerical solution of optimization test cases by genetic algorithms, INRIA Research Report, 3622, 1999.

[3] MARCO N., LANTERI S., DÉSIDÉRI J., MANTEL B., PÉRIAUX J. Parallel genetic algorithms applied to optimum shape design in aeronautics, Lecture Notes in Computer Science (Euro-Par'97 Parallel Processing), Vol. 1300/1997, Springer, 1997, pp. 856 –863.

[4] GOLDBERG D. E. Genetic algorithms in search, optimization, and machine learning, Addison-Wesley Publishing Company, Inc. 1989.

[5] ÁLMOS A., GYŐRI S., HORVÁTH G., VÁRKONYINÉ K. A. Genetikus algoritmusok Szerk. Várkonyiné Kóczy Annamária, Typotex, Bp., 2002.

[6] DUDY LIMA, YEW-SOON ONGA, YAOCHU JINB, BERNHARD SENDHOFFB, BU-SUNG LEEA Efficient Hierarchical Parallel Genetic Algorithms using Grid computing, Future Generation Computer Systems, Volume 23, Issue 4, May 2007, Pages 658-670

[7] CANTU-PAZ E. Designing efficient master-slave parallel genetic algorithms, Genetic Programming 1998: Proceedings of the Third Annual Conference, University of Wisconsin, 1998, pp. 455–463.

[8] GAGNÉ C., PARIZEAU M., DUBREUIL M. The master-slave architecture for evolutionary computations revisited, Proceedings of Genetic and Evolutionary Computation (GECCO 2003) Edited by Cantu-Paz, E. et al., Springer, 2003, pp 1578–1579.

[9] ASTEASUAIN F., CARBALLIDO J. A., VAZQUEZ G. E., PONZONI I. Using computational intelligence and parallelism to solve an industrial design problem, Lecture Notes in Computer Science, (Advances in Artificial Intelligence – IBERAMIA-SBIA 2006), Springer, Vol. 4140, 2006, pp. 188–197.

[10] Ansys Inc., [online]. http://www.ansys.com.

[11] CFD Flow Modeling Software and Services from Fluent Inc., [online]. http://www.fluent.com.

[12] Sysnoise, [online]. http://www.lmsintl.com SYSNOISE.html.

[13] HERRERA J., HUEDO E., MONTERO R. S., LLORENTE I. M. Embarrassingly distributed and master-worker paradigms on the grid, Lecture Notes in Computer Science (Scientific Applications of Grid Computing), Springer, Vol. 3458, 2005, pp. 108–119.

[14] HATWAGNER M., HORVÁTH A. The effect of computer network errors on genetic algorithms, Pollack Periodica, Vol. 2, No. 2, 2007, pp. 3–12.

[15] HATWÁGNER, F. MIKLÓS AND HORVÁTH, A.: Error handling techniques of genetic algorithms in parallel computing environment, Pollack Periodica, volume Volume 3, number Number 2, pages 3-14, ISSN 1788-1994, 2008.