József Attila Tudományegyetem

Csallner András Erik

Intervallum-felosztási eljárások a globális optimalizálásban

doktori értekezés

Témavezető:
Dr. Csendes Tibor
egyetemi docens

Szeged, 1999

Ezúton szeretném megköszönni elsősorban témavezetőm, Dr. Csendes Tibor fáradozását, építő jellegű ötleteit és türelmét, mellyel nélkülözhetetlen segítséget nyújtott ezen dolgozat megírásához. Nem különben köszönet illeti Markót Mihály Csabát, aki a C-XSC programozású tesztfuttatások elvégzését átvállalta.

Köszönet Dr. Imreh Balázsnak és Dr. Megyesi Lászlónak, akik egyetemi tanulmányaim során megismertették és megszerettették velem az optimalizálás témakörét. További köszönettel tartozom Dr. Friedler Ferencnek és Kovács Zoltánnak, akik nélkül nem kezdtem volna komolyabban globális optimalizálással foglalkozni, és akik gyakorlati problémákat adtak a kezembe és elvezettek Dr. Csendes Tiborhoz, hogy megtalálhassam magamnak az intervallum-felosztási módszerek robusztus és megbízható eszköztárát.

Végül, de nem utolsósorban hálával tartozom feleségemnek, aki eltűri, hogy kevesebb időt töltök vele, mint kedvenc számítógépem és könyveim társaságában.

Tartalomjegyzék

Algoritmusok jegyzéke

Ábrák jegyzéke

Táblázatok jegyzéke

Előszó

Az optimalizálás a matematika, illetve informatika igen széles területe, mely számos elméleti és gyakorlati problémát, módszert és eredményt foglal magába. Ez a terület a matematika olyan ágainak jelentős részét képezi, mint az analízis, a numerikus matematika vagy az operációkutatás megfelelő fejezetei.

Ennél szűkebb a folytonos feladatok köre, mely nem tartalmazza a kombinatorikus és egyéb diszkrét problémák, illetve azok megoldásának vizsgálatát.

A folytonos optimalizálási feladatok területe is tovább bontható különböző szempontok szerint. Az egyik ilyen szempont a korlátozási feltételek létezése. Amennyiben az optimalizálandó függvény, azaz a célfüggvény értelmezési tartományának minden pontja szóba jöhet a feladat megoldásánál, úgy korlátozási feltételek nélküli optimalizálási feladatról beszélünk. A független változók összefüggéseire adott megszorító feltételek, azaz ún. korlátozási feltételek létezése esetén a célfüggvény értelmezési tartományának csupán azon pontjai jöhetnek szóba, melyek az adott feltételeket kielégítik.

Egy másik szempont a célfüggvény és az esetleges korlátozási feltételek speciális tulajdonságai, mint a linearitás vagy konvexitás. Ezek kihasználása nagyban megkönnyítheti a problémák vizsgálatát és megoldását, de nem ad választ az általános problémák megoldására. A későbbiekben részletesen tárgyalt eljárások vizsgálatakor főként az általános esettel foglalkozunk, mely semmilyen megszorítást nem tesz a célfüggvények alakjára vonatkozóan, kivéve, hogy kiszámításuk megfelelő közelítéssel számítógép segítségével lehetséges legyen.

A folytonos optimalizálási feladatok egy további osztályozási szempontja a megoldások globális volta. Ha egy adott feladat aktuális megoldása olyan, hogy nem adható jobb, az esetleges korlátozási feltételeket kielégítő megoldás a célfüggvény értelmezési tartományán, akkor a megoldást globálisnak nevezzük. Abban az esetben, ha ez az adott megoldásnak csupán valamilyen környezetére érvényes, a megoldás lokális. Az utóbbi fogalomkörbe tartozó feladatok vizsgálatát egyrészt a valós életből vett gyakorlati problémák bizonyos osztályainak jellemzői tették szükségessé, mint például egyes rendszerek stabilitásának vizsgálata. Ez alatt értendő, hogy egy önszabályzó rendszer esetében egy lokális optimum megfelelő környezetéből a vezérlés a rendszert mindig az adott lokális optimumhely felé kényszerítheti. Ezen feladatok megoldását nagyban könnyíti, hogy az adott célfüggvény deriváltfüggvényének (amennyiben létezik) zérushelyei lokális szélsőértékhelyek, ha nem nyeregpontok. A szimbolikus algebrai számítógépes rendszerek is segítséget nyújthatnak a klasszikus optimalizálási problémák megoldásától egészen a grafikai megjelenítésig (ld. pl. [45]). Ettől eltekintve is számos hatékonynak mondható eljárás létezik a lokális optimalizálási feladatok megoldására. A valóságban némely egyszerű approximációs módszer aránylag biztonságosan megoldja a gyakorlati lokális optimalizálási problémákat. A globális optimum kere-

sése azonban más, komolyabb erőfeszítéseket igényel, és kiszámítása NP-nehéz probléma. Egyszerűbb esetben a feladat megoldható úgy is, hogy az összes lokális optimum pont közül a legjobbat keressük. Ehhez azonban meg kell találnunk minden lokális optimumot. Az eredeti feladat célfüggvényének differenciálhatósága esetén ezzel ekvivalens feladat, hogy egy nemlineáris egyenletrendszer minden gyökét megkeressük. Gyakorlati vizsgálatok is azt mutatják [33], hogy erre a hatékony lokális keresők önmagukban nem nyújtanak megfelelő eszközt. A lokális módszerek globális problémák megoldásához való felhasználásakor azok átgondolt, összehangolt alkalmazása azonban lényegesen jobb eredmények elérését teszi lehetővé.

Világos, hogy hagyományos lokális kereséssel teljes bizonyossággal globális megoldást találni tetszőleges, ismeretlen szerkezetű feladat esetén lehetetlen. Ez azon múlik, hogy bármilyen lokális módszer csupán lokális információt használ — a gradiens módszer például adott pontból a függvény lokális növekedése legmeredekebb irányának és a gradiens abszolút értékének ismeretét hasznosítja. Ezen lokális információk azonban semmit sem mondanak arról, hogy a célfüggvény hogyan viselkedik egy bizonyos környezeten kívül. Annak záloga tehát, hogy egyáltalán reményünk lehessen egy globális optimalizálási probléma megnyugtató megoldására, olyan adatok kiszámítása és felhasználása, melyek a célfüggvényre vonatkozó globális információt is hordoznak valamely tartomány pontjai felett.

A több, mint négy évtizede létező intervallum analízis [60] tette lehetővé, hogy ilyen módszerek létrejöhessenek, és egy adott feladat optimumára elméletileg tetszőleges pontossággal alsó, illetve felső korlátot adjanak. Bár az eljárások egy része szintén évtizedek óta létezik, azok állandó javítása és vizsgálata a módszerek elméleti jelentősége miatt egyre fontosabb feladattá válik. Az intervallum analízis segítségével bizonyos kérdésekre bizonyító erejű válaszok nyerhetők, és a gyakorlati alkalmazások terén is jelentős előrelépések várhatók a repülőgép-tervezéstől a robotikáig.

Ezen dolgozat öt fő részre tagozódik. Az első fejezetben bemutatjuk magát a problémát, valamint rövid áttekintést adunk a globális optimalizálás egyéb területein elért eredményekről, azok előnyös tulajdonságairól és hátrányairól. A második fejezet az intervallum analízis alapjait, az intervallumos műveletek elméletének területét mutatja be. Ennek segítségével vezetjük be az intervallumos módszereket és használatuk gyakorlatát. A harmadik fejezet az intervallum-felosztási globális optimalizálási módszerekkel foglalkozik. Ebben a részben részletesen tárgyaljuk az intervallum-felosztási algoritmusokban rejlő lehetőségeket, összegezzük az eddigi eredményeket, kiegészítjük azokat és további elméleti eredményeket közlünk. Ezek egyrészt ezen módszerek konvergenciájának részletes vizsgálatát tartalmazzák, másrészt a vizsgálódásba új elemeket vonnak be. A negyedik fejezet az intervallum-felosztási eljárások egy új ágát mutatja be, melyen belül egy hatékony mód-

szer felállításának a lehetőségeit járja körül, részletesen leírva annak elméleti és gyakorlati hatásait. A záró fejezet az algoritmusok egy osztályán új lehetőséget mutat be a gyorsított intervallum-felosztási eljárások elméleti vizsgálatához, és annak segítségével több eredményt is közöl. Ezek után következik az összefoglalás magyar és angol nyelven, mely a közölt eredményeket fogja össze, és a további vizsgálódási körökre, nyitott problémákra mutat rá. A dolgozatot a könnyebb áttekinthetőség érdekében az irodalomjegyzéken és tartalomjegyzéken kívül algoritmusok, ábrák és táblázatok jegyzékei, valamint tárgymutató és függelék egészítik ki.

1. Bevezetés

1.1. Az optimalizálási probléma

A globális optimalizálási probléma általános alakját a következőképpen adhatjuk meg:

$$\min_{x \in S} f(x). \tag{1}$$

Az (1) jelöléseiben S-et ($S \subseteq \mathbf{R}^n$ valamely $n \in \mathbf{N}$ értékre) a lehetséges megoldások halmazának nevezzük, a minimalizálandó f: $\mathbf{R}^n \to \mathbf{R}$ függvény pedig a probléma célfüggvénye. A feladat megoldása a globális minimum f^*, a globális minimum helye x^* ($f^*=f(x^*)$), azonban különböző elméleti és gyakorlati meggondolások végett szokás lokális vagy más néven helyi megoldásokról is beszélni, melyeket a következőképpen definiálhatunk:

1. Definíció. Azt mondjuk, hogy $x' \in S$ *lokális minimumhelye* (1)-nek, ha $\exists \varepsilon > 0$ úgy, hogy $\forall x \in S$ értékre, melyre igaz, hogy $\| x - x' \| < \varepsilon$, teljesül az $f(x') \leq f(x)$ összefüggés.

Korlátozási feltételek nélküli feladatról szigorú értelemben akkor beszélünk, ha S nem valódi részhalmaza az n-dimenziós valós térnek, azaz ha $S = \mathbf{R}^n$. Korlátozott optimalizálási probléma esetén a lehetséges megoldások S halmazát többféleképpen is megadhatjuk. Az S egy szokásos megadása a következő alakban történik:

$$S = \{x : g_j(x) \leq 0 \, j = 1, \ldots, m\}, \tag{2}$$

ahol m a korlátozási feltételek száma. Itt a korlátozó függvények, azaz a g_j-k alakjáról még semmit sem feltételezünk, ezért amennyiben egy adott feladat más alakban szerepel, az könnyedén az (2)-nek megfelelő alakra hozható. Így a $g_j(x) \geq 0$ alakú feltétel helyettesíthető a $0 - g_j(x) \leq 0$ feltétellel, valamint a $g_j(x) = 0$ feltétel a $g_j(x) \leq 0$ és $0 - g_j(x) \leq 0$ feltételpárral.

Az (1) és (2) által leírt globális optimalizálási problémák néhány gyakori speciális formája a következőképpen definiálható:

- *lineáris optimalizálási probléma, illetve lineáris programozási feladat:* a feladat felírásában szereplő összes függvény, azaz f és minden g_j lineáris;

- *lineárisan korlátozott optimalizálási probléma:* minden g_j korlátozási függvény lineáris, azaz S egy általánosított poliéder, f tetszőleges függvény;
- *intervallumokkal korlátozott feladat:* olyan speciális, lineárisan korlátozott optimalizálási probléma, melyben az S egy zárt intervallum;
- *kvadratikus programozási feladat:* speciális lineárisan korlátozott optimalizálási probléma, melynek célfüggvénye, f, kvadratikus függvény;
- *konvex optimalizálási probléma:* a felírásban szereplő minden függvény konvex;
- *nemlineáris optimalizálási probléma:* a felírásban szereplő valamely függvény nem lineáris.

A fenti definíciók nem adják osztályozását az optimalizálási problémáknak; a kvadratikus és a konvex optimalizálási problémák esetében például a két problémahalmaznak van közös része, de mindkettőnek van olyan része is, mely a másikhoz nem tartozik. A lineáris és nemlineáris problémák azonban osztályozást adnak meg.

Az optimalizálási problémák nehézségét a bonyolultságelméletből ismert fogalmakkal is leírhatjuk. Azon feladatok osztályát, melyek valamilyen determinisztikus algoritmus segítségével a feladat leírásának függvényében polinomiális idő alatt megoldhatók, *P problémaosztály*nak nevezzük. A feladatok egy másik lényeges bonyolultsági osztályát alkotják a nemdeterminisztikus algoritmussal polinomiális idő alatt megoldható feladatok. Ez utóbbi osztály az *NP problémaosztály*. Más szóval, a P osztály azon feladatokat tartalmazza, melyeket „könnyű" megoldani, az NP osztály pedig azokat, melyekre „könnyű" egy adott értékről eldönteni, hogy az adott feladatnak megoldása-e. A P nyilván részhalmaza az NP problémaosztálynak. Az ellentétes tartalmazást eddig sem bizonyítani, sem cáfolni nem sikerült. Egy feladatot *NP-teljes*nek nevezünk, ha NP-beli, és bármely NP-beli probléma polinomiális idő alatt az adott feladat alakjára hozható. Az NP osztály legnehezebben megoldható feladatainak osztálya ez, mivel ha egy NP-teljes feladatról kiderülne, hogy P-beli, akkor az a definíció értelmében azt jelentené, hogy P=NP. Ennél általánosabb definíciót kapunk, ha nem tesszük fel egy problémáról azt sem, hogy NP-beli: azt mondjuk, hogy egy feladat *NP-nehéz*, ha bármely NP-beli probléma polinomiális idő alatt az adott feladat alakjára hozható.

Ezek után lássuk néhány optimalizálási feladatfajta bonyolultságát. A lineáris programozási feladatokat megoldó algoritmusok között létezik olyan, mely bármely lineáris problémát polinomiális idő alatt megold (például az ellipszoid algoritmus vagy bizonyos más belső pontos módszerek), így a lineáris programozási feladatok osztálya P-beli. Ezzel szemben már a kvadratikus programozási feladatok esetében a probléma NP-nehéz, például ha a célfüggvény nem pozitív szemidefinit kvadratikus függvény, azaz nem konvex. Sőt, indefinit kvadratikus programozási feladat esetében annak eldöntése, hogy adott pont lokális minimume, szintén NP-nehéz probléma.

A globális optimalizálás tehát az esetek túlnyomó többségében NP-nehéz problémák megoldását jelenti. A következőkben röviden bemutatunk néhány eljárást, melyek hatékony megoldást adnak az (1) feladat speciális eseteire.

1.2. A globális optimalizálási módszerek áttekintése

A globális optimalizálás témakörében számos módszer létezik, melyek többsége erősen támaszkodik a célfüggvény valamilyen speciális tulajdonságára. Bizonyos módszerek a függvények széles osztályain felhasználhatók, azonban eredményes működést ezek esetében is általában csak a speciális feltételeknek eleget tevő feladatok esetében várhatunk el. A következőkben röviden, felsorolásszerűen áttekintjük a globális optimalizálási módszerek leggyakrabban előforduló válfajait (ld. például [4, 32, 38, 56]). Eltekintünk a lineáris esettől, amely elég speciális és szűkeskörű irodalommal rendelkező terület. Kihagytuk a felsorolásból az olyan speciális eseteket is, mint a multiplikatív programozás, a tört programozás vagy a trajektória módszerek.

1.2.1. D.C. optimalizálás

Ahhoz, hogy a d.c. (difference of convex functions) optimalizálásról beszélhessünk, először megadjuk a d.c. függvények definícióját:

2. Definíció. Az f: $C \to \mathbf{R}$ ($C \subseteq \mathbf{R}^n$ konvex halmaz) függvényre azt mondjuk, hogy *d.c. függvény*, ha léteznek olyan p és q konvex függvények p,q: $C \to \mathbf{R}$, melyekre $f(x)=p(x)-q(x)$ minden $x \in C$ esetén.

A d.c. optimalizálási módszerekkel megoldható feladatok osztálya ezek után a következő általános alakban írható fel:

$$\min_{x \in C} f(x)$$
$$g_j(x) \leq 0 \quad (j = 1,\ldots,m),$$

ahol C egy zárt, konvex részhalmaza $\mathbf{R}^n$-nek, valamint f és a g_j függvények d.c. függvények. Bizonyítható, hogy minden ilyen alakú feladat ún. kanonikus alakra hozható. A kanonikus d.c. optimalizálási probléma alakja:

$$\min_{x \in D} c^{\mathrm{T}} x$$
$$g(x) \geq 0,$$

ahol D egy zárt, konvex részhalmaza $\mathbf{R}^n$-nek, és g: $\mathbf{R}^n \to \mathbf{R}$ konvex függvény.

Általában elég nehéz probléma adott d.c. függvényhez megadni egy d.c. felbontást, azaz a függvényt konvex függvények különbségeként felírni. Mindazonáltal minden folytonos függvény tetszőlegesen jól közelíthető d.c. függvénnyel, így az alábbiakban felsorolásra kerülő módszerek közelítő megoldást képesek szolgáltatni tetszőleges folytonos függvények esetében is.

Számos eljárás létezik, mely képes speciális esetekben hatékonyan megoldani a kanonikus d.c. optimalizálási problémát. Ezek közül csak néhányat említünk itt.

Élkövető algoritmus:

A módszer működésének feltételei a következők:

- a D halmaz olyan politóp legyen, melynek belseje nem üres,
- a $g(x)\geq 0$ ún. *lényeges feltétel* legyen, azaz a keresett optimális megoldást a célfüggvény a $g(x)=0$ helyen vegye fel,
- a lehetséges megoldások halmaza *robusztus*, azaz belső pontjainak halmaza pozitív mértékű.

Az élkövető algoritmus a fenti feltételek teljesülése esetén a kanonikus d.c. optimalizálási problémát véges lépésben megoldja.

Kúpos korlátozás és szétválasztás algoritmus:

Az algoritmus a következő alakú feladatok megoldására alkalmas:

$$\min_{\substack{x\in X, y\in Y \\ (x,y)\in Z}} c(x,y)$$

$$g(y)\geq 0,$$

ahol c lineáris függvény, X és Y politópok, Z poliéder, g pedig folytonos és kvázikonvex.

Amennyiben az eljárás véges lépésben véget ér, akkor a feladat vagy nem oldható meg, vagy az eredményként kapott megoldás optimális. Ellenkező esetben, ha az algoritmus iterációinak sorozata egymásba ágyazott halmazrendszereket épít fel, azok minden torlódási pontja optimális megoldása a feladatnak [38, 56].

Prizmatikus algoritmus politópokon:

A megoldandó optimalizálási probléma alakja a következő:

$$\min_{x\in D} f(x)-g(x),$$

ahol D egy politóp, f és g pedig konvex függvények. Maga az eljárás lényegében egy korlátozás és szétválasztás elvén működő algoritmus.

Az algoritmus konvergenciájára ugyanaz jellemző, mint a kúpos korlátozás és szétválasztás algoritmusára.

1.2.2. Kvadratikus programozás

A kvadratikus optimalizálási problémák általános alakja a következőképpen írható fel:

$$\min_{x \in D} \tfrac{1}{2} x^{\mathrm{T}} Q x + c^{\mathrm{T}} x ,$$

ahol D poliéder, Q pedig valós, szimmetrikus mátrix. Ezen problémakör jelentős irodalommal rendelkezik, és számos módszer született megoldására, ezért itt inkább ezen problémák megoldásának nehézségére mutatunk rá.

A feladat Q paramétermátrixától nagyban függ annak megoldhatósága, illetve a megoldó algoritmusok bonyolultsága. Három egymástól lényegében különböző esetet szokás a kvadratikus optimalizálási feladatok körében megkülönböztetni, melyek a következők:

- ha Q pozitív szemindefinit — konvex kvadratikus programozás,
- amennyiben Q indefinit — indefinit kvadratikus programozás,
- Q negatív szemidefinit esetén — konkáv kvadratikus minimalizálás.

Pardalos és Vavasis 1991-ben bizonyították [50], hogy az első, azaz a konvex programozás esetét leszámítva a kvadratikus programozási feladat NP-nehéz. A konvex programozás klasszikus esetét leszámítva tehát a kvadratikus optimalizálási problémák nehézsége nem tér el az általános globális optimalizálási problémákétól.

1.2.3. Lipschitz optimalizálás

A Lipschitz optimalizálás [49] szintén speciális alakú feladatok megoldására alkalmas eljárások gyűjtőneve. Az általános Lipschitz optimalizálási probléma a következőképpen írható fel:

$$\min_{x \in D} f(x) ,$$

ahol a $D \subseteq \mathbf{R}^n$ kompakt halmaz, f pedig *Lipschitz-folytonos függvény* D felett, azaz $\forall x_1, x_2 \in D$ értékekre $|f(x_1)-f(x_2)| \le L \|x_1-x_2\|$ teljesül. Ahhoz, hogy ezt a feladatosztályt megoldhassuk, további speciális feltételeket szükséges szabni a D kompakt halmazra. Általában négy ilyen speciális esetet szokás vizsgálni:

- $D=\{x \in \mathbf{R}^n : a \le x \le b\}$, azaz D egy több dimenziós intervallum;
- D egy n-dimenziós szimplex;
- $D=\{x \in \mathbf{R}^n : Ax \le b\}$, azaz egy politóp;
- $D=\{x \in \mathbf{R}^n : g_j(x) \le 0,\ j=1,\ldots,m\}$, ahol $g_j(x)$ Lipschitz-folytonos minden j-re valamely L_j halmaz felett.

A különböző speciális esetekben más-más módszer alkalmazható a feladatok megoldására. A legrégebbi eljárás az ún. *Piyavskii-Shubert módszer*, vagy más néven *fűrészfog algoritmus* [51]. Ez egy iteratív eljárás, mely minden olyan esetben alkalmazható és konvergál is, amikor D egy intervallum vagy más speciális halmaz [49].

Amennyiben D egy szimplex, és létezik egy olyan $T \subset D$ ponthalmaz, melynek pontjaiban az f értékei előre ismertek vagy könnyen kiszámíthatók (előnyös ilyenkor a szimplex csúcsainak halmazát T-nek választani), akkor a fűrészfog algoritmuséhoz hasonló, de annál jobb alsó becslés adható egy lineáris közelítés segítségével. Az egyéb, fentebb felsorolt feladatosztályokra számos, a korlátozás és szétválasztás elvén működő eljárás létezik.

1.2.4. Homotópia módszerek

A homotópia módszerek valójában egyenletrendszereket oldanak meg, speciális esetekben azonban, amikor az optimalizálási feladat célfüggvénye differenciálható, ez lehetőséget ad az optimalizálási feladatok megoldására is. Két dolog miatt is érdemes megemlíteni ezeket a módszereket. Az egyik, hogy az egyenletrendszereket megoldó algoritmusok optimalizálásra való felhasználásának ezen elve elég gyakran kerül felhasználásra a gyakorlati problémák esetében, a másik, hogy a módszer bizonyos változatai korlátos tartomány felett tetszőleges analitikus, folytonosan differenciálható függvény esetén alkalmazhatók.

A módszer lényege, hogy amennyiben a megoldandó egyenletrendszer alakja $f(x)=a$, a $H(x,t)=(1-t)f^0(x)+t(f(x)-a)$ homotópiafüggvény felírását követően a $0 \le t \le 1$ paramétert $t=0$ értéken rögzítjük. Ebben az esetben az $f^0(x)=0$ egyenletrendszer megoldásával adódik a homotópiafüggvény $H(x,0)=0$ megoldása. A t paraméter változtatásával eljutva a $t=1$ esethez a homotópiafüggvény megoldása éppen az eredeti egyenletrendszer megoldását szolgáltatja. Az f^0 függvény megválasztásának, illetve az algoritmus egyéb részleteinek leírásától itt eltekintünk.

A módszer igazán akkor hatékony, ha f polinom.

1.2.5. Sztochasztikus módszerek

A globális optimalizálás hatékony, aránylag robusztus és megbízható, általánosan alkalmazható válfaját alkotják a sztochasztikus módszerek. A feladatosztály, melyre alkalmazhatók, igen tág, a kompakt halmazon folytonos függvények optimalizálási problémáiból áll. Jó alkalmazhatóságuknál fogva számos változatuk létezik. A leggyakrabban használt osztályukat a kétfázisú módszerek alkotják. Jelentős elméleti háttérrel rendelkeznek még a véletlen keresők, valamint a véletlen

függvény módszerek, utóbbi gyakorlati alkalmazhatóságáról, sőt, akár hatékony megvalósításáról azonban a kutatások jelenlegi fázisában még nincs szó.

A *kétfázisú módszerek* nevüket onnan kapták, hogy két, egymástól jól elkülöníthető részből állnak: az első fázist a globális rész alkotja, mely többnyire véletlen pontokban történő függvényértékelésből áll, a második fázist ezen pontokból induló lokális keresés képezi. A kétfázisú módszereknek is számos csoportja létezik. Három fő csoportot szokás külön kezelni, melyek a következők:

- *egyszerű véletlen keresők:* ezek tulajdonképpen csak globális fázisból állnak, elméletileg 1 valószínűséggel konvergálnak, de konvergenciájuk sebessége a változószám növelésével exponenciálisan csökken;
- *multistart eljárások:* a globális fázis után minden új pontból helyi keresőt indítanak, és megvizsgálják a kapott pontokban a függvényértékeket;
- *klaszterező eljárások:* a globális fázisban kijelölt, illetve a lokális fázisban kapott pontokat nem izoláltan kezelik, hanem különféle normák felhasználásával klaszterekbe sorolják őket úgy, hogy egy klaszteren belül lévő pontok bármelyikéből az adott lokális kereső ugyanazt a lokális minimumhelyet találná meg [23].

A sztochasztikus módszerek esetében az 1 valószínűségű konvergencia miatt nagyon lényeges kérdés a helyes megállási feltételek megállapítása. Az egyszerű véletlen keresőknél ez a következő módon fogalmazható meg. Jelölje a lehetséges megoldások (kompakt) halmazát D, az optimumhelyekét D^*, az optimumhelyek ε sugarú környezetét pedig $B_\varepsilon(D^*) := \{x \in D : \|x - x^*\| \le \varepsilon$ valamely $x^* \in D^*$ pontra$\}$. Ebben az esetben annak a valószínűsége, hogy k darab véletlenszerűen választott pontból legalább egy valamelyik globális optimumhely ε sugarú környezetébe essen, $1-(1-\mu(B_\varepsilon(D^*)))^k$, ahol a μ egy valós mérték. Így ha azt akarjuk, hogy ez a valószínűség legalább $1-\delta$ legyen valamely kicsi, pozitív δ értékre, a véletlen módon kiválasztandó pontok k számára a következő alsó korlát adódik:

$$n \ge \frac{\log \delta}{\log(1-\mu(B_\varepsilon(D^*)))}.$$

Ez a korlát elméletileg helyes, azonban figyelmen kívül hagyja a feladat specifikusságát. Egy jó megállási feltételnek a sztochasztikus optimalizálási módszerek esetében függenie kell a véletlen pontok választásától, magától a problémától, a választott lokális módszertől, a várható információvesztéstől és a rendelkezésre álló számítási környezettől. Jól megválasztott megállási feltétel esetén a sztochasztikus módszerek elég megbízhatóan működhetnek.

1.3. Globális optimalizálás — az általános eset

Az előző alfejezet pontjaiban a teljesség igénye nélkül felsorolt példák jellegzetes képét adják a széleskörűen használt, illetve vizsgált globális optimalizálási módszereknek. A feladatokban szereplő függvények alakjával szemben leggyakrabban támasztott követelmények elég szigorúak. A legjobb algoritmusok természetesen a lineáris eset megoldására léteznek, de más speciális esetekben is megbízható eljárások állnak rendelkezésre; így például a kvadratikus, a konvex vagy a d.c. programozási feladatok esetében. Egyik probléma tehát, hogy ha a problémaosztályok széles skáláját kívánjuk megoldani, minden osztály számára más-más módszer alkalmazandó. Másrészt a feladat sajátosságait néha nem is olyan könnyű felismerni, nem beszélve annak kihasználására tett kísérletek esetén felmerülő problémákról (például amilyen d.c. programozás esetén a kanonikus alak felírása). Ezek a gondok azonban általában nem jelentkeznek ennyire élesen, mivel a gyakorlati életben gyakran egy-egy jól körülhatárolható osztály feladatait szükséges más-más paraméterekkel megoldani. Nehézségek akkor merülnek fel leginkább, ha az adott feladatosztályra nem létezik megbízható eljárás.

A másik probléma tehát a megbízhatóság. Mivel az eddig vázolt eljárások többsége csupán lokális információt használ fel, az általános alakú, több lokális minimummal rendelkező célfüggvények esetében már egyszerűnek tűnő problémák megbízható megoldása is komoly nehézségekbe ütközhet. A pontos és megbízható módszerek keresése a 90-es években egyre nagyobb hangsúlyt kapott, és azok az alkalmazások széles területein lassan felválthatják az aránylag gyors, de általános esetben megbízhatatlan eredményeket szolgáltató eljárásokat.

A következő fejezetekben — az intervallum-matematikába történő bevezetést követően — bemutatásra kerül a globális optimalizáló módszerek egy megbízható osztálya, az intervallumos eljárások köre.

2. Intervallum analízis, az intervallum-felosztás elve

Az intervallum analízis használata a hagyományos optimalizálási eljárások két alapvető hiányosságát igyekszik helyrehozni. Egyrészt nem ismeretes olyan hagyományos, valós függvény kiértékelésen alapuló módszer, amely tetszőleges pontossággal körül tudná határolni a globális optimumhelyek halmazát, valamint magát az optimumértéket. Ez azt jelenti, hogy amennyiben egy algoritmus egy feladatot bizonyíthatóan meg is old, a megoldás pontosságára elég nehéz korlátot adni. Másrészt a hagyományos eljárások többsége magán a feladaton kívül valamilyen elég speciális tulajdonságra és információra (például d.c. felbontás vagy Lipschitz konstans) támaszkodik. Intervallum analízis felhasználásával lehetővé válik olyan módszerek létrehozása, melyek képesek csupán a feladatban megadott paraméterek segítségével (a függvényt kiszámító eljárás birtokában) előre rögzített hibahatárokon belül megadni egy tetszőleges globális optimalizálási feladat megoldását. Mi több, adott esetben tetszőleges pontossággal tudják körülhatárolni a globális optimumhelyek halmazát.

Az intervallum analízis atyjaként R.E. Moore-t szokás emlegetni [46], az első intervallumokkal kapcsolatos vizsgálatok kezdete azonban az 50-es évekre nyúlik vissza, és a lengyel M. Warmus [63] és a japán T. Sunaga [60] nevéhez fűződik. Azóta számos monográfia jelent meg, mely az intervallum analízis tárgykörét taglalja, néhány jelentős könyv ezek közül R.E. Moore [47], G. Alefeld és J. Herzberger [1], H. Ratschek és J. Rokne [53] és [55], E.R. Hansen [36] és R.B. Kearfott [40] művei.

Ebben a fejezetben először megadjuk az intervallum analízis alapvető fogalmait és elméletét, melyek a későbbiekben tárgyalásra kerülő eredmények magyarázatához szükségesek. Ezután az intervallum-aritmetika megvalósításának lehetőségeit soroljuk fel, külön kitérve azok tulajdonságaira. Végül megmutatjuk az intervallum-aritmetika felhasználását a globális optimalizálásban.

2.1. Jelölések és fogalmak

A számítógépek folytonos feladatok kiszámításához való felhasználásakor a lebegőpontos számábrázolás kétfajta hibát is eredményezhet. Elsődlegesen a valós számok rögzített hosszúságú kerekített ábrázolásakor, másodsorban az ebből eredő hibák további számítások során való halmozódásakor. Az intervallum analízis

elve mindkét hibát kiküszöböli oly módon, hogy a valós szám közelítő ábrázolása helyett az értéket egy alsó és felső korlátból álló számpárral jellemzi, melyek egy tetszőleges valós számmal szemben már lehetnek az adott lebegőpontos aritmetika segítségével pontosan ábrázolható értékek. Így az adott számpár által rögzített intervallum középpontja a pontos értéktől soha sem tér el jobban, mint az intervallum szélességének a fele.

Megadjuk az intervallumok pontos definícióját, ahogyan azt a dolgozatban használni fogjuk:

3. Definíció. Az $X=[a,b]=\{x\in\mathbf{R}^n: a\le x\le b\}$ mennyiséget *valós, kompakt intervallum*nak nevezzük, ahol $a\in\mathbf{R}^n$ és $b\in\mathbf{R}^n$ az *intervallum alsó*, illetve *felső korlátja*.

A továbbiakban az egydimenziós valós kompakt intervallumok halmazát **I**-vel, az n-dimenziósokét pedig $\mathbf{I}^n$-nel jelöljük. Ha $D\subseteq\mathbf{R}^n$ egy tetszőleges halmaz, akkor $\mathbf{I}(D)$-vel jelöljük D intervallumait, azaz $\mathbf{I}(D)=\{X\in\mathbf{I}^n: X\subseteq D\}$. Más szóval $\mathbf{I}=\mathbf{I}(\mathbf{R})$, illetve $\mathbf{I}^n=\mathbf{I}(\mathbf{R}^n)$. Az intervallumokat nagybetűs szedéssel különböztetjük meg a valós értékektől. Az $X\in\mathbf{I}^n$ intervallum alsó és felső korlátját az $\mathrm{lb}: \mathbf{I}^n\to\mathbf{R}^n$, illetve az $\mathrm{ub}: \mathbf{I}^n\to\mathbf{R}^n$ függvények segítségével adjuk meg, így tehát

$$X=\{x\in\mathbf{R}^n : \mathrm{lb}(X)\le x\le \mathrm{ub}(X)\}.$$

Egy *intervallum szélességét* a $\mathrm{w}: \mathbf{I}^n\to\mathbf{R}$ függvénnyel definiáljuk, ezen definíció szerint $\mathrm{w}(X)=\mathrm{ub}(X)-\mathrm{lb}(X)$, $X\in\mathbf{I}^n$. A zéró szélességű, egyetlen pontból álló intervallumokat *pontintervallum*oknak nevezzük. A későbbiekben használatos további jelölés az *intervallum középpontját* szolgáltató $\mathrm{m}: \mathbf{I}^n\to\mathbf{R}^n$ függvény, melyre $\mathrm{m}(X)=(\mathrm{lb}(X)+\mathrm{ub}(X))/2$, $X\in\mathbf{I}^n$.

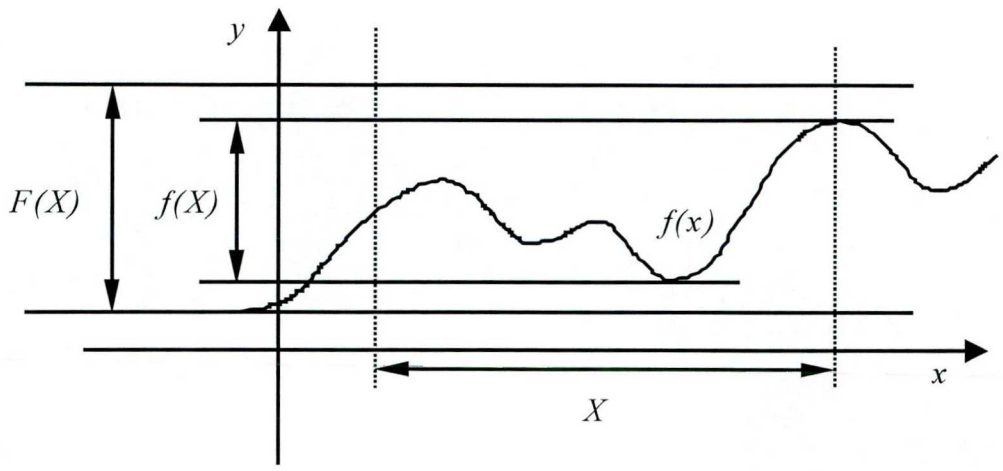
Ugyanakkor a valós függvényeknek megfeleltetünk bizonyos intervallum értelmezési tartományú és értékkészletű függvényeket, melyek eleget tesznek a következő elvnek:

4. Definíció. Legyen $f: D\to\mathbf{R}$ valós függvény, ahol $D\subseteq\mathbf{R}^n$. Az $F: \mathbf{I}(D)\to\mathbf{I}$ intervallum értékű függvényt az f valós függvény egy *befoglalófüggvényének* nevezzük, amennyiben teljesül rá a következő összefüggés:

$$\forall X\in\mathbf{I}(D)\quad f(X)\subseteq F(X), \tag{3}$$

ahol $f(X)=\{f(x) : x\in X\}$ az f függvény X feletti értékkészletét jelöli.

A (3)-ban megfogalmazott összefüggést az *intervallum-aritmetika befoglalási elvé*nek nevezzük, és szemléletesen a 1. Ábrán látható egydimenziós esetben.



1. Ábra. A befoglalás elve: $F(X)$ tartalmazza $f(X)$-et, az f értékkészletét az X intervallum felett.

A négy aritmetikai alapművelet intervallumokra való kiterjesztésének definíciója az $X, Y \in \mathbf{I}$ jelölést használva a következőképpen adható meg [46, 47]:

$$\begin{aligned} X+Y &= [\mathrm{lb}(X)+\mathrm{lb}(Y), \mathrm{ub}(X)+\mathrm{ub}(Y)], \\ X-Y &= [\mathrm{lb}(X)-\mathrm{ub}(Y), \mathrm{ub}(X)-\mathrm{lb}(Y)], \\ X\cdot Y &= [\min(\mathrm{lb}(X)\mathrm{lb}(Y), \mathrm{lb}(X)\mathrm{ub}(Y), \mathrm{ub}(X)\mathrm{lb}(Y), \mathrm{ub}(X)\mathrm{ub}(Y)), \\ &\quad \max(\mathrm{lb}(X)\mathrm{lb}(Y), \mathrm{lb}(X)\mathrm{ub}(Y), \mathrm{ub}(X)\mathrm{lb}(Y), \mathrm{ub}(X)\mathrm{ub}(Y))], \\ X/Y &= X\cdot[1/\mathrm{ub}(Y), 1/\mathrm{lb}(Y)], \quad \text{feltéve, hogy } 0 \notin Y. \end{aligned} \tag{4}$$

A fenti definíciók bizonyos értelemben a legjobb befoglalófüggvényeket szolgáltatják a négy aritmetikai alapműveletre mint kétváltozós függvényre, azaz a befoglalófüggvény értéke egyenlő a műveletek, azaz az érintett függvények értékkészletével.

Az alapműveletek ezen definíciója matematikailag azonban néhány szokásos algebrai tulajdonsággal nem rendelkezik [53]. Így a kivonás és az osztás nem inverzei az összeadásnak, illetve a szorzásnak, például $[0,1]-[0,1]=[-1,1]\neq[0,0]$, illetve $[1,2]/[1,2]=[1/2,2]\neq[1,1]$. Igaz azonban, hogy $\forall X \in \mathbf{I}$ $X-X \supseteq [0,0]$ és $X/X \supseteq [1,1]$. Hasonlóképpen a szorzás nem disztributív az összeadásra nézve, érvényes azonban a *szubdisztributivitás*, azaz $\forall X, Y, Z \in \mathbf{I}$-re $X(Y+Z) \subseteq XY+XZ$. Az összeadás és a szorzás kommutatívak és asszociatívak. Teljesül továbbá egy fontos összefüggés, melynek definíciója:

5. Definíció. Az F: $\mathbf{I}(D) \to \mathbf{I}$ ($D \subseteq \mathbf{R}^n$) intervallumfüggvény *izoton* (avagy a *befoglalásra nézve monoton*), ha

$$\forall X,Y \in \mathbf{I}(D), \quad X \subseteq Y \quad \Rightarrow \quad F(X) \subseteq F(Y). \tag{5}$$

A négy alapművelet (4)-ben megadott definíciója izoton függvényeket definiál.

A befoglalófüggvények megvalósításához több módszer is rendelkezésre áll, melyek különböző pontossággal és eltérő információ-, illetve műveletigénnyel rendelkeznek. Egy befoglalófüggvény pontosságát, használhatóságát több módon is jellemezhetjük. A következő definíció [47] segítségével ezen tulajdonságot mérjük.

6. Definíció. Azt mondjuk, hogy az $f : D \to \mathbf{R}$ $(D \subseteq \mathbf{R}^n)$ valós függvény $F : \mathbf{I}(D) \to \mathbf{I}$ befoglalófüggvénye α-*konvergens*, ha $\exists c \geq 0$ valós konstans és $\alpha > 0$, hogy

$$\forall X \in \mathbf{I}(D) \qquad \mathrm{w}(F(X)) - \mathrm{w}(f(X)) \leq c\,\mathrm{w}^{\alpha}(X). \tag{6}$$

Az intervallum-aritmetikai megvalósítás lehetőségeit adjuk meg a következő alfejezetekben. A konkrét megvalósítások esetében nem tárgyaljuk külön a gyakorlati szempontból elengedhetetlen speciális kerekítések témakörét. Az intervallum analízis önmagában nem veszi figyelembe az adott számítógépes környezetet. Ez azt eredményezhetné, hogy egyes intervallumok alsó vagy felső korlátai az adott számítógépen nem lennének pontosan ábrázolhatók. A kellemetlen következmények elkerülése végett minden egyes intervallumos művelet elvégzése után egy ún. *kifelé kerekítés* szükséges, azaz minden $[a,b]$ $(a,b \in \mathbf{R}^n)$ intervallum helyett az $[a_{\mathrm{M}}, b_{\mathrm{M}}]$ intervallumot tekintjük, ahol $a_{\mathrm{M}} \leq a$ és $b_{\mathrm{M}} \geq b$ általában az eredeti korlátokhoz legközelebb eső racionális értékek, melyek az adott számítógépes környezetben pontosan ábrázolhatók. Ha tehát egy tetszőlegesen adott X intervallum kifelé kerekített változatát X_{M}-mel jelöljük, akkor bármely F megfelelő intervallumfüggvény esetében az $F(X)$ érték helyett valójában az $F_{\mathrm{M}}(X_{\mathrm{M}})$ számítandó.

2.2. Befoglalófüggvények

2.2.1. A természetes kiterjesztés

A négy alapművelet (4)-ben megadott definíciója mellé tetszőleges elemi valós függvényre megadható olyan befoglalófüggvény, amely pontos befoglalást ad, azaz bármely argumentum esetén a befoglalófüggvény értéke egyenlő a valós függvénynek az adott intervallum feletti értékkészletének *intervallumburká*val (itt intervallumburok alatt az a legszűkebb intervallum értendő, amely tartalmazza az értékkészlet halmazát). Az elemi függvények mellett ez bármely, logikai műveletekkel

megadott függvényre is igaz, mint például az előjelfüggvény. Végtelen értékek előfordulása esetén szükségessé válik az intervallumok fogalmának kiterjesztése (ld. pl. [57]), mellyel itt külön nem foglalkozunk.

Feltesszük, hogy minden olyan elemi függvény befoglalófüggvényét megadtuk a fenti módon, melyekből egy adott f: $D \to \mathbf{R}$ ($D \subseteq \mathbf{R}^n$) valós függvény felépíthető. Ebben az esetben az f egy befoglalófüggvénye megadható rekurzív módon úgy, hogy a valós függvényt alkotó elemi függvények kiszámítási sorrendjében értékeljük ki az argumentumként megadott, illetve a számítások során adódó intervallumokon a megfelelő elemi függvények befoglalófüggvényeit. Ezt a módszert, illetve az ily módon kapott befoglalófüggvényt nevezzük az f függvény *természetes befoglalófüggvény*ének vagy *természetes intervallum-kiterjesztés*ének. Szokásos ezt a fajta kiterjesztést *naiv intervallum-kiterjesztés*nek is nevezni.

A valós függvények és természetes befoglalófüggvényeik között nem kölcsönösen egyértelmű a megfeleltetés. Ugyanazt a valós függvényt általában többféleképpen is felépíthetjük elemi függvények és az alapműveletek segítségével. A valós függvények különböző alakú természetes kiterjesztései általában eltérnek. Egy egyszerű példa erre a szubdisztributivitás egyenes következménye abban az esetben, amikor nem teljesül az egyenlőség. Tekintsük ehhez az $f(x)=x(1-x)=x-x^2$ függvényt. Ennek természetes kiterjesztése mind az $F_1(X)=X(1-X)$, mind pedig az $F_2(X)=X-X^2$ befoglalófüggvény, mégis, például az $X=[0{,}5,\ 1]$ érték esetében $F_1(X)=[0,\ 0{,}5]$, míg $F_2(X)=[-0{,}5,\ 0{,}75]$. Az értékkészletet, azaz az $f(X)=[0,\ 0{,}25]$ értéket mindkettő tartalmazza ugyan, fennáll azonban a szigorú $f(X) \subset F_1(X) \subset F_2(X)$ tartalmazás.

Bár az elemi függvények befoglalófüggvényei definíció szerint pontosak — abban az értelemben, hogy megegyeznek a megfelelő valós függvény argumentum feletti értékkészletének intervallumburkával —, az azokból felépített természetes kiterjesztések közül általában már egyik sem az. Ez azt eredményezi, hogy tetszőleges intervallum felett teljesül ugyan a (3) befoglalási elv, a természetes befoglalófüggvény által szolgáltatott intervallum azonban lényegesen bővebb, mint az értékkészlet intervallumburka. Igaz azonban a következő állítás [53].

1. Tétel (Ratschek-Rokne). Bármely f: $D \to \mathbf{R}$ ($D \subseteq \mathbf{R}^n$) valós függvény egy F természetes kiterjesztése 1-konvergens, azaz

$$\forall X \in I(D) \quad \mathrm{w}(F(X)) - \mathrm{w}(f(X)) = \mathcal{O}(\mathrm{w}(X)) .$$

A függvények széles osztályaira olyan befoglalófüggvényeket, melyek kiszámítása a valós függvény alakjának ismeretében automatikusan elvégezhető, legegyszerűbb a természetes befoglalással generálni. Ennél jobb eredmények, bizonyos esetekben 2-konvergens, azaz kvadratikusan konvergens befoglalófüggvények konstruálhatók azonban a középponti formulák segítségével.

2.2.2. A középponti formulák

A középponti formulákkal befoglalófüggvények széles osztályai definiálhatók. A középponti formulák vagy középponti függvények (central forms) általános definíciója a következő [53]:

7. Definíció. Legyen f: $D \to \mathbf{R}$ ($D \subseteq \mathbf{R}^n$) valós függvény, $X \in \mathbf{I}(D)$ intervallum, $D_0 \subseteq D$ halmaz, β: $\mathbf{I}(X) \to D_0$ valós vektor értékű függvény. Az s: $X \times D_0 \to \mathbf{R}$ függvényt definiáljuk a következőképpen: $s(x,c)=f(x)-f(c)$ minden $x \in X$ és $c \in D_0$ értékre. Ha létezik egy olyan pozitív egész r, továbbá S: $\mathbf{I}(X) \to \mathbf{I}$, valamint G^p: $\mathbf{I}(X) \to \mathbf{I}^m$ ($p=1,\ldots,r$) intervallum függvények úgy, hogy

$$s(x,\beta(Y)) \in S(Y) \subseteq \sum_{p=1}^{r} (Y-\beta(Y)) \cdot G^p(Y),$$

minden $Y \in \mathbf{I}(X)$, $x \in Y$ esetén, akkor az

$$F(Y) = f(\beta(Y)) + S(Y)$$

intervallum függvényt az *f középponti függvényének* nevezzük *X felett*, ahol β a *kifejtési pontot kijelölő függvény*.

A középponti függvények konvergenciájáról általában a következő mondható [47].

2. Tétel (Moore). Legyen F a 7. Definíciónak megfelelő középponti függvény. Ha az ott megadott G^p ($p=1,\ldots,r$) függvények korlátosak és a β kifejtési pontot kijelölő függvényre igaz, hogy minden $Y \in \mathbf{I}(X)$-re $\beta(Y) \in Y$, akkor F 1-konvergens befoglalófüggvénye f-nek.

Speciális függvények esetében jobb befoglalófüggvények is adhatók. Ilyen eset, amikor az f valós függvény Lipschitz folytonos [42].

3. Tétel (Krawczyk-Nickel). Ha a 7. Definícióban szereplő $r=1$ és G^1 Lipschitz folytonos, akkor az F középponti függvény 2-konvergens, amennyiben a β kifejtési pontot kijelölő függvényre igaz, hogy minden $Y \in \mathbf{I}(X)$-re $\beta(Y) \in Y$.

A középponti függvények gyakorlatban széles körben alkalmazott változatai a Taylor formulák vagy Taylor függvények. Ezek szoros kapcsolatban állnak a valós analízisből ismert formulákkal, pontos definíciójuk a következőképpen adható meg [53]:

8. Definíció. Az n-dimenziós valós f függvény k-adrendű Taylor függvénye ($k \geq 1$) X felett

$$T_k(Y,c) = f(c) + \sum_{|\lambda|=1}^{k-1} \frac{D^\lambda f(c)}{\lambda!}(Y-c)^\lambda + \sum_{|\lambda|=k} \frac{F^{(\lambda)}(Y)}{\lambda!}(Y-c)^\lambda ,$$

ahol $Y \in \mathbf{I}(X)$, $F^{(\lambda)}$ az f λ-dik deriváltfüggvényének egy befoglalófüggvénye, továbbá a következő rövidítéseket és jelöléseket használjuk:

$$c = \mathrm{m}(Y),$$
$$\lambda = (\lambda_1,\ldots,\lambda_n), \lambda_i \in \mathbf{N}(i = 1,\ldots,n),$$
$$|\lambda| = \sum_{i=1}^{n} \lambda_i,$$
$$\lambda! = \prod_{i=1}^{n} \lambda_i!,$$
$$D^\lambda f(x) = \frac{\partial^{\lambda_1+\ldots+\lambda_n} f(x)}{\partial x_1^{\lambda_1} \cdot \ldots \cdot \partial x_n^{\lambda_n}},$$
$$D^\lambda f(x) \in F^{(\lambda)}(Y) \in \mathbf{I} \forall Y \in \mathbf{I}(X), x \in Y.$$

A Taylor függvények valóban befoglalófüggvények, valamint a 7. Definíció értelmében középponti függvények is. Speciális alakjuknál fogva azonban a 2. és 3. Tételeknél erősebb állítás is igaz rájuk [53].

4. Tétel (Ratschek-Rokne). Ha $k>1$ és a parciális deriváltfüggvények $F^{(\lambda)}$ befoglalófüggvényei korlátosak minden $|\lambda| = k$ értékre, akkor a k-adrendű Taylor függvény 2-konvergens.

A Taylor függvények definíciójában c megadása eltérhet a függvény argumentumában megadott intervallum középpontjától. A kiszámítás ötlete E. Baumanntól származik [2], és segítségével javítható a Taylor függvények által adható befoglalások szélessége. A pontos definíció a következő.

9. Definíció. Az alábbi két értéket az $Y \in \mathbf{I}(X)$ *intervallum Baumann közepei*nek nevezzük:

$$c_i^- = \begin{cases} \mathrm{ub}(Y)_i & \mathrm{ub}(F_i'(Y)) \le 0, \\ \mathrm{lb}(Y)_i & \mathrm{lb}(F_i'(Y)) \ge 0, \\ \dfrac{\mathrm{ub}(F_i'(Y))\mathrm{lb}(Y)_i - \mathrm{lb}(F_i'(Y))\mathrm{ub}(Y)_i}{\mathrm{ub}(F_i'(Y)) - \mathrm{lb}(F_i'(Y))} & \text{különben.} \end{cases}$$

$$c_i^+ = \begin{cases} \mathrm{lb}(Y)_i & \mathrm{ub}(F_i'(Y)) \le 0, \\ \mathrm{ub}(Y)_i & \mathrm{lb}(F_i'(Y)) \ge 0, \\ \dfrac{\mathrm{lb}(F_i'(Y))\mathrm{lb}(Y)_i - \mathrm{ub}(F_i'(Y))\mathrm{ub}(Y)_i}{\mathrm{lb}(F_i'(Y)) - \mathrm{ub}(F_i'(Y))} & \text{különben.} \end{cases}$$

ahol az alsó index a komponenst jelöli.

A Baumann közepek segítségével jobb Taylor függvények adhatók, mely tulajdonságot a következő két egyenlőtlenség írja le [55]:

$$\mathrm{lb}(T_1(Y,\beta(Y))) \le \mathrm{lb}(T_1(Y,c^-)),$$
$$\mathrm{ub}(T_1(Y,c^+)) \le \mathrm{ub}(T_1(Y,\beta(Y))),$$

ahol β tetszőleges kifejtési pontot kijelölő függvény, melyre $\beta(Y) \in Y$ teljesül. A futtatási eredmények azt mutatják [55], hogy a Baumann közepek használatával a befoglalófüggvény hívások 10-20 százaléka takarítható meg optimalizálási feladatok megoldása esetén.

További feladat a deriváltfüggvények és azok befoglalófüggvényeinek megadása. Ezen értékek kiszámítása történhet numerikus módszerekkel, szimbolikus eljárásokkal és egyéb hatékony módon, melyre a következő két alfejezetben mutatunk be néhány kiszámítási módszert.

2.3. Differenciálhányados függvények befoglalófüggvényei

2.3.1. Az automatikus differenciálás

A Taylor függvények számításánál szükség van a deriváltfüggvény, a Hesse mátrix, illetve elméletileg akár magasabb rendű differenciálhányados függvények befoglalófüggvényeinek megadására, melyre egyszerű módszert nyújt az *automatikus differenciálás*.

Az automatikus differenciálás fogalma alapvetően két módszert takar. Az egyik az ún. *előre tartó módszer* (angolul forward) [52], amelyet az alábbiakban röviden ismertetünk, és az egyszerűség kedvéért automatikus differenciálásnak nevezünk. A másik a *hátra tartó módszer* (angolul backward) [30], amelyet más néven gyors automatikus differenciálásnak is neveznek. Ez az eljárás valóban hatékonyabb az előre tartó módszernél, megvalósítása azonban nagyobb tárigényű és komolyabb programozási hátteret követel.

Az (előre tartó) automatikus differenciálás valós f: $\mathbf{R}^n \to \mathbf{R}$ függvények differenciálhányados függvényeinek az értékét számolja ki a 2.2.1. alfejezetben leírt természetes intervallum-kiterjesztés rekurzív elvéhez hasonlóan. Az ezeket az összefüggéseket leíró képleteket külön-külön szükséges megadni a különböző rendű differenciálhányadosok kiszámolásához. Ebben az értelemben az automatikus dif-

ferenciáláskor megkülönböztetünk gradiens, Hesse, vagy magasabb rendű aritmetikát.

Az intervallumaritmetikában való felhasználhatóság érdekében a valós pontokat mindenütt intervallumos befoglalásokkal helyettesítjük. Ez nem változtat a módszer működésén, mivel minden befoglalófüggvény felfogható két valós értékű függvényként az intervallumok komponensei mint valós vektorok felett.

Ennek fényében a gradiensaritmetika a következő alakú párokkal operál:

$$\Gamma=(G,G'), \qquad G\in\mathbf{I},\ G'\in\mathbf{I}^n.$$

A $C(X)=C$ ($X\in\mathbf{I}^n$, $C\in\mathbf{I}$) konstansfüggvénynek a $\Theta(X)=(C,0)$, míg a $P_i(X)=X_i$ ($X_i\in\mathbf{I}$) projekciónak a $\Pi_i(X)=(X_i,E^i)$ párt feleltetjük meg, ahol E^i az i-dik n-egységvektor pontintervallumát jelöli. A négy alapművelet definíciója az $X=(X,X')$ és $Y=(Y,Y')$ párokra a következőképpen adható meg.

$$\begin{aligned}
\mathrm{X}+\mathrm{Y} &= (X+Y,X'+Y'),\\
\mathrm{X}-\mathrm{Y} &= (X-Y,X'-Y'),\\
\mathrm{X}\cdot\mathrm{Y} &= (X\cdot Y,Y\cdot X'+X\cdot Y'),\\
\mathrm{X}/\mathrm{Y} &= \left(X/Y,\frac{Y\cdot X'-X\cdot Y'}{Y^2}\right), \quad \text{feltéve, hogy } 0\notin Y.
\end{aligned}$$

Az f: $\mathbf{R}\to\mathbf{R}$ valós elemi függvény F: $\mathbf{I}\to\mathbf{I}$ befoglalófüggvényére az $X=(X,X')$ feletti értéket az alábbi módon definiáljuk.

$$\Phi(\mathrm{X})=(F(X),F'(X)\cdot X'),$$

ahol F' az f deriváltfüggvényének egy befoglalófüggvénye.

Ehhez hasonlóan megadhatóak az összefüggések a Hesse-aritmetikára vonatkozóan is. Ekkor a számítást intervallumokból álló hármasokon végezzük, amelyek a következő alakúak.

$$\Omega=(H,H',H''), \qquad H\in\mathbf{I},\ H'\in\mathbf{I}^n,\ H''\in\mathbf{I}^{n\times n}.$$

Két ilyen hármas között a négy alapművelet úgy végzendő, hogy az eredmény első két komponense a gradiensaritmetikát használva számítódik, a harmadik komponensek kiszámítási szabályait pedig a következő összefüggések szolgáltatják.

$$\begin{aligned}
\mathrm{Z}=\mathrm{X}+\mathrm{Y} &\Rightarrow Z''=X''+Y'',\\
\mathrm{Z}=\mathrm{X}-\mathrm{Y} &\Rightarrow Z''=X''-Y'',\\
\mathrm{Z}=\mathrm{X}\cdot\mathrm{Y} &\Rightarrow Z''=X\cdot Y''+X'\cdot Y'+Y'\cdot X'+Y\cdot X'',\\
\mathrm{Z}=\mathrm{X}/\mathrm{Y} &\Rightarrow Z''=\frac{X''-Z'\cdot Y'-Y'\cdot Z'-Z\cdot Y''}{Y}, \quad \text{feltéve, hogy } 0\notin Y,
\end{aligned}$$

ahol értelemszerűen $\mathrm{X}=(X,X',X'')$, $\mathrm{Y}=(Y,Y',Y'')$ és $\mathrm{Z}=(Z,Z',Z'')$. Legyen most f: $\mathbf{R}\to\mathbf{R}$ tetszőleges valós elemi függvény F: $\mathbf{I}\to\mathbf{I}$ befoglalófüggvénnyel, valamint jelölje az első és második deriváltfüggvények befoglalófüggvényeit F', illetve

F''. Ekkor az adott elemi függvényt egy tetszőleges $X=(X,X',X'')$ hármasra a következőképpen értelmezzük:

$$\Phi(X) = (F(X), F'(X)\cdot X', F'(X)\cdot X'' + F''(X)\cdot X\cdot X).$$

Az automatikus differenciálás széles körben nem használatos, mivel külön aritmetikát igényel, akárcsak az intervallumos módszerek. Az operator overloadinggal felszerelt objektumorientált programozási nyelvek azonban könnyedén lehetővé teszik a megvalósítást az automatikus differenciálás esetében éppúgy, mint az intervallum-aritmetika esetében.

2.3.2. Lejtő függvények

A Taylor formulához hasonló középponti függvények kiszámításához az automatikus differenciálásnál jobb módszer is adható, melyet az ún. *lejtő függvények* [41] használata biztosít. Az alábbi rövid ismertetés az egydimenziós esetet mutatja be, több dimenzióra az ötlet ugyanúgy kidolgozható (pl. [36]). Legyen f valós függvény. Az f lejtő függvénye alatt a következő függvényt értjük:

$$g(x,y) = \frac{f(y)-f(x)}{y-x}.$$

Az egyszerűbb eset az, amikor f racionális függvény. Ekkor $f(y)-f(x)$ analitikus úton eloszthatő y-x-szel, így explicit módon kiszámítva a lejtő függvény értékét. Amennyiben $X\in\mathbf{I}$ tetszőleges, és $x,y\in X$, akkor nyilván

$$f(y)\in f(x)+g(x,X)(y-x)$$

teljesül minden $y\in X$-re, és a Taylor formulákhoz hasonlóan megad egy elsőfokú közelítést az f függvényre X felett. Mivel a definícióból adódóan $g(x,y)\in f'(X)$ mindig teljesül, így a lejtő függvények segítségével általában jobb elsőfokú közelítés adható, mint a megfelelő Taylor függvényekkel.

Könnyen igazolható, hogy a négy alapműveletre a lejtő függvények a következőképpen adhatók meg:

$f(x)$	$g(x,y)$
konstans	0
x	1
$f_1(x)\pm f_2(x)$	$g_1(x,y)\pm g_2(x,y)$
$f_1(x)f_2(x)$	$f_1(x)g_2(x,y)\pm g_1(x,y)f_2(y)$
$\dfrac{f_1(x)}{f_2(x)}$	$\dfrac{g_1(x,y)f_2(x)-f_1(x)g_2(x,y)}{f_2(x)f_2(y)}$

A táblázatban foglalt összefüggések segítségével felírható bármely racionális függvény. Más befoglalófüggvényekhez hasonlóan a lejtő függvények segítségével megadott befoglalás is függ a függvény alakjától, azaz a valós függvény azonos átalakításokkal általában olyan alakra hozható, melynél a vele kifejezett befoglalófüggvény pontosabb, esetleg tökéletes korlátokat szolgáltat az adott függvény értékkészletére vonatkozóan.

Érdemes megjegyezni, hogy komplex függvények befoglalása esetén a lejtő függvényekkel ellentétben a Taylor formula egyáltalán nem is használható, mivel nem igaz a középértéktétel.

Irracionális függvények befoglalása esetén a módszer annyiban módosul, hogy az adott függvény analitikus felírásában szereplő irracionális elemi függvényekhez azok deriváltfüggvényét vesszük igénybe a rekurzív kiszámítás során. Az így keletkező vegyes alak a benne szereplő lejtő függvények miatt szintén pontosabb eredményeket szolgáltat általában, mint akár deriváltfüggvényeket, vagy akár automatikus differenciálást alkalmazva.

2.4. Az intervallum-felosztás elve

A befoglalófüggvények segítségével a valós függvények széles osztályain megadhatók az értelmezési tartomány speciális részhalmazai, nevezetesen részintervallumai felett alsó és felső korlátok az értékkészlet halmazára. Olyan befoglalófüggvény létezése, amely véges lépésben tetszőleges függvényre pontos befoglalást ad, nem ismeretes, és az optimalizálási probléma NP-nehéz voltánál fogva nem is valószínű, hogy létezik. Egyes befoglalófüggvények azonban lehetőséget adnak iteratív eljárások felépítésére, melyek segítségével az optimum helye, illetve értéke tetszőleges pontossággal megközelíthető.

Ezen módszerek közé tartoznak az *intervallumos Newton módszer*ek, melyek a szakirodalomban először Moore-nál kerülnek elő [47], majd később számos finomításon mentek keresztül. Mai formájukban jól használhatók az intervallumos globális optimalizáló eljárásokban, feltételezik azonban, hogy az érintett intervallum kicsi, és a célfüggvény kétszer folytonosan differenciálható, és ki is használják az első és második deriváltfüggvények ismeretét. Elvük a valós Newton módszerek elvén alapszik. Az intervallumos Newton módszereket leginkább a későbbiekben tárgyalásra kerülő intervallum-felosztási algoritmusok keretein belül, azok gyorsítására használják. A gyakorlatban ugyan igen hatékonyak, hatásuk azonban elméleti eszközökkel nehezen kezelhető. A tesztek során is jó, de elég eltérő eredményeket szolgáltatnak. Mivel ezen dolgozat igyekszik minél általánosabb esetekben vizsgálni a globális optimalizálási feladatokat, valamint azokra elméleti eredményeket adni, így ezen eljárások részletesebb vizsgálatát a jelen anyagból kihagytuk.

Az intervallum-felosztási módszerek működése azon alapszik, hogy a jelenleg rendelkezésre álló, automatikusan kiszámítható befoglalófüggvények szinte mindegyike α-konvergens valamilyen $\alpha>0$ értékre. Ez azt jelenti, hogy csökkentve a befoglalófüggvény argumentumában szereplő intervallum szélességét, az egyre jobban közelíti a megfelelő valós függvény értékkészletét, azaz

$$w(F(Y)) - w(f(Y)) \to 0, \quad \text{ha } w(Y) \to 0. \tag{7}$$

Folytonos célfüggvények esetében ez azt is jelenti egyben, hogy a befoglalófüggvény szélessége zéróhoz tart, tehát

$$w(F(Y)) \to 0, \quad \text{ha } w(Y) \to 0.$$

Az α–konvergencia ily módon lehetővé teszi a korlátozás és szétválasztás elvének felhasználását. Ebből az elvből a következőekben ismertetésre kerülő általános eljárás építhető fel az (1) feladat megoldására az $S \in \mathbf{I}^n$ esetben. Az eljárás során kezdetben csupán egyetlen intervallumot tekintünk, amely maga az S tartomány.

1. Algoritmus. A korlátozás és szétválasztás elvének hasznosítása a globális minimalizálási feladatok intervallum-aritmetika segítségével való megoldására.

1. Bontsuk fel a vizsgálatban résztvevő intervallumok valamelyikét kisebb részintervallumokra!
2. Töröljük az újonnan keletkezett intervallumok közül a további vizsgálódásokból azokat, melyeken a célfüggvény befoglalófüggvény értékének alsó korlátja nagyobb egy már rendelkezésre álló, a minimum értékére adott felső korlátnál! Folytassuk az eljárást az 1. pontban!

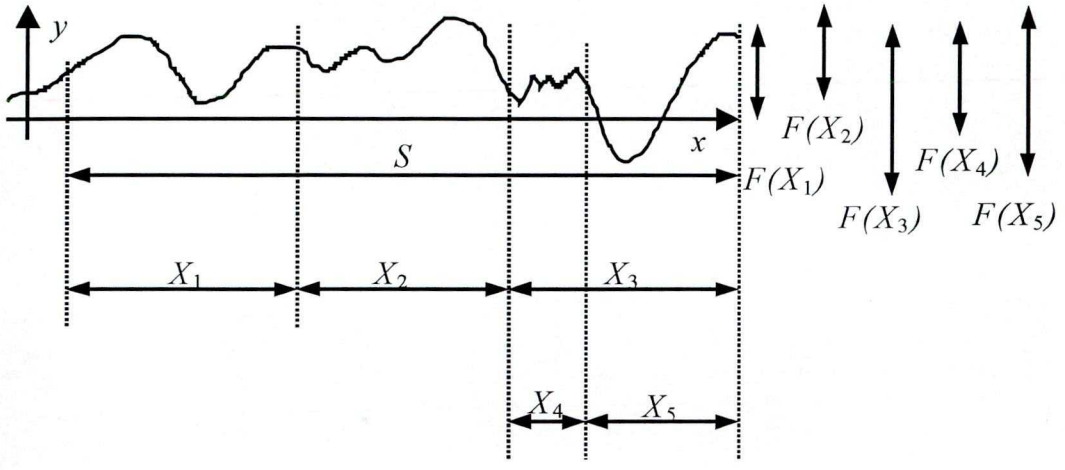
A fenti eljárásvázlatban a szétválasztást az intervallumok továbbdarabolása adja, míg a korlátozásra a befoglalófüggvények részintervallumok felett számított értékei adnak lehetőséget. Az algoritmus működését a 2. Ábra szemlélteti.

Az eljárással kapcsolatban két megjegyzést kell tennünk.

1. Megjegyzés. Az 1. Algoritmus a jelen formában soha nem áll le, miközben az el nem távolított részintervallumok össztérfogata monoton csökken. Az aktuális össztérfogatok sorozatának alsó korlátja a globális minimumhelyek intervallumburkai térfogatának összege (ami nem feltétlenül nulla).

Hasonló tulajdonság figyelhető meg a részintervallumokon számított célfüggvény alsó korlátainak esetében is, amennyiben a befoglalófüggvény olyan, hogy teljesíti a (7) feltételt (az ún. *zéró-konvergenciát*), ahol minden Y az S részintervalluma. Ez a következőképpen fogalmazható meg.

2. Megjegyzés. Ha feltesszük, hogy a befoglalófüggvény eleget tesz (7)-nek, akkor izoton befoglalófüggvény esetén az 1. Algoritmus végrehajtása során a megmaradt intervallumokon számított alsó korlátok minimuma monoton növő sorozatot alkot. A globális minimum ezen sorozatnak felső korlátja. Nem izoton befoglalás esetén ilyen részsorozat mindig kiválasztható a sorozatból.



2. Ábra. Az intervallum-felosztás elve: a kezdeti S intervallumot előbb felosztjuk az X_1, X_2, X_3 intervallumokra, majd az X_3, intervalumot az X_4, X_5 intervallumokra.

A felső korlátokra hasonló megállapítás nem tehető. Igaz azonban, hogy ha az 1. Algoritmus 1. pontjában felbontásra kerülő intervallumok egy-egy véletlen módon kiválasztott pontjában felvett célfüggvényértékek sorozatát tekintjük, akkor annak alsó torlódási pontja éppen a globális minimum.

A következő fejezetben részletes leírásra kerülnek az intervallum-felosztási módszerek, azok válfajai és tulajdonságaik. A vizsgálatok középpontjában a különböző algoritmusok konvergenciája, illetve az általuk szolgáltatott eredmény pontossága áll.

3. Az intervallum-felosztási eljárások

A következőkben kizárólag a következő alakú globális optimalizálási problémákkal foglalkozunk:

$$\min_{x \in X} f(x), \tag{8}$$

ahol $X \in \mathbf{I}^n$ több dimenziós valós intervallum és f: $\mathbf{R}^n \to \mathbf{R}$ tetszőleges, számítógéppel kiszámítható valós függvény. A gyakorlatban leggyakrabban előforduló korlátozási feltételek nélküli optimalizálási problémák ilyen alakban mindig felírhatók. A feladat ilyenkor nem tartalmaz valódi, függvényekkel megadott korlátozási feltételeket, csupán a feladatban előforduló változókra (paraméterekre) rögzít bizonyos alsó és felső korlátokat.

A korlátozott feladatok esetében a megoldást egyrészt a célfüggvény módosításával érhetjük el. A célfüggvényhez olyan p: $\mathbf{R}^n \to \mathbf{R}$ ún. *büntetőfüggvény*t adunk, mely a lehetséges megoldások halmazán belül nulla értéket vesz fel, azon kívül pedig pozitív. Jelöljük a lehetséges megoldások halmazát S-sel, és legyen a globális optimum értéke $f^* := \min_{x \in S} f(x)$! A p büntetőfüggvény akkor jó, ha minden $x \notin S$ esetén $f^* < f(x) + p(x)$. Az intervallumos módszerek alkalmazásakor— más módszerekkel ellentétben — a büntetőfüggvényeket nem szükséges a lehetséges megoldások határán a célfüggvényhez illeszteni, amennyiben az eljárás nem tartalmaz a célfüggvény folytonosságát vagy differenciálhatóságát feltételező kiegészítő eljárásokat. Így általában a $p=c$ választás jó, ahol c egy elég nagy konstans. További megoldások találhatók a [37] forrásban.

Az intervallum-felosztási módszerek a 2.4. alfejezetben ismertetett elvet alkalmazzák a globális optimum befoglalásának finomítására. Ezen módszerek kétféleképpen gyorsíthatók. Az egyik mód a befoglalófüggvények javítása, a másik a kiegészítő eljárások, illetve az egyes lépések tökéletesítése. A következőkben az utóbbi utat választva előbb áttekintjük az eddig elért eredményeket, és újabb elméleti eredményekkel egészítjük ki őket, melyeket részben numerikus eredményekkel is demonstrálunk.

Az intervallum-felosztási eljárások számos, egymással ekvivalens módon definiálhatók (pl. [7, 9, 10]). A következő egyszerű eljárásvázlat egy ilyen lehetséges definíciót ad meg:

2. Algoritmus. Az intervallum-felosztási eljárás modellalgoritmusa.

1. Legyen L egy üres lista, $A := X$ az aktuális intervallum! Állítsuk a k iteráció-számlálót $k := 1$ értékre!

2. Daraboljuk az A intervallumot véges $s \geq 2$ számú A_i $(i=1,\ldots,s)$ részintervallumra úgy, hogy $A = \cup_{i=1}^{s} A_i$ és $(\forall i,j=1,\ldots,s;\ i \neq j)$ $\text{int}(A_i) \cap \text{int}(A_j) = \varnothing$ teljesüljön, ahol az „int" egy halmaz belsejét jelöli!
3. Vegyük fel a listára az újonnan keletkezett intervallumokat, azaz legyen $L := L \cup \{A_1\} \cup \ldots \cup \{A_s\}$!
4. Töröljünk bizonyos elemeket a listáról, melyek nem tartalmazhatnak globális optimumhelyet!
5. Válasszunk ki egy új $A \in L$ aktuális intervallumot a listáról, és emeljük le onnan, azaz legyen $L := L \setminus \{A\}$!
6. Ha a megállási feltétel nem teljesül, növeljük az iteráció-számlálót, $k := k+1$, és térjünk vissza a 2. lépéshez!
7. Nyomtassuk ki az eredményt, és STOP!

A 2. Algoritmus részleteket nem rögzíti, a konkrét eljárások az egyes lépések pontosításával nyerhetők. Az algoritmus leállásakor a listán maradt részintervallumok uniója tartalmazza az összes globális minimumhelyet.

A modern intervallum-felosztási módszerek felépítése néha eltér ugyan a fenti modellalgoritmus szerkezetétől, elméletileg azonban az esetek túlnyomó többségében annak megfelelően működik, annak segítségével jól leírható, illetve arra visszavezethető. Az elméleti vizsgálatokhoz egyszerűsége miatt a 2. Algoritmust használjuk. A modellalgoritmus egyes lépéseinek rögzítése, illetve kifejtése az intervallum-felosztási módszerek széles körét definiálja.

Az algoritmus 1. lépése az előkészítő rész, mely csak egyszer kerül végrehajtásra. Az L listán mindig az X azon részintervallumai találhatók, melyekről az algoritmus eddig nem tudta még eldönteni, kizárhatók-e a további vizsgálatokból. A szükséges függvényhívások számának csökkentése érdekében a listán minden egyes Y részintervallummal együtt tárolhatók további információk, mint például az Y felett számított befoglalófüggvény érték alsó korlátja, vagy a célfüggvény deriváltfüggvényére adható befoglalás vetületeinek szélességei. Az iteráció-számláló a későbbiekben ismertetésre kerülő elméleti vizsgálatok miatt szükséges.

3.1. Az intervallumok felosztása

A 2. Algoritmus 2. lépésével kezdődik az iterációs rész. Azok a módszerek, amelyekben $s=2$ és a keletkezett két új részintervallum egybevágó, azaz az A intervallumot két egyenlő részintervallumra vágjuk, az *intervallumfelező eljárások*. A leggyakoribb és egyben legegyszerűbb megvalósítása a 2. lépésnek, amikor az A intervallumot az egyik legszélesebb élére merőlegesen vágjuk két egyenlő részre. A

vágás iránya azonban lényegesen megváltoztathatja az algoritmus sebességét. Az erre vonatkozó vizsgálatok négy különböző módszert emelnek ki a vágás irányának kijelölésére vonatkozólag [25]. A vágás irányának kijelölése minden lépésben egy-egy k (k=1,...,n) szám kiválasztását jelenti, mely egy D: $\mathbf{N}\rightarrow\mathbf{R}$ segédfüggvény segítségével adható meg:

$$k := \min_{j=1}^{n} \{j : \mathrm{D}(j) = \max_{i=1}^{n} \mathrm{D}(i)\},$$

ahol k adja meg azt a koordinátát, melyre merőlegesen a vágás történni fog. A k kiválasztását a D függvény definíciója egyértelműen meghatározza. Az egyes vágási irányokat rögzítő szabályokhoz tartozó D függvények:

A szabály. Ez a fentebb említett legszélesebb élre merőleges vágást adó irány [47, 59], melyet a következő D ad meg:

$$\mathrm{D}(i) := \mathrm{w}(X_i),$$

ahol az alsó index a komponenst — vetületet jelöli. Ez a technika uniform részintervallumok generálását célozza.

B szabály. A következő függvényt G.W. Walster ajánlotta, és E.R. Hansen írta le először [36]:

$$\mathrm{D}(i) := \mathrm{w}(F_i'(X))\mathrm{w}(X_i).$$

Ez a D függvény a célfüggvény egydimenziós projekcióinak változására ad felső korlátot.

C szabály. Formailag a B szabályban szereplő D segédfüggvényhez hasonlóan adható meg ez a szabály, melyet D. Ratz definiált először [57]:

$$\mathrm{D}(i) := \mathrm{w}(F_i'(X)(X_i - \mathrm{m}(X_i)).$$

Amennyiben $\mathrm{lb}(F_i'(X))=0$ vagy $\mathrm{ub}(F_i'(X))=0$, a B és C szabályok megegyeznek. Más esetekben ez nem igaz.

D szabály. Az A szabályhoz hasonlóan ez sem igényli a deriváltfüggvény befoglalófüggvényének kiszámítását, a hozzá tartozó segédfüggvény [25]:

$$\mathrm{D}(i) := \begin{cases} \mathrm{w}(X_i) & 0 \in X_i, \\ \mathrm{w}(X_i) / \min\limits_{x \in X_i} |x| & 0 \notin X_i. \end{cases}$$

A D szabály az A szabályhoz képest általában jobban kiküszöböli a befoglalófüggvény pontatlanságából adódó hibát, és előnyös a különböző nagyságrendű komponensek esetén.

A vágás során további jelentőséggel bírhat s, a keletkező részintervallumok száma. Az s növelésének következményeit felületesen vizsgálva annyi megállapítható, hogy kevesebb iteráció szükségeltetik ugyanolyan sűrűségű felbontás eléréséhez, ugyanakkor egy-egy iteráció több elemi műveletet követel. Az egy iterációs

cikluson belüli vágások számának növelésére több lehetőség is rendelkezésre áll, melyeket bővebben a 4. fejezetben tárgyalunk.

A szakirodalomban eddig alig vizsgált lehetőség a vágási arányoknak az ekvidisztáns beosztáson alapuló darabolástól eltérő megoldása. Kézenfekvőnek tűnik, hogy bizonyos esetekben az eljárás gyorsítható, ha az aktuális intervallum azon határához közelítve, mely határ közelében a célfüggvény a legkisebb értéket felveszi, egyetlen iterációs cikluson belül sűrítjük a vágások számát. Ennek természetes egyszerűsítése, hogy a 4. lépésben azokat az elemeket töröljük a további keresésből, melyek felett a célfüggvény bizonyíthatóan szigorúan monoton (ld. 3.3. fejezet).

3.2. Az intervallum-felosztási módszerek listakezelése

A 2. Algoritmus 3. lépésében a vágás következtében előálló új intervallumokat beillesztjük az L listába. A lista gyakorlati megvalósításakor figyelembe kell venni az algoritmus 5. lépésének végrehajtását. Attól függően, hogy milyen szabály szerint jelöljük ki, illetve emeljük le az új aktuális intervallumot a listáról, a lista elemeit célszerű lehet rendezetten tartani. A kiválasztást meghatározó értékek szerint rendezve a listát, arról mindig a legelső elem választható aktuális intervallumként. Ebben az esetben a lista $|L|$ méretétől függően legalább $s \cdot \log|L|$ művelet konstansszorosára van szükség az új intervallumoknak a rendezett listába való beillesztéséhez. Rendezetlen lista esetében ez s konstansszorosára redukálható, míg az új aktuális intervallum kijelöléséhez ebben az esetben a lista mind az $|L|$ elemét meg kell vizsgálni. Így, ha c_i-k a megfelelő konstansok, akkor egyetlen iteráció esetén a 4. lépésben történő törlés műveletétől eltekintve rendezetlen lista esetében összesen

$$c_1|L|+c_2 s, \tag{9}$$

míg rendezett listánál

$$c_3+c_4 s\log|L| \tag{10}$$

a listakezelés műveletigénye a legrosszabb esetben [17, 20]. Ha az eljárást k_0 iterációs cikluson keresztül folytatjuk, és eközben nem törlünk elemet a listáról, akkor megmutatjuk, hogy a kétféle listakezelés milyen nagyságrendbeli műveletigénnyel bír [20].

3.2.1. Rendezetlen listakezelés

Tekintsük először a rendezetlen esetet. A következőkben belátjuk, hogy ebben az esetben a listakezelésre fordított műveletigény az elvégzett iterációs ciklusok számától négyzetesen függ [20].

5. Tétel. Ha a 2. Algoritmusban alkalmazott lista rendezetlen, és az új listaelem kiválasztása szükségessé teszi a listaelemek vizsgálatát, akkor a k_0-dik iterációig a listakezelésre fordított műveletigény a legrosszabb esetben

$$T(s,k_0) = \Theta(sk_0^2). \tag{11}$$

Bizonyítás. A korábban megállapított (9) műveletigény alapján egyetlen iterációs ciklusban legrosszabb esetben $c_1|L|+c_2s$ művelet szükséges az aktuális intervallum listáról való kiválasztásához, illetve az újonnan keletkező intervallumok listára helyezéséhez, ahol a c_1, c_2 konstansok nem függenek a k_0, $|L|$ és s értékek egyikétől sem.

Tekintsük a 2. Algoritmus működését. Az első iterációs ciklus végén az L listán s-1 darab elem található, valamint minden további iteráció során a listaelemek száma s-1-gyel nő a legrosszabb esetben, tehát amikor nem törlünk semmit az L listáról. Ebből adódik, hogy $|L|=(s-1)k$ a k-dik iterációs ciklus végén. Ily módon a k_0-dik lépésig a listaelemek kezeléséhez szükséges műveletigény összesen

$$T(s,k_0) = \sum_{k=1}^{k_0}(c_1|L|+c_2s) = \sum_{k=1}^{k_0}(c_1(s-1)k+c_2s) = c_1(s-1)\frac{k_0^2+k_0}{2}+c_2sk_0,$$

amiből következik az állításbeli (11) egyenlőség. □

Lényeges, hogy a bizonyítás valóban nem tételezi fel sem az aktuális intervallum kiválasztására, sem annak darabolására vonatkozó szabályok ismeretét, azaz tetszőleges, a 2. Algoritmus által meghatározott intervallum-felosztási eljárás esetén alkalmazható. Ez a következőkben ismertetésre kerülő rendezett esetre akkor nem igaz, ha a lista rendezettségének megtartásához csupán konstansnyi műveletre van szükség minden iterációs ciklusban, mint például Hansen algoritmusánál.

3.2.2. Rendezett listakezelés

A következő eredmény a 5. Tételhez hasonló állítást fogalmaz meg rendezett listakezelés esetén [20].

6. Tétel. Ha a 2. Algoritmusban alkalmazott lista rendezett, és az új listaelem kiválasztása szükségessé teszi a listaelemeknek a lista rendezését adó érték szerinti

vizsgálatát, akkor a k_0-dik iterációig a listakezelésre fordított műveletigény a legrosszabb esetben

$$T(s,k_0)=\Theta(sk_0(\log s+\log k_0)). \tag{12}$$

Bizonyítás. A (10) műveletigény szerint a 2. Algoritmus egyetlen iterációs ciklusban a listaelemek kezelésére fordított műveletigény $c_3+c_4 s\log|L|$, ahol c_3 és c_4 független konstansok, valamint az 5. Tétel bizonyításából $|L|=(s-1)k$ a k-dik iteráció végén. A k_0-dik iterációig összegezve tehát

$$\begin{aligned} T(s,k_0) &= \sum_{k=1}^{k_0}\left(c_3+c_4 s\log(k(s-1))\right)= \\ &= c_3k_0+c_4 s\sum_{k=1}^{k_0}\log k+c_4 s\log(s-1)\cdot k_0. \end{aligned} \tag{13}$$

A (13) egyenlőségben eredményül kapott összeg második tagját a következőképpen becsülhetjük felülről:

$$\begin{aligned} \sum_{k=1}^{k_0}\log k \le \int_1^{k_0+1}\log x\,\mathrm{d}x &= \frac{1}{\ln a}\left[x\ln x-x\right]_1^{k_0+1}= \\ &= \frac{1}{\ln a}\left((k_0+1)(\ln(k_0+1)-1)+1\right)-\frac{k_0}{\ln a}, \end{aligned} \tag{14}$$

ahol a a kérdéses logaritmusfüggvény alapja, melynek értéke a nagyságrendet jól láthatóan nem befolyásolja.

Behelyettesítve a (14) egyenlőtlenséget a (13) egyenletbe, majd elvégezve a szükséges átalakításokat kapjuk a

$$T(s,k_0)\le c_3k_0+c_4 s\left(\frac{1}{\ln a}\left((k_0+1)(\ln(k_0+1)-1)+1\right)-\frac{k_0}{\ln a}\right)+c_4 s\log(s-1)\cdot k_0$$

egyenlőtlenséget. Ez igazolja az állítást. □

A (12) összefüggés szerint a k_0-dik iteráció befejeztéig a listakezelés $k_0\log k_0$ nagyságrendű műveletet igényel, azaz lényegesen kevesebbet, mint a rendezetlen esetben. Fontos azonban megjegyezni, hogy ez akkor igaz csupán, ha a rendezett L lista megvalósításakor olyan adatstruktúrát alkalmazunk, mely optimális műveletigényű keresést képes megvalósítani.

A listakezeléskor felmerül egy hibrid megoldás lehetősége is, melynek részleteit a következőkben fejtjük ki.

3.2.3. Vegyes listakezelés

Megvalósítható egy vegyes listakezelés is, mely a rendezett és rendezetlen listakezelés módszereit egyesíti. Ennek egyik változata [24] az, amikor egy p_0 előre rögzített hosszúságú külön listán tároljuk a kiválasztási rendezés szerinti legjobb első p_0 darab intervallumot. Ezt minden iterációs ciklusban az új részintervallumokkal frissítjük, ha lehetséges. Ellenkező esetben a rendezetlen listából pótoljuk az éppen törölt aktuális intervallumot.

Ehhez hasonló lehetséges módszer [20], hogy az L listának csupán az első, legfeljebb p_0 konstans darabból álló részét tartjuk rendezetten, a lista fennmaradó részébe egyszerűen „last in" módszerrel helyezzük fel a listaelemeket. A lista első p ($1 \le p \le p_0$) eleme a kiválasztási rendezés szerint talált legjobb elemekből áll. Ekkor minden egyes új aktuális intervallum kiválasztása a listáról valami c_1 konstans időt követel, mivel a lista első p elemének rendezettsége miatt a lista legelső eleme választható aktuális intervallumként. Egy új részintervallum listára helyezésekor először megvizsgáljuk, hogy az beilleszthető-e az L rendezett elemei közé. Ha igen, akkor beírjuk, és ezzel $p=p_0$ esetén a lista eddigi p_0-dik eleme a rendezetlen részbe tolódik, ha nem, akkor hozzáfűzzük őt a lista végéhez. Ez legrosszabb esetben $c_2 \log p_0$ műveletet igényel. Az új aktuális elem kiválasztása, valamint az új s darab elem listára helyezése összesen $c_5+c_6 s \log p_0$ műveletet igényel, ahol c_5 és c_6 konstansok. Ez önmagában azt jelentené, hogy tetszőleges k_0-dik iterációig a listakezelés műveletigénye csupán lineárisan függ k_0 értékétől, mivel p_0 értéke konstansként rögzíthető. Ez az eredmény azonban nem igaz például akkor, ha p_0 iterációs cikluson keresztül nem sikerül a lista rendezett részébe új intervallumot beilleszteni, azaz a lista rendezetlenné válik. Általános esetben p_0 értékét elég nagynak választva ez persze ritkán következik be, de elméletileg tetszőlegesen gyakran előfordulhat. Az ebből adódó hátrányokat elkerülendő két további módosítást eszközlünk az eljáráson, melyek az alábbiak.

1. A rendezett rész elemeit csak akkor pótoljuk, ha a rész teljesen kiürült, akkor azonban a listából a rendezés szerinti legjobb p_0 darabbal teljesen feltöltjük.
2. A p_0 értékét nem rögzítjük konstansként, hanem úgy állapítjuk meg, hogy az mindig az L lista hosszának egy konstans κ-adrésze legyen.

Az így előálló listakezelő algoritmust *hibrid listakezelés*nek fogjuk nevezni. A következő tétel ennek műveletigényét adja meg.

7. Tétel. Alkalmazzuk a 2. Algoritmusban a hibrid listakezelési módszert. Ekkor a k_0-dik iterációig a listakezelésre fordított műveletigény legrosszabb esetben

$$T(s,k_0)=O(k_0 \log(sk_0)). \qquad (15)$$

Bizonyítás. A feltételek szerint az elemeket ebben az esetben két elkülönített részben tároljuk, az egyik rendezett, a másik nem. Ha az összes tárolt elemek száma $|L|$, akkor a rendezett rész legfeljebb $p_0=\kappa|L|$ elemből áll, ahol $0<\kappa<1$ egy rögzített konstans.

A listakezelés folyamata két fő esetre különíthető el, az egyik az az eset, amikor a rendezett részben vannak elemek, a másik, amikor a rendezett rész kiürül, és azt a rendezetlen részből új elemekkel töltjük fel. Legrosszabb esetben az előbbi csupán p_0 iteráció után befejeződik, mivel a rendezett rész kiürül. A következő művelet, azaz a rendezett rész feltöltése egyetlen iteráció alatt elvégezhető.

A rendezett rész második esetbeli feltöltése a következőképpen zajlik. Először a rendezetlen listarész elemeiből rendezzük az első p_0 darabot

$$t_1=c_5 p_0 \cdot \log p_0 \tag{16}$$

idő alatt. Ezután a maradék $|L|-p_0$ elemet beillesztjük az új rendezett részbe, a kihulló elemek átkerülnek a rendezetlen rész már vizsgált elemei közé. Minden ilyen beillesztés $\log p_0$ nagyságrendű idő alatt elvégezhető. Az összes elemre ez

$$t_2=c_6(|L|-p_0)\cdot \log p_0 \tag{17}$$

műveletigényt jelent.

Az 5. Tétel bizonyításában alkalmazott gondolatmenet alapján tudjuk, hogy a k_0-dik iteráció után legrosszabb esetben a listaelemek száma $|L|=k_0(s-1)$. Alkalmazzuk ezt a (17) egyenletre, és a kapott képletet adjuk hozzá (16)-hoz:

$$t=c_5 p_0 \cdot \log p_0+c_6(k_0(s-1)-p_0)\cdot \log p_0. \tag{18}$$

Tudjuk, hogy az adott, L listájú iterációban p_0 értéke $\kappa|L|$, ahol κ rögzített konstans. Ez azonban azt jelenti, hogy

$$p_0=\kappa|L|=\kappa k_0(s-1). \tag{19}$$

Alkalmazzuk most ezt az összefüggést (18)-ban.

$$t=(c_5\kappa+c_6(1-\kappa))k_0(s-1)\ \log(\kappa k_0(s-1)). \tag{20}$$

Ezt a t darab műveletet legrosszabb esetben is csupán minden p_0-dik iteráció után szükséges elvégeznünk, ahol ebben az esetben p_0 értéke monoton növekszik. Amortizált analízist [5] alkalmazva a t által megadott műveletigényt szétoszthatjuk a rákövetkező p_0 iteráció között, megkapva így a műveletigény egyetlen iterációra vonatkoztatott átlagos értékét. Ha ehhez az $|L|$ aktuális értékét használjuk, akkor ezzel alsó becslést kapunk $|L|$ növő értékére, és így ezen keresztül p_0-ra. Ezek alapján az egyetlen iterációra vonatkoztatott időbonyolultság

$$k_0^{-1}T(s,k_0)\le \frac{t}{p_0}+c_7=\frac{t}{\kappa k_0(s-1)}+c_7, \tag{21}$$

ahol c_7 az első lépcső műveletigénye. Helyettesítsük most be a (20) egyenletet a (21) összefüggésbe:

$$k_0^{-1}T(s,k_0) \le \frac{c_5\kappa + c_6(1-\kappa)}{\kappa}\log(\kappa k_0(s-1)) + c_7,$$

ami megfelel a tételben kimondott eredménynek. □

A 7. Tétel elméleti legrosszabb esetre vonatkozó felső korlátja k_0-ban ugyanolyan jó, mint a rendezett esetben (6. Tétel), az s-től való függés szerint pedig annál is jobb. Vegyük észre, hogy ehhez az eredményhez a hibrid listakezelés mindkét feltétele szükséges. Amennyiben ugyanis az 1. feltételt megsértve a rendezett részt folyamatosan pótoljuk ki p_0 elemre, akkor legrosszabb esetben minden egyes iterációban végig kell keresnünk a lista rendezetlen részét, ami végső soron ugyanazt a k_0-tól való függést eredményezi, mint a rendezetlen eset (ld. 5. Tétel). A második feltételt elhagyva és p_0 értékét állandónak tekintve a p_0-nak a listahossztól való függése megszűnik, így az amortizált analízis olyan tagot eredményez a képletben, mely k_0-t első fokon tartalmazza egy iterációra vetítve, ami a felső korlátban másodfokú tagot ad, a rendezetlen esethez hasonlóan.

További kérdés, hogy vajon tovább növelhető-e a listakezelés hatékonysága, ami a k_0-tól való függést illeti. A következő tétel erre ad választ.

8. Tétel. Amennyiben a 2. Algoritmusban az elemek listára helyezése és az új aktuális elem kiválasztása nem eredményez azonos rendezettséget, akkor a listakezelés műveletigényének k_0-tól, azaz az elvégzett iterációk számától való függésére kapott $k_0 \log k_0$ nagyságrend legrosszabb esetben optimális.

Bizonyítás. Azt fogjuk igazolni, hogy amennyiben az adott nagyságrend nem lenne optimális, létezne olyan algoritmus, amely tetszőleges n elemet $o(n\log n)$-nél határozottan gyorsabban tudna rendezni.

Jelölje a rendezendő n darab adatot $a_1, a_2, \ldots, a_n$. Lineáris időben meghatározható a legnagyobb elem, $max := \max_{i=1,\ldots,n} a_i$, továbbá legyen a egy tetszőleges, max-nál nagyobb érték. Ezután az algoritmus a következő.

Tekintsük a 2. Algoritmus egy olyan megvalósítását, melyben a listakezelés legrosszabb esetben $k_0 \log k_0$-nál hatékonyabb, feltéve, hogy az algoritmus k_0 lépésben véget ér. Most felhelyezzük az n darab elemünket a listára, és elkezdjük őket onnan sorban levenni a kérdéses rendezés szerint. Minden eltávolított elem helyett legalább két új elemet helyezünk a listára, melyek értéke a fentebb megadott a.

Elégezve n darab iterációt, a megadott n darab adatot így a rendezésnek megfelelő sorrendben emeltük le a listáról kevesebb, mint $n\log n$ idő alatt, ami ellentmondás. □

1. Következmény. A rendezett és hibrid listakezelés legrosszabb esetben optimális műveletigényű.

3.2.4. A Hansen algoritmusok listakezelése

E.R. Hansen algoritmusának listakezelése jóval egyszerűbb az eddig tárgyaltaknál, a FIFO listakezelésnek felel meg. Ez azt jelenti, hogy mindig a lista legszélesebb intervalluma kerül felosztásra, és a listán szereplő intervallumok darabolása uniform intervallumokat eredményez. Az algoritmus ezáltal elég robusztus, maga a listakezelés minden iterációs ciklusban konstans műveletigényű, tehát a listakezelés iterációszámtól való függése lineáris. Hátránya, hogy a listaelemek törlése (a modellalgoritmus 4. lépése) nélkül semmi mást nem csinál, csupán egyenlő méretű, kis szélességű részintervallumokra darabolja a kiindulási intervallumot. Így a módszer — gyorsító eljárások kizárása esetén — legjobb esetben sem képes gyorsabban működni a legrosszabb esetnél, azaz a keresési fát szélességében teljesen bejárja.

A 2. Algoritmusban bevezetett L listán kívül az újabb eljárásokban szokás egy további, úgynevezett *eredménylistát* is felhasználni. Ez azt célozza, hogy az L lista elemei közül azokat, amelyek bizonyos korlátokon belül megközelítik a globális optimumot vagy annak helyét, leemeljük az L listáról, és áthelyezzük az eredménylistára. Így a további elsőrendű vizsgálatokból ezek a részintervallumok kizárhatók. Nagyobb jelentőséggel ez akkor bír, ha az optimumhelyek teljes halmazát kívánjuk minél pontosabban meghatározni. Az eredménylista a modellalgoritmusba beilleszthető, például az 5. lépés részeként. Ekkor az új 5. lépés a következő alakot veszi fel:

> 5′. Válasszunk ki egy új $A \in L$ aktuális intervallumot a listáról, és emeljük le onnan, azaz legyen $L := L \setminus \{A\}$! Ha A megfelel bizonyos feltételeknek, helyezzük fel az eredménylistára, és ismételjük az 5′. lépést! Egyébként térjünk rá a következő lépésre!

A 6. lépésben szereplő megállási feltétel ekkor az L lista kiürülése. Így a 2. Algoritmus vizsgálata magában foglalja ezen kettős listakezelést megvalósító eljárások vizsgálatát is.

3.2.5. Listakezelési eljárások legjobb esetben

A listakezelés iterációszámtól függő műveletigénye nyilván egy módszer esetében sem lehet jobb a lineárisnál, mivel minden iterációs ciklusban történik legalább konstans műveletigényű listaművelet. Ebből következik, hogy Hansen algoritmusa az elméleti vizsgálatok szempontjából optimális listakezelést hajt végre, listakezelésre fordított műveletigénye mind legjobb, mind legrosszabb esetben lineáris.

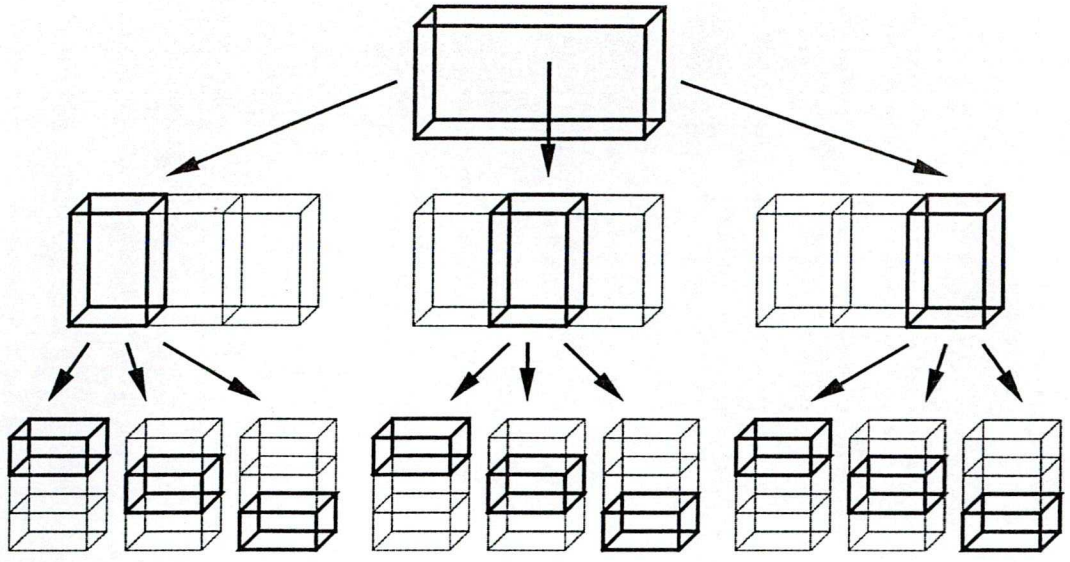
Rendezetlen listakezeléskor a legjobb eset is kvadratikus műveletigényt eredményez, mivel minden egyes lépésben végig kell vizsgálni a teljes listát az új aktuális elem kiválasztásához.

Rendezett listát alkalmazva legjobb esetben mindig az aktuális intervallum továbbdarabolásakor keletkező új intervallumok egyben a rendezés szerinti legjobbak. Így azok listára helyezése konstans műveletet igényel csupán. A rendezett listakezelés tehát a legjobb esetben lineáris műveletigényű. A hibrid listakezelés hasonló okok miatt szintén lineáris listakezelési műveletet igényel a legjobb esetben.

3.3. Gyorsító eljárások

A modellalgoritmus 4. lépésében bizonyos listaelemek törlésével kizárhatók a vizsgálatból azok a részintervallumok, melyek nem tartalmaznak globális optimumhelyet. Az ilyen törlési eljárásokat szokás *gyorsító eljárások*nak nevezni. Az intervallum-felosztási eljárások valójában a korlátozás és szétválasztás elvét alkalmazzák. Ezen belül a szétválasztás fázisa maga a vágás, míg a korlátozást bővebb értelemben a gyorsító eljárások biztosítják. A korlátozás és szétválasztás elvét fával ábrázolva (3. Ábra), annak minden csúcsán elhelyezhető egy-egy érték, amely az adott csomópontot reprezentáló A intervallumon a befoglalófüggvény $\mathrm{lb}(F(A))$ alsó korlátja. Eközben mindig az aktuális intervallum egy tetszőleges c pontjában kiszámítjuk a célfüggvény $f(c)$ értékét (garantáltan felfelé kerekítve). Ez nyilván felső korlát a globális minimumra nézve, így minden olyan A intervallum elhagyható a listából, melyen az $\mathrm{lb}(F(A))$ alsó korlát nagyobb, mint a talált legkisebb $f(c)$ érték.

A 2. Ábrán például az X_5 intervallum $\mathrm{m}(X_5)$ középpontjában a célfüggvény értéke nulla, és ez a felső korlát kizárja az X_2 intervallumot a további vizsgálatokból, ezért az az L listából törölhető. Ez a gyorsító eljárás, melyet *vágási tesztnek* (*cut-off test*) szokás nevezni, a legegyszerűbb és leggyakrabban alkalmazott módszer, mely Hansen módszerének szerves részét is képezi. Az $f(c)$ korlátot az algoritmus során mindig a talált legkisebb célfüggvény értékkel helyettesítjük, így az iterációs ciklusok során kiszámolt felső korlátokból álló sorozat monoton módon csökken. Más gyorsító eljárások segítségével szintén törölhetők elemek az eljárás fájából, ilyenkor a korlátozás fogalma a klasszikus értelemben vettől kissé eltér, és a módszer fájának korlátozását — annak bizonyos ágainak levágását — jelenti.



3. Ábra. Az intervallum-felosztások fájának első három szintje $s=3$ esetben.

A vágási teszt többféleképpen alkalmazható:

(a) Minden iterációs ciklus során a 4. algoritmikus lépésben a listáról töröljük a kiszűrt elemeket,

(b) mindig csupán az aktuális intervallumokat vizsgálva, azokat adott esetben darabolás előtt elvetjük és az 5. iterációs lépésben új aktuális intervallumot választunk,

(c) darabolás után a 3. lépésben csak azokat az új részintervallumokat illesztjük a listába, melyek a vágási teszt alapján még tartalmazhatnak globális minimumhelyet.

A háromféle megoldás más-más szempontból célravezető. A lista sorozatos átfésülése folyamatosan alacsonyan tartja a lista méretét, mely gyakorlati megvalósítások esetén egyébként már egyszerűbb feladatok megoldása esetén is meghaladhatja a rendelkezésre álló tárkapacitást. Ebben az esetben azonban a lista rendezettségétől függően legrosszabb esetben minden iterációs ciklusban a lista minden egyes elemét meg kell vizsgálni (ld. 3.2. fejezet). Amennyiben a lista a célfüggvény értékek alsó korlátja szerint rendezett, elegendő a lista végéről kezdeni, és az első nem törölhető listaelemnél abbahagyni a keresést. A másik két fentebb említett módszer a lista ideiglenes növelése árán megtakarítja a rendezetlen (vagy kizárási szempontból indifferens módon rendezett) lista teljes átfésülését. Csupán az aktuális elemeket vizsgálva azonban a listán fennmaradhatnak olyan elemek, melyek az időközben javított $f(c)$ korlát alapján már törölhetőek lettek volna. Ennél valamivel jobb megoldást kínál a darabolás (2. lépés) után keletkező részintervallumok vizsgálata, és a gyakorlatban ez is a legelterjedtebb eljárás.

A következőkben megvizsgáljuk, hogy a vágási teszt egyes megvalósításainál mekkora a teljes algoritmus műveletigénye. Mivel a legrosszabb esetben is a lista kezeléséhez és egyéb adminisztratív lépésekhez szükséges műveletek időigénye csekély a szükséges függvényhívásokéhoz képest, célszerű az egyszerűség kedvéért a szükséges függvényhívások számának kiszámítására szorítkozni. Először a legelterjedtebb, (c) pontban ismertetett megvalósítással foglalkozunk [22].

1. Lemma. A 2. Algoritmus törlésének megvalósításakor a fenti (c) pontban leírtakat alkalmazva az egyetlen iterációs ciklus végrehajtásához szükséges befoglalófüggvény hívások száma $\mathcal{O}(s)$, ha a vágási irány meghatározása nem vesz igénybe függvényhívást, és $\mathcal{O}(s+n)$ különben.

Bizonyítás. Mivel a befoglalófüggvények hívása a megfelelő valós függvények kiszámításához szükséges műveletigény konstansszorosát használják (függetlenül a kiszámítás módjától), a bizonyítás során nem teszünk különbséget a különböző függvénytípusok között. Tekintsük az egyes algoritmikus lépésekben szükséges függvényhívások számát.

Az 1. lépés az iteratív eljárás előkészítő része, így az a további vizsgálatokból kihagyható. Megjegyzendő azonban, hogy bizonyos esetekben az előkészítő részben az $f(c)$ kezdeti értékének nem a végtelent, vagy valami hasonlóan triviális kezdőértéket adunk, hanem esetleg valamilyen lokális keresővel egy elég jó felső korlátot keresünk a globális minimum értékére. Mivel enélkül az eljárás első számos iterációja a rossz felső korlát miatt a maximális, lépésenkénti s-1-gyel növelné az L listát, célszerű a gyakorlatban alkalmazni ezt a megoldást.

A 2. lépésben attól függően, hogy mi alapján választjuk ki az adott iteráció aktuális intervallumának vágási irányát (3.1. fejezet), szükség lehet néhány függvényhívásra. Amennyiben az A vagy D szabályok valamelyikét használjuk, nincs függvényhívás, ellenkező esetben a parciális deriváltfüggvény befoglalófüggvényének n-szeri hívására van szükség.

Az algoritmus 3. és 4. lépését egymással ötvözve a (c) esetben leírtak szerint hajtjuk végre. Mivel minden újonnan keletkezett A_i részintervallumra ki kell számítani a vágási teszthez az $\mathrm{lb}(F(A_i))$ korlátokat, így összesen s darab függvényhívásra van szükség a két lépés együttes elvégzéséhez. Ezt tekinthetjük úgy, hogy a 4. lépés használja a függvényhívásokat, míg a 3. lépéshez erre önmagában nincs szükség.

Az 5. lépés sohasem használ függvényhívásokat. Elméletben elképzelhető ugyan olyan megvalósítás, amikor rendezett lista esetében az L lista rendezését függvényértékek adják, és azok nem kerülnek a lista elemeivel együtt tárolásra, ez azonban a gyakorlatban nem fordul elő. Ekkor ugyanis az s darab új elem mindegyikének listába illesztése optimális eljárást választva is $\mathcal{O}(\log|L|)$ darab függvényhívást eredményezne, ahol $|L|=\mathcal{O}(s^m)$, ha m mélységben vagyunk az eljárás fájában. Rendezetlen lista esetében nyilván nincs szükség függvényhívásokra.

A modellalgoritmus 6. lépésében a megállási feltétel kiszámításához szükség lehet bizonyos függvényértékekre, ezek azonban az algoritmus előző lépéseiből tárolhatók, így külön függvényhívást nem eredményeznek.

Függvényhívásokat tehát csupán a 2. és 4. lépések használhatnak. A 4. lépés a vágási teszt miatt s darab függvényhívást igényel. A 2. lépésben a célfüggvénytől függetlenül választva a vágási irányt nincs függvényhívás, különben n darab függvényhívás szükséges. Ezek alapján a lemma mindkét állítása bizonyítást nyert. □

Mi történik akkor, ha a (c) vágási teszt helyett az (a) vagy a (b) megoldást választjuk? Könnyen ellenőrizhető, hogy a 3. és 4. lépés kivételével a többi függvényhívásai nem változnak. A (b) eljárást választva a 4. lépés legfeljebb egyetlen függvényhívást igényel, az (a) módszerhez pedig önmagában nincs is szükség függvényhívásra. A 3. lépés azonban mind az (a), mind pedig a (b) megvalósítás esetén s függvényhívást tesz szükségessé. Ellenkező esetben az 1. Lemma bizonyításában leírtak miatt az 5. lépés $\mathcal{O}(s^{m+1})$ extra függvényhívást kívánna meg (ahol m ismét a felosztás fájának aktuális szintje), ami elkerülhető az s darab új intervallum korlátainak kiszámításával és a listán való tárolásával. Ez alapján a következő megjegyzés tehető.

3. Megjegyzés. A 2. Algoritmus 4. lépésének megvalósításakor az (a) vagy (b) pontban leírtakat alkalmazva az egyetlen iterációs ciklus végrehajtásához szükséges befoglalófüggvény hívások száma mindkét esetben megegyezik az 1. Lemma eredményeivel.

A kapott eredmény jól tükrözi azt a tényt, hogy vágási teszt használata esetén minden keletkező részintervallum felett ki kell számítani a befoglalófüggvény értékét. Ebben az értelemben a vágási teszt három megvalósítási formája nem okoz lényegileg eltérő futási sebességet egyetlen iteráción belül.

A vágási teszten kívül számos más gyorsító eljárás is használható, amelyek mindegyike kombinálható a vágási teszttel és bármely másik gyorsító eljárással. Az egyes eljárások egymásra gyakorolt hatása még nyitott probléma, némely egyszerűbb módszerek esetleg gyengíthetik vagy helyettesíthetik más, nagy műveletigényű eljárások hatását. Ennek ellenére a gyakorlat azt mutatja, hogy több gyorsító eljárás együttes alkalmazása minden esetben javít az intervallum-felosztási algoritmus összevont sebességén sok feladat átlagában.

Az egyes gyorsító eljárások alkalmazhatósága függ a feladat f célfüggvényének tulajdonságaitól is. Amennyiben f folytonos függvény, a vágási teszt hatékonyabbá tehető azáltal, hogy bizonyos iterációs ciklusok után az f ezen tulajdonságát kihasználó helyi keresőket indít valamely részintervallumon a globális minimum egy jobb felső korlátjának megtalálására.

Ha f-nek létezik a gradiense, akkor felhasználható a monotonitási teszt, amely azt vizsgálja, hogy egy adott részintervallum felett a célfüggvény szigorúan mo-

noton-e. Ebben az esetben ez az intervallum minden részintervallumával együtt elhagyható. Ha az f gradiensfüggvényének befoglalófüggvénye F': $\mathbf{I}^n \to \mathbf{I}^n$, akkor a modellalgoritmus 4. lépése a következő alakot veszi fel:

4'. Töröljünk azokat az X_i elemeket a listáról, melyekre $0 \notin F'(X_i)$!

Amennyiben előre kizártuk, hogy az X_i határán az intervallum korlátok nem idézhetnek elő globális optimumhelyet (valódi korlátozási feltételek nélküli globális optimalizálási probléma), akkor a 4'. pontban kiszűrt elemek valóban nem tartalmazhatnak globális optimumhelyet.

Ha a célfüggvény folytonosan differenciálható, akkor a vágási teszthez szükséges felső korlátok sorozata előállítható legmeredekebb lejtő módszerekkel, melyek általában hatékonyabban működnek.

Ha f kétszer folytonosan differenciálható, akkor minden iterációs ciklusban a darabolás után keletkező részintervallumok mindegyike további vizsgálatoknak vethető alá az intervallumos Newton módszerek [36, 55, 57] segítségével, így azoknak csak bizonyos darabjai kerülnek fel az L listára. Az eredetileg valós iteratív módszer lényegét szolgáltató képletek egydimenziós esetben:

$$X_0 = X,$$

$$X_{i+1} = \left(\mathrm{m}(X_i) - \frac{f'(\mathrm{m}(X_i))}{F''(X_i)} \right) \cap X_i .$$

A módszer nyilván a célfüggvény deriváltfüggvényének zérushelyeit keresi meg. Gyorsító eljárásként használva általában egyetlen Newton lépést szokás megtenni az X_i elemre, szűkítve ezzel az adott részintervallumot. Ebből a szempontból a Newton módszer használata nem közvetlenül a listaelemeket törli, vagy a darabolás után keletkező részintervallumokat zárja ki a további vizsgálatokból, de csökkentheti a listára kerülő elemek össztérfogatát, finomítva ezzel a globális optimumhelyre és ezen keresztül az optimumra adott befoglalást. Az intervallumos Newton módszer alkalmas továbbá annak az eldöntésére is, hogy egy adott részintervallum egyetlen minimumpontot tartalmaz-e.

Inkább a befoglalófüggvények javítása érhető el azokkal a módszerekkel, melyeket néhány helyen az irodalom szintén a gyorsító eljárásokhoz sorol (pl. [55]), mivel javítják az intervallum-felosztási eljárások sebességét. Amennyiben f egyszer vagy többször folytonosan differenciálható, alkalmazhatók a befoglalás kiszámítására a középponti formulák (Taylor formula), az automatikus differenciálás, a lejtő függvények, a Baumann közepek, illetve más, a befoglalófüggvények pontosítását szolgáló módszerek. A gyakorlatban gyorsító eljárásnak minősül minden olyan módszer, amely csökkenti az előre meghatározott pontosságú közelítés eléréséhez szükséges műveletigényt.

3.4. Intervallum-kiválasztási szabály

A 2. Algoritmus 5. lépésében kerül kiválasztásra a következő iterációs ciklus A aktuális intervalluma, azaz a korlátozás és szétválasztás fájában (3. Ábra) a következő vizsgálandó csomópont. A kiválasztás több módon történhet, melyek mindegyike lényegében változtatja meg az algoritmust. A két leggyakrabban használt kiválasztási szabály R.E. Moore-tól [47] és S. Skelboe-tól [59], illetve E.R. Hansentől [34, 35] származik. Mindkét megoldás eredetileg intervallumfelező eljárásokhoz készült. A kiválasztási szabályokat tartalmazó két algoritmus szintén visszavezethető a 2. Algoritmusra.

R.E. Moore és S. Skelboe algoritmusa a listáról azt az elemet választja ki, amelyen a célfüggvény befoglalófüggvényének alsó korlátja a legkisebb. Ezzel a darabolások során eddig előállt intervallumok közül annak kiválasztását igyekszik megvalósítani, melyben a legnagyobb valószínűséggel található globális optimumhely. Ebben az értelemben tulajdonképpen egy mohó stratégiát követ. A Moore-Skelboe-féle eljárásnak az eredetivel ekvivalens, a 2. Algoritmusra visszavezethető formája a következő.

3. Algoritmus. R.E. Moore és S. Skelboe algoritmusa.

1. Legyen L egy üres lista, $A:=X$ az aktuális intervallum! Állítsuk a k iteráció-számlálót $k:=1$ értékre!
2. Osszuk fel az A intervallumot 2 darab A_i $(i=1,2)$ részintervallumra oly módon, hogy azt a leghosszabb élére merőleges irányban félbe vágjuk!
3. Vegyük fel a listára az újonnan keletkezett intervallumokat a célfüggvény befoglalófüggvényének alsó korlátaival együtt, azaz legyen $L:=L\cup(A_1,\mathrm{lb}(F(A_1)))\cup(A_2,\mathrm{lb}(F(A_2)))$ úgy, hogy a lista elemei második komponenseik szerint növekvő sorrendben rendezettek legyenek!
4. —
5. Válasszuk ki a listáról az $A\in L$ legelső intervallumot, és emeljük le onnan, azaz legyen $L:=L\setminus(A,\mathrm{lb}(F(A)))$!
6. Ha a megállási feltétel nem teljesül, növeljük az iteráció-számlálót, $k:=k+1$, és térjünk vissza a 2. lépéshez!
7. Nyomtassuk ki az eredményt, és STOP!

A Moore-Skelboe algoritmus eredeti változata nem tartalmaz semmiféle gyorsító eljárást, ennek ellenére önmagában is elég gyorsan igen jó korlátokat szolgáltat az optimum értékére vonatkozóan.

E.R. Hansen módszerének egyszerűbb listakezelése ellenére egy gyorsító eljárást tartalmaznia kell, hogy az intervallum-kiválasztási szabály ne duzzassza túl hosszúra a listát. Hansen az algoritmusában erre a vágási tesztet választotta, mely a célfüggvényről semmilyen speciális ismeretet nem feltételez. Hansen módszerének a 2. Algoritmusból származtatható alakja az alábbi.

4. Algoritmus. E.R. Hansen algoritmusa.

1. Legyen L egy üres lista, $A:=X$ az aktuális intervallum, $\tilde{f}:=\mathrm{ub}(F(\mathrm{m}(A)))$! Állítsuk a k iteráció-számlálót $k:=1$ értékre!
2. Osszuk fel az A intervallumot 2 darab A_i $(i=1,2)$ részintervallumra oly módon, hogy azt a leghosszabb élére merőleges irányban félbe vágjuk!
3. Helyezzük a lista végére az újonnan keletkezett intervallumokat a célfüggvény befoglalófüggvényének alsó korlátaival együtt, azaz legyen $L:=L\cup(A_1,\mathrm{lb}(F(A_1)))\cup(A_2,\mathrm{lb}(F(A_2)))$!
4. Töröljünk a listáról minden olyan elemet, melynek második komponense nagyobb, mint $\tilde{f}$.
5. Válasszuk ki a listáról az $A\in L$ legelső intervallumot, és emeljük le onnan, azaz legyen $L:=L\setminus(A,\mathrm{lb}(F(A)))$! Legyen $\tilde{f}:=\min(\tilde{f},\mathrm{ub}(F(\mathrm{m}(A))))$.
6. Ha a megállási feltétel nem teljesül, növeljük az iteráció-számlálót, $k:=k+1$, és térjünk vissza a 2. lépéshez!
7. Nyomtassuk ki az eredményt, és STOP!

Mindkét algoritmus garantálja, hogy a lista elemeinek uniója minden iterációs ciklusban tartalmazza az összes globális optimumhelyet.

A két eljárás hatékonysága nagyban függ a megállási feltételtől (amit 3.5. fejezetben tárgyalunk részletesen), amely valójában azt tükrözi, hogy csupán az optimum értékét, vagy annak helyét is elég pontosan meg kívánjuk-e határozni. A Moore-Skelboe algoritmus esetében a lista rendezettségénél fogva mindig az a részintervallum kerül továbbdarabolásra, amely várhatóan a legkisebb függvényértéket tartalmazza. A befoglalás pontatlansága miatt ez nem mindig teljesül. Az általánosan használt befoglalófüggvények α–konvergenciája (6. Definíció) miatt azonban igaz a (7) összefüggés, így az intervallumoknak a darabolások során csökkenő szélessége miatt a befoglalás egyre pontosabbá válik, ha csupán a tendenciát tekintjük. A darabolásnak ez a módja eredményesen törekszik tehát a globális optimumérték minél jobb befoglalására, míg a listán maradt többi elem között megmaradhatnak akár $\mathrm{w}(X)$ szélességűek is. Hansen algoritmusának lényege ezzel szemben egy sokkal egyszerűbb, *uniform darabolás*. Ez azt jelenti, hogy mivel a kiválasztási szabály mindig a legrégebben listára került, azaz a legszélesebb lista-

beli intervallumot választja ki továbbdarabolás céljából, a listaelemek egymástól legfeljebb egyetlen vágásban különböznek. (Amennyiben olyan, az intervallumokat továbbdaraboló gyorsító eljárásokat alkalmazunk, mint például az intervallumos Newton módszer, akkor persze nem feltétlenül igaz, hogy a legrégebben darabolt intervallum a legszélesebb.)

Mindkét algoritmus konvergál bizonyos feltételek mellett és bizonyos módon, ahogyan azt a következő három tétel kimondja. Az első ilyen eredmény Moore és Skelboe algoritmusára vonatkozik [55].

9. Tétel (Ratschek-Rokne). Amennyiben az f célfüggvény F befoglalófüggvénye teljesíti a (7) feltételt, a 3. Algoritmus által létrehozott $\mathrm{lb}(F(A))$ számokból álló sorozat alulról konvergál a globális minimum értékéhez.

Az optimumhely befoglalására a Moore-Skelboe algoritmus esetében semmiféle eredmény nem adható, mint azt a következő példa is mutatja.

Tekintsük az $f(x)=e^x-0{,}13x$ célfüggvényt (4. Ábra). f-nek nyilván egyetlen minimumhelye van, éspedig az x^*=-2,040220829 pont, ahol felveszi az $f(x^*)$=0,3952287077 globális minimum értékét. Alkalmazzuk most a Moore-Skelboe algoritmust az X=[-2,09 , -1,91] intervallumra. Tegyük fel, hogy egy globális optimumhelyet ε=0,1 pontossággal meg kívánunk határozni. Az első felezés után a listán megtalálható a két szomszédos Y=[-2,09 , -2], illetve Z=[-2 , -1,91] intervallum. Természetes kiterjesztést alkalmazva az F befoglalófüggvény kiszámítására adódik, hogy

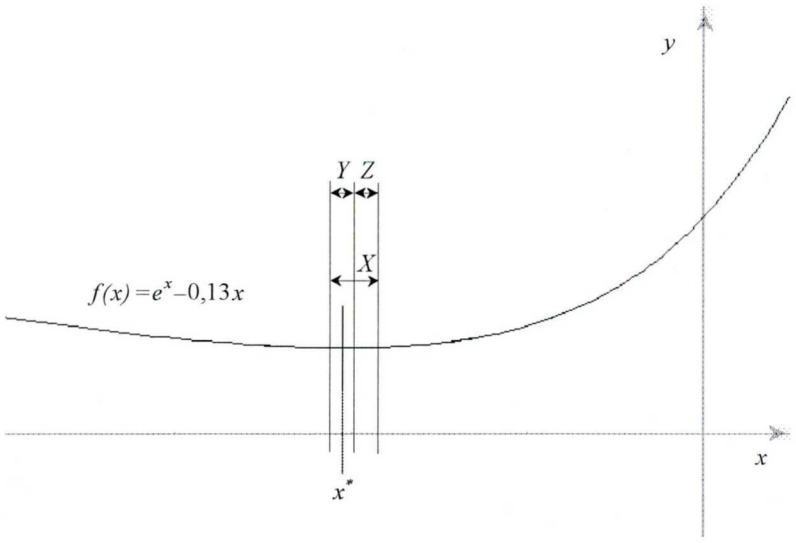
$$F(Y)=[0{,}3836871358\ ,\ 0{,}4070352832] \text{ és}$$
$$F(Z)=[0{,}3836352832\ ,\ 0{,}4080803866].$$

Mindkét intervallumra $\mathrm{w}(Y),\mathrm{w}(Z)<\varepsilon$, másrészt $\mathrm{lb}(F(Z))<\mathrm{lb}(F(Y))$, a listán tehát Z kerül az első helyre, míg Y csak a második. Ennek ellenére Z listáról való leemelésekor nem állíthatjuk, hogy megkaptuk a globális minimumhely egy ε pontosságú közelítését, mivel látszik, hogy ugyan $f(x^*)\in F(Z)$, azonban $x^*\notin Z$, pedig a globális minimumhely nincs sem X, sem pedig Y vagy Z határán.

A Moore-Skelboe algoritmus azonban viszonylag egyszerű feltételek mellett alkalmazható a globális optimum értékének befogalására. A konvergencia sebességére vonatkozó vizsgálatok azt mutatják, hogy további feltételek szükségesek ahhoz, hogy az előre meghatározott pontosságú befoglalás eléréséhez szükséges műveletszámra felső korlátot adhassunk [54, 55]. Ennek megadásához a következő két fogalom bevezetése szükséges:

10. Definíció. Legyen S egy konvergens sorozatokból álló halmaz. Azt mondjuk, hogy S *konvergenciarendje* $\{x_n\}$, ha bármely $\{y_n\}\in S$ sorozathoz létezik egy $p>0$ szám úgy, hogy $\forall n\in\mathbf{N}$ értékre $|y_n-y^*|<p\cdot\max_{m\geq n}|x_m|$, ahol $\{x_n\}$ egy nullához tartó konvergens sorozat, y^* pedig az $\{y_n\}$, sorozat határértéke. Továbbá, S

tetszőlegesen lassan konvergál, ha nem létezik hozzá olyan nullához tartó számsorozat, mellyel S konvergenciarendje megegyezne.



4. Ábra. Példa a Moore-Skelboe algoritmus bizonytalanságára az optimumhely megállapításakor.

A következő tétel a Moore-Skelboe algoritmus konvergenciájának sebességét adja meg [54].

10. Tétel (Ratschek-Rokne). Legyen F az f célfüggvény egy olyan befoglalófüggvénye, melyre teljesül, hogy

$$\mathrm{w}(F(Y)) \leq c\,\mathrm{w}^{\alpha}(Y) \text{ minden } Y \in \mathbf{I}(X) \text{ esetén.} \tag{22}$$

Jelölje S a Moore-Skelboe algoritmus által generált, az aktuális intervallumok alsó korlátjaiból álló összes sorozatok halmazát. Ekkor S tetszőlegesen lassan konvergál.

A 10. Tétel feltétele igen erős. (22) miatt egyrészt F α-konvergens, másrészt f folytonos és nyilván teljesül (7) is. Így a 9. Tétel értelmében az alsó korlátok konvergálnak a globális optimum értékéhez, a 10. Tétel szerint azonban ez a konvergencia tetszőlegesen lassú lehet. Pontosabban fogalmazva a tétel állítása azt mondja ki, hogy teszőleges konvergens számsorozathoz található olyan folytonos függvény és hozzá olyan α-konvergens befoglalófüggvény, hogy az alsó korlátok sorozata az adott számsorozatnál lassabban konvergál. A Moore-Skelboe algoritmus gyakorlati felhasználhatósága tehát csupán a (7) vagy akár a (22) feltétel kikötése mellett kérdéses. A legjobb esetben azonban igaz a következő állítás.

11. Tétel. Legyen f: $\mathbf{R}^n \to \mathbf{R}$ tetszőleges célfüggvény, melynek F befoglalófüggvénye α-konvergens. Ekkor legjobb esetben a Moore-Skelboe algoritmus k-dik iterációjának A aktuális intervallumára igaz, hogy

$$\mathrm{lb}(f(X))-\mathrm{lb}(F(A)) \le c 2^{-\alpha \lfloor k/n \rfloor} \mathrm{w}^{\alpha}(X), \tag{23}$$

ahol c a 6. Definícióból származó konstans.

Bizonyítás. Legyen az f egy befoglalófüggvénye a következőképpen definiálva: $G(Y):=[\mathrm{lb}(f(Y))-c\mathrm{w}^{\alpha}(Y),\ \mathrm{ub}(f(Y))]$. Itt tehát az F α-konvergenciájából származó c és α értékre építünk. Egyrészt G nyilván α-konvergens, másrészt az f tetszőleges F α-konvergens befoglalófüggvényére igaz, hogy $\mathrm{lb}(G(Y)) \le \mathrm{lb}(F(Y))$ minden $Y \in \mathbf{I}(X)$, ellenkező esetben $\mathrm{ub}(F(Y)) < \mathrm{ub}(f(Y))$ lenne, ami ellentmondana F befoglalófüggvény mivoltának. Ezen jelölések mellett a következő egyenlőtlenségek állíthatók fel tetszőleges $Y \in \mathbf{I}(X)$ intervallumra, melyek bizonyítják az állítást:

$$\begin{gathered}\mathrm{lb}(f(X))-\mathrm{lb}(F(Y)) \le \\ \le \mathrm{lb}(f(Y))-\mathrm{lb}(F(Y)) \le \mathrm{lb}(f(Y))-\mathrm{lb}(G(Y))=c\mathrm{w}^{\alpha}(Y)= \\ =c(2^{-\lfloor k/n \rfloor}\mathrm{w}(X))^{\alpha}.\end{gathered}$$

Az utolsó egyenlőség abból a feltételezésből adódik, hogy minden egyes felezéskor egy olyan újonnan keletkezett intervallumot választottunk ki újabb felezésre, mely tartalmaz globális optimumhelyet, azaz a legjobb esetet választottuk. □

A konvergencia-sebesség növelésének sarkalatos pontja a befoglalófüggvény izotonitása. Ezen feltétel mellett már legrosszabb esetben is felső korlát adható a módszer által szolgáltatott alsó korlát és a globális minimum értékének eltérésére [54]:

12. Tétel (Ratschek-Rokne). Legyen F az f célfüggvény egy izoton, α-konvergens befoglalófüggvénye. Ekkor

$$\mathrm{lb}(f(X))-\mathrm{lb}(F(A))=\mathcal{O}(k^{-\alpha/n}), \tag{24}$$

ahol A a Moore-Skelboe algoritmus k-dik iterációjának aktuális intervalluma.

A (24) egyenlet azt mutatja, hogy a Moore-Skelboe algoritmus konvergenciasebessége legrosszabb esetben exponenciális. Ugyanez az eredmény igaz Hansen algoritmusára is, azzal a módosítással, hogy ebben az esetben a befoglalófüggvényre az izotonitást nem szükséges kikötni [11, 12, 14]. Ez azon múlik, hogy Hansen algoritmusa uniform felezést hajt végre, ily módon az L listán található intervallumok mérete jól becsülhető. A következőkben némi előkészítés után bebizonyítjuk az erre vonatkozó tételt, ennek érdekében először bevezetjük az iterációs szint elnevezést [14].

11. Definíció. Azt mondjuk, hogy Hansen algoritmusának k-dik iterációs ciklusa az *l-dik (iterációs) szinten* van, ha az L listán található elemek szélességének maximuma $2^{-l}\mathrm{w}(X)$.

4. Megjegyzés. Ha k és $k+1$ iterációs ciklusok, és k az l-dik szinten van, akkor $k+1$ vagy az l-dik, vagy pedig az $l+1$-dik iterációs szinten található.

A 4. Megjegyzés szerint a 11. Definíció értelmes, azaz az iterációs szintek fogalma osztályozást határoz meg az iterációk halmazán, valamint ezek az osztályok sorrendben egymás után következő bizonyos számú iterációt foglalnak magukba.

Az alábbiakban következő 2., 3. és 4. Lemmák során mindig feltesszük, hogy Hansen algoritmusában a vágási teszt nem törölt listaelemet. Ezt a legrosszabb eset vizsgálatakor mindig megtehetjük, mivel a vágási teszt tetszőleges megvalósításához konstruálható olyan feladat (akár kontinuum számosságú is), melyen a vágási teszt nem képes egyetlen esetben sem elemet törölni az algoritmus L listájáról. A legegyszerűbb ilyen példa az $f(x)=a$ konstans függvény. Tetszőleges F befoglalófüggvényt alkalmazva $\mathrm{lb}F(Y)\leq f(x)=a$ minden $Y\in\mathbf{I}(X)$ és $x\in X$ értékekre. Így csupán a vágási teszt alapján a vizsgálatokból az iterációs eljárás semelyik lépésében sem zárható ki listabeli részintervallum.

Az iterációs szintek további jellemzéséhez először is megállapítjuk, hogy hány darab — egymást követő — iteráció tartozik egy iterációs szinthez. Ehhez a következő állítás szükséges, mely a 11. Definíció egyenes következménye [14].

2. Lemma. A Hansen algoritmussal az l-dik szint iterációs ciklusait elvégezve a tárolt intervallumok N_l száma a 2^n-szeresére nő, azaz $N_{l+1}=2^n N_l$.

Bizonyítás. Legyen Y egy tárolt intervallum, és legyen az aktuális iterációs ciklus egy adott szint első iterációja! Az Y intervallumot az eljárás során először a legszélesebb élére merőleges síkkal felezzük. Miután ezt a műveletet a többi tárolt intervallumon is elvégeztük, az iménti felezés eredményeként keletkező két új intervallum mindegyikén elvégezzük ugyanezt az eljárást.

Egy új iterációs szintre akkor érünk, amikor a felezéseket mind az n irányra merőlegesen elvégeztük. Ezáltal az Y intervallum 2^n darab vele hasonló részintervallumra hullott szét. Mivel eredetileg a tárolt elemek száma N_l volt, adódik az állítás. □

Ez alapján már megállapítható, hogy hány iteráció tartozik egy adott szinthez [14]:

3. Lemma. Az l-dik szint iterációs ciklusainak száma egyenlő az illető szint kezdeti listaelemeinek számát szorozva 2^n-1-gyel, ahol a 2. Lemma szerint a kezdeti elemszám 2^{nl}. Így az l-dik szinthez tartozó iterációk száma $2^{nl}(2^n-1)$.

Bizonyítás. Legyen Y egy tárolt intervallum, és legyen az aktuális iterációs ciklus egy adott szint első iterációja! Az Y intervallumot a szint iterációs ciklusai során 2^n darab egybevágó részintervallumra osztjuk fel. Minden iteráció során csupán egyetlen felezést végzünk, ezért a felezéseket leszámlálva megkaphatjuk az Y teljes felosztásához szükséges iterációs ciklusok számát.

Jelölje Y^0 az Y intervallumot mint geometriai alakzatot! Az ennek felezésekor előálló egybevágó intervallumokat mint alakzatokat jelölje Y^1, továbbá általában az Y^i felezése után előálló intervallumot Y^{i+1} (i=0, …, n-1). Ha $T(i)$ jelöli az Y^i teljes felosztásához még szükséges felezések számát, akkor nyilván $T(i)=2T(i-1)$. Ezek alapján az Y^0, és így az Y teljes felosztáshoz is $T(n)=2^n-1$ felezés szükséges.

Így egyetlen intervallum teljes felosztásához 2^n-1 iterációs ciklusra van szükség, amivel a lemmát igazoltuk □

Az iterációs szintek teljes jellemzését a következő állítás zárja, mely pontosan meghatározza, mely iterációk tartoznak egy bizonyos szinthez, valamint egy ezzel ekvivalens megállapítást tesz [14]:

4. Lemma. Az l-dik szint k-val jelölt iterációs ciklusára igaz, hogy

$$2^{nl} \le k \le 2^{n(l+1)}-1,$$

valamint ebből következően teljesül, hogy

$$k^{-1/n} \le 2^{-l} \le 2(k+1)^{-1/n}. \tag{25}$$

Bizonyítás. A $2^{nl} \le k$ és a $k \le 2^{n(l+1)}-1$ egyenlőtlenségek közül nyilván elég az elsőt bizonyítani, mivel a képletet az l+1-dik szintre alkalmazva azonnal adódik a második egyenlőtlenség. Bizonyítandó tehát, hogy az l-dik szint első iterációs ciklusa éppen a $k=2^{nl}$-dik.

A bizonyítást l szerinti indukcióval végezzük. Az állítás l=0-ra igaz. Legyen most $l \ge 0$ tetszőlegesen rögzített, és tegyük fel, hogy az állítás igaz. Ekkor ahhoz, hogy az l+1-dik szint első iterációs ciklusához érjünk, a 3. Lemma szerint $2^{nl}(2^n-1)$ darab iterációra van szükség, azaz a következő szint első iterációs ciklusa

$$2^{nl}+2^{nl}(2^n-1)=2^{nl}\cdot 2^n=2^{n(l+1)},$$

ami bizonyítja az állítást. Ezzel a lemmát igazoltuk. □

A Hansen algoritmus konvergencia-sebességét kimondó tétel előtt szükség van egy általánosabb állításra, melyet később még alkalmazni fogunk, és amely általános esetben ad felső korlátot a globális minimumra adható intervallumos alsó becslés mértékére [14]:

5. Lemma. Tetszőleges $Y \subseteq X \in \mathbf{I}^n$ intervallumokra mindig teljesül, hogy

$$\mathrm{lb}(f(X))-\mathrm{lb}(F(Y))\le c\mathrm{w}^{\alpha}(Y), \tag{26}$$

ahol F az f egy α-konvergens befoglalófüggvénye, c pedig a 6. Definícióból származó konstans.

Bizonyítás. A lemma állítása a következő könnyen ellenőrizhető egyenlőtlenségekből következik:

$$\begin{gathered}\mathrm{lb}(f(X))-\mathrm{lb}(F(Y))\le \\ \le \mathrm{lb}(f(Y))-\mathrm{lb}(F(Y))=\mathrm{ub}(f(Y))-\mathrm{w}(f(Y))-\mathrm{ub}(F(Y))+\mathrm{w}(F(Y))\le \\ \le \mathrm{w}(F(Y))-\mathrm{w}(f(Y))\le \\ \le c\mathrm{w}^{\alpha}(Y).\end{gathered}$$

□

Ezek után könnyen bizonyítható, hogy Hansen algoritmusa legrosszabb esetben a Moore-Skelboe algoritmushoz hasonlóan exponenciális konvergenciát mutat [14]:

13. Tétel. Ha F az f célfüggvénynek α-konvergens befoglalófüggvénye, akkor legrosszabb esetben is teljesül, hogy

$$\mathrm{lb}(f(X))-\mathrm{lb}(F(A))\le c(2\mathrm{w}(X))^{\alpha}(k+1)^{-\alpha/n}, \tag{27}$$

ahol A a Hansen-algoritmus k-dik iterációjának aktuális intervalluma.

Bizonyítás. Alkalmazzuk az 5. Lemma (26) eredményét az A aktuális intervallumra,

$$\mathrm{lb}(f(X))-\mathrm{lb}(F(A))\le c\mathrm{w}^{\alpha}(A),$$

majd helyettesítsük be a kapott egyenlőtlenség jobboldalába az iterációs szintek 11. Definícióját:

$$\mathrm{lb}(f(X))-\mathrm{lb}(F(A))\le c(2^{-l}\mathrm{w}(X))^{\alpha}.$$

Alkalmazva ezután a 4. Lemma (25) eredményét, adódik a következő egyenlőtlenség:

$$\mathrm{lb}(f(X))-\mathrm{lb}(F(A)\le c(2(k+1)^{-1/n}\mathrm{w}(X))^{\alpha}.$$

A kapott eredményt átalakítva kapjuk az állítást. □

Összevetve a 12. és a 13. Tételek eredményeit látszik, hogy Hansen algoritmusa az izotonitási feltételt elhagyva is legrosszabb esetben exponenciális konvergenciát mutat. Ebben az értelemben a Hansen algoritmus robusztusabb, bár a széleskörben elterjedt és a 2.2. fejezet pontjaiban tárgyalt befoglalófüggvények a középponti formula kivételével izotonak. Nem izoton befoglalófüggvények adódhatnak azonban mérési eredményekből származó, explicit módon meg nem adott célfüggvé-

nyek esetén is, így a különböző ipari, kísérleti alkalmazások során ez a különbség szerepet játszhat.

Explicit módon megadott célfüggvények esetén a tesztfeladatokon végzett futtatási eredmények szerint azonban a Moore-Skelboe algoritmus sebességben lényegesen jobb eredményeket szolgáltat, mint Hansen eljárása (ld. 1., 2. és 3. Táblázat). Ez annak tudható be, hogy átlagos esetben Moore és Skelboe intervallumkiválasztási szabálya a legrosszabb esetnél gyorsabban képes megfelelő befoglalást adni az optimum értékére vonatkozólag, sőt, elérheti a 11. Tételben megfogalmazott legjobb esetet is. Ezzel szemben Hansen módszerére a legjobb esetben sem adható a 13. Tételbeli $\mathcal{O}(k^{-\alpha/n})$ konvergenciánál jobb eredmény, amint azt a következő tétel kimondja:

14. Tétel. Legyen f: $\mathbf{R}^n \to \mathbf{R}$ tetszőleges célfüggvény, melynek F befoglalófüggvénye α-konvergens. Ekkor legjobb esetben a Hansen algoritmus k-dik iterációjának A aktuális intervallumára igaz, hogy

$$\mathrm{lb}(f(X))-\mathrm{lb}(F(A))\leq c\,\mathrm{w}^{\alpha}(X)k^{-\alpha/n}, \tag{28}$$

ahol c a 6. Definícióból származó konstans. Az $\mathcal{O}(k^{-\alpha/n})$ rendnél jobb felső becslés az alsó korlátok konvergenciájára nem adható.

Bizonyítás: A legjobb eset vizsgálata itt az algoritmus szempontjából legjobb esetet jelenti, így a 13. Tételbeli levezetéssel megegyezően igaz, hogy

$$\mathrm{lb}(f(X))-\mathrm{lb}(F(A))\leq c(2^{-l}\mathrm{w}(X))^{\alpha}, \tag{29}$$

mivel a becslés független az algoritmus által éppen megoldandó feladattól. Az eljárás során a legjobb esetben 2^{-l} értékére a (25) egyenlőtlenségek alapján $2^{-l}=k^{-1/n}$ teljesül, amit a (29) egyenlőtlenségbe behelyettesítve adódik a tétel állítása.

A következőkben belátjuk, hogy a (27)-ben és (28)-ban adott felső korlát nagyságrendben éles. Válasszuk ehhez a lehető legrosszabb, még α-konvergens befoglalófüggvényt. Feltesszük tehát, hogy az A aktuális intervallum esetében az α-konvergencia (6) feltételében egyenlőség áll fenn. Ebből adódik, hogy $Y=A$ helyettesítés esetén az 5. Lemma (26) egyenlőtlenségében szintén az egyenlőség áll fenn, amennyiben $\mathrm{lb}(f(X))=\mathrm{lb}(f(A))$ azaz az aktuális intervallum tartalmaz globális minimumhelyet, valamint $\mathrm{ub}(f(A))=\mathrm{ub}(F(A))$ teljesül, tehát a befoglalófüggvényünk által szolgáltatott felső korlát az aktuális intervallumon pontos. A 4. Lemma (25) összefüggése miatt legjobb esetben is, tehát amikor még éppen egy új iterációs szintre lépünk $2^{-l}=k^{-1/n}$ teljesül. A leírtakat formulába öntve adódik az állítás második része, azaz létezhet olyan eset, amikor

$$\begin{aligned}&\mathrm{lb}(f(X))-\mathrm{lb}(F(A)=\\ =c\,\mathrm{w}^{\alpha}(A)&=c(2^{-l}\mathrm{w}(X))^{\alpha}=c(k^{-1/n}\mathrm{w}(X))^{\alpha}=\\ &=\mathcal{O}(k^{-\alpha/n}).\end{aligned}$$

□

Másrészt a 13. Tétel (27) egyenlőtlenségének baloldalán álló kifejezés csupán abban az esetben fejezi ki a globális minimumérték alsó közelítésének jóságát, ha az A intervallum az Y listaelemek közül egy olyan, amelyiken $\mathrm{lb}(F(Y))$ minimális.

Arra vonatkozólag, hogy ez a tény, valamint a 14. Tétel a gyakorlatban mennyire érezteti hatását, a következő futtatási eredmények szolgálnak példával [14]. A futtatásokhoz C++ programozási nyelven önállóan megvalósított intervallum-aritmetikát használtunk. A tesztfeladatok nagyobbik része közismert problémákból került ki, míg az utolsó négy feladatot mi magunk konstruáltuk. A feladatok leírása megtalálható a függelékben.

A tesztfeladatokon végrehajtott futtatások során első alkalommal mindkét intervallum-kiválasztási szabály esetében alkalmaztuk a vágási tesztet, míg a második esetben egyikhez sem. Ily módon lehetőség nyílt az intervallum-kiválasztási szabályok eltérő hatását anélkül vizsgálat alá vonni, hogy az eredeti Moore-Skelboe algoritmusba beépített gyorsító eljárás az eredményeket eltérően befolyásolta volna. Megállási feltételként a $w(F(A))<\varepsilon$ (ld. 3.5. fejezet (30) feltétel) képletet alkalmaztuk. Minden esetben a két intervallum-kiválasztási szabályon kívül a kísérleteket elvégeztük véletlen intervallum-kiválasztással is. Ezek eredményeinél tíz független futtatás átlagos, illetve legjobb eredményeit rögzítettük. Az 1. és a 2. Táblázatok első oszlopában találhatók a tesztfeladatok elnevezései, illetve azok rövidítése. A második oszlop tartalmazza a megállási feltételhez alkalmazott ε értékeket, a harmadik oszlop pedig a kiindulási intervallumok szokásos értelemben vett térfogatát. A negyediktől a hetedik oszlopig a Moore-Skelboe, a Hansen és a véletlen intervallum-kiválasztási szabállyal ellátott módszerek által szolgáltatott eredmények találhatók. Az eredmények hiányoznak azokban az esetekben, ahol az eljárások által kezelt L lista betelt, vagy a véletlen intervallum-kiválasztás esetén, ha a kapott eredmények szórása olyan mértékű volt, hogy az eredmények nem szolgáltak semmiféle információval.

Az 1. és a 2. Táblázatban az első, vágási teszttel kiegészített futtatássorozat eredményei találhatók. Az 1. Táblázatban foglaltuk össze az egyes módszerek által felhasznált függvényhívások számát. Jól látszik, hogy bár Hansen algoritmusa jobb eredményeket ért el, mint a véletlen kiválasztás, a Moore-Skelboe algoritmus lényegesen kevesebb függvényhívás árán képes volt az előre rögzített pontossággal befoglalást adni a globális minimum értékére. Moore és Skelboe intervallum-kiválasztási szabálya esetén átlagosan 3705, míg Hansen módszerénél 8175 függvényhívásra volt szükség (10 eset figyelembevételével). A véletlen kiválasztás átlagos esetben 10258, de még legjobb esetben is 8261 függvényhívást igényelt a 80 eredményes futtatás átlagában, mely feladatokra a Moore-Skelboe algoritmus 3004, Hansen módszere pedig 8422 függvényhívást igényelt átlagban.

Érdekes ugyanakkor, hogy a tendencia nem ilyen egyértelmű a felhasznált maximális listahossz esetében. Az erre vonatkozó eredmények a 2. Táblázatban találhatók, és a tesztfeladatoktól függően erősen eltérnek egymástól. Átlagban a vizs-

gált problémákon a maximális listahossz az első két módszernél közel azonosnak mutatkozott (352, illetve 358), míg a véletlen intervallum-kiválasztási szabály ennél határozottan jobb (átlagos esetben 220, legjobb esetben 166) eredményeket adott. Ez utóbbi eredményt azonban nem érthetjük úgy, mintha ez a módszer általában kedvezőbb tárigényű lenne, hiszen 3 feladatot éppen a listahosszak túlnövekedése miatt nem tudott mind a 10 független futtatás során megoldani.

A tesztfüggvény jellemzői			Módszerek			
Függvény	**ε**	**Térfogat**	**Mo.-Sk.**	**Hansen**	**V (átlag)**	**V (min.)**
Levy 1	$5 \cdot 10^{-2}$	8	10617	11351	–	–
Levy 2	10^{-4}	20	2402	3016	–	–
Rosenbrock	10^{-18}	10^{6}	934	3918	3410	2956
THCB	10^{-18}	5	1370	2101	2130	1960
Booth	10^{-18}	10^{14}	1097	1926	2431	1999
Matyas	10^{-4}	400	18089	19317	33559	27026
Powell	10^{-7}	10^{4}	1489	10614	12635	10354
DSC 2	10^{-18}	3969	231	941	1173	869
DSC 3	10^{-18}	250047	353	4573	4789	2850
DSC 4	10^{-18}	15752961	472	23988	21938	18070
DSC 5	10^{-18}	992436543	591	–	–	–

1. Táblázat. Függvényhívások száma a vágási teszt használata esetén.

A vágási teszt nélkül végzett futtatások esetében csupán a maximális listahosszakra adunk meg eredményeket (3. Táblázat). Ennek az az oka, hogy mivel semmiféle gyorsító eljárást nem alkalmaztunk, a maximális listahossz egyben a végső listahossz is, és ennek konstansszorosa a szükséges függvényhívások száma. Ebben az esetben a Moore-Skelboe és a Hansen algoritmus által szolgáltatott átlagos eredmények (35 és 616 a Levy 1 feladat figyelmen kívül hagyása mellett) aránya jelentősen eltér a vágási teszt alkalmazásakor kapott eredmények (2. Táblázat) arányától. Ennek oka, hogy Hansen kiválasztási szabálya szélességi keresést valósít meg az iterációk fájában, ami törlés nélkül jelentős felesleges iterációs lépést, és ezáltal listaelemet eredményez. A 2. és 3. Táblázat eredményei közvetlenül nem összehasonlíthatók, mivel a vágási teszt nélküli futtatásokhoz új kiindulási és megállási feltételeket kellett szabni.

3.5. Megállási feltétel

A 2. Algoritmus 6. lépése tartalmazza az iterációs eljárás megállási feltételét. A megállási feltétel két szempontból is különös figyelmet érdemel. Először, helyes megválasztása esetén csökkenthető a kívánt eredmény eléréséhez szükséges iterációs ciklusok száma, amennyiben a megállást ahhoz az információhoz kötjük, melyre ténylegesen kíváncsiak vagyunk. Másodszor, a különböző megállási feltételeket az egyes eljárásokkal ötvözve, azok hatékonysága lényegesen megváltozhat.

A tesztfüggvény jellemzői			Módszerek			
Függvény	**ε**	**Térfogat**	**Mo.-Sk.**	**Hansen**	**V (átlag)**	**V (min.)**
Levy 1	$5 \cdot 10^{-2}$	8	1830	1475	–	–
Levy 2	10^{-4}	20	413	357	–	–
Rosenbrock	10^{-18}	10^{6}	120	87	62	41
THCB	10^{-18}	5	91	37	26	22
Booth	10^{-18}	10^{14}	139	11	18	13
Matyas	10^{-4}	400	826	483	589	454
Powell	10^{-7}	10^{4}	196	321	254	213
DSC 2	10^{-18}	3969	32	20	25	17
DSC 3	10^{-18}	250047	55	122	104	77
DSC 4	10^{-18}	15752961	75	671	684	490
DSC 5	10^{-18}	992436543	95	–	–	–

2. Táblázat. A maximális listahossz a vágási teszt használata esetén.

A megállási feltételek alapvetően két osztályba sorolhatók [18, 21, 19]. Az első ilyen osztályba tartoznak azok, melyek az optimum értékére követelnek meg bizonyos pontosságú közelítést, a másikba az optimum helyére vonatkozót előírók.

A tesztfüggvény jellemzői			Módszerek	
Függvény	ε	Térfogat	Mo.-Sk.	Hansen
Levy 1	$5\cdot10^{-2}$	8	2929	–
Levy 2	10^{-1}	20	43	1495
Rosenbrock	10^{-3}	1	13	511
THCB	$5\cdot10^{-2}$	5	99	721
Booth	$5\cdot10^{-3}$	20	15	298
Matyas	$5\cdot10^{-2}$	4	115	1023
Powell	2	625	30	173
DSC 2	10^{-4}	25	8	255
DSC 3	10^{-2}	125	10	1023
DSC 4	$5\cdot10^{-2}$	625	8	135
DSC 5	$5\cdot10^{-2}$	3125	10	527

3. Táblázat. A maximális listahossz a vágási teszt használata nélkül.

3.5.1. Az optimumérték befoglalása

Amennyiben az elsődleges cél a célfüggvény optimumértékének bizonyos hibakorlátok közötti kiszámítása, a megállási feltételnek is ezt a célt kell tükröznie. A legegyszerűbb ilyen megállási feltétel a következő:

$$w(F(A)) < \varepsilon. \tag{30}$$

Ezt használva megállási feltételként, a kapott $F(A)$ intervallum garantáltan tartalmazza a globális optimumértéket, amennyiben az új listaelem kiválasztása Moore és Skelboe szabálya szerint történik. Ez abból adódik, hogy a listaelemek uniója tartalmazza az összes globális optimumhelyet, így arra az Y listaelemre, melyre $\mathrm{lb}(F(Y))$ minimális, $\mathrm{lb}(F(Y)) \le \mathrm{lb}(f(X))$ teljesül. Másrészt $\mathrm{lb}(f(X)) \le \mathrm{ub}(F(Y))$ minden Y listaelemre igaz, mivel F befoglalófüggvénye f-nek. Más intervallum-kiválasztási szabályokat alkalmazva az előbbi megállapítás nem feltétlenül igaz, mivel $\mathrm{lb}(F(A))$ nagyobb lehet a globális minimumértéknél. Valóban, Hansen kiválasztási szabályát felhasználva például az uniform darabolás miatt tetszőleges A aktuális intervallum esetében teljesülhet a (30) feltétel. Így (30) nem feltétlenül teljesül azon Z listelemekre, melyekre $\mathrm{lb}(f(X)) \in F(Z)$, még ha azokat sikerül is a többi listaelem közül kiszűrnünk. Az alábbi állítás azonban mindig igaz [21].

15. Tétel. Teljesüljön a 4. Algoritmus 6. lépésében az A aktuális intervallumra a (30) megállási feltétel, tegyük fel továbbá, hogy az f célfüggvény Lipschitz-folytonos, és F befoglalófüggvénye α–konvergens. Igaz ekkor, hogy

$$\mathrm{ub}(F(A)) - \min_{(Y,y)\in L\cup\{A\}} y < c2^{-\alpha l}\mathrm{w}^{\alpha}(X) + K2^{-l}\,\mathrm{w}(X) + \varepsilon, \tag{31}$$

ahol X a kiindulási intervallum, l az aktuális iterációs szint, c a befoglalófüggvény α–konvergenciájának 6. Definíciójából származó együttható, K a célfüggvény Lipschitz-konstansa, $\varepsilon>0$ pedig a megállási feltételből származó szám.

Bizonyítás. Legyen Y egy tetszőleges, a listán tárolt intervallum. Ekkor igaz, hogy

$$\mathrm{lb}(F(A)) \le \mathrm{ub}(F(Y)). \tag{32}$$

Valóban, tegyük fel az ellenkezőjét. Nyilván $\tilde{f} \le \mathrm{ub}(F(\mathrm{m}(Y))) \le \mathrm{ub}(F(Y))$ minden listabeli Y intervallumra teljesül, így (32) ellenkezőjét feltéve adódna, hogy $\tilde{f} < \mathrm{lb}(F(A))$. Ez azonban azt jelentené, hogy az A intervallumot már az eddigi lépések valamelyikében törölnünk kellett volna az L listáról, a (32) összefüggés tehát igaz.

Innen azonnal adódik, hogy

$$\begin{aligned} &\mathrm{ub}(F(A)) - \mathrm{lb}(F(Y)) = \\ &= \mathrm{ub}(F(A)) - \mathrm{lb}(F(A)) + \mathrm{lb}(F(A)) - \mathrm{lb}(F(Y)) \le \\ &\le \mathrm{ub}(F(A)) - \mathrm{lb}(F(A)) + \mathrm{ub}(F(Y)) - \mathrm{lb}(F(Y)) = \\ &= \mathrm{w}(F(A)) + \mathrm{w}(F(Y)). \end{aligned} \tag{33}$$

Y tetszőleges választása és (30), illetve (33) miatt tehát

$$\begin{aligned} &\mathrm{ub}(F(A)) - \min_{(Y,y)\in L\cup\{A\}} \mathrm{lb}(F(Y)) \le \\ &\le \mathrm{ub}(F(A)) - \mathrm{lb}(F(Y)) < \mathrm{w}(F(Y)) + \varepsilon. \end{aligned} \tag{34}$$

Másrészt feltettük, hogy f Lipschitz folytonos, így létezik egy olyan K konstans, hogy $|f(x_1)-f(x_2)| \le K\|x_1-x_2\|$, tetszőleges $x_1,x_2\in X$ értékekre. Legyen most $x_1,x_2\in Y$ olyan, hogy minden $x\in Y$-ra $f(x_1)\le f(x)\le f(x_2)$ teljesüljön. Ekkor, a maximum normát alkalmazva nyilván $\|x_1-x_2\| \le \mathrm{w}(Y)$. A fenti jelölések mellett igaz továbbá, hogy $\mathrm{w}(f(Y)) = |f(x_1)-f(x_2)|$. A két összefüggésből, azaz f Lipschitz-folytonosságából következik tehát, hogy tetszőleges $Y\in\mathbf{I}(X)$-re

$$\mathrm{w}(f(Y)) \le K\mathrm{w}(Y). \tag{35}$$

Rendezzük át F α–konvergenciájának (6) egyenlőtlenségét $\mathrm{w}(F(Y))$-ra. Hansen intervallum-kiválasztási szabálya (3. és 5. iterációs lépés) miatt minden tárolt Y intervallumra $\mathrm{w}(Y)\le\mathrm{w}(A)$ teljesül. Behelyettesítve a $\mathrm{w}(A)\le 2^{-l}\mathrm{w}(X)$ összefüggést a 11. Definícióból, majd ebbe a (35) egyenlőtlenséget, végül ismét a 11. Definícióból adódó $\mathrm{w}(Y)\le 2^{-l}\mathrm{w}(X)$ összefüggést, kapjuk, hogy

$$\begin{aligned} &\mathrm{w}(F(Y)) \le c\mathrm{w}^{\alpha}(Y) + \mathrm{w}(f(Y)) \le c\mathrm{w}^{\alpha}(A) + \mathrm{w}(f(Y)) \le \\ &\le c2^{-\alpha l}\mathrm{w}^{\alpha}(X) + K\mathrm{w}(Y) \le c2^{-\alpha l}\mathrm{w}^{\alpha}(X) + K2^{-l}\mathrm{w}(X). \end{aligned} \tag{36}$$

Visszahelyettesítve (36)-et (34)-be kapjuk a tétel állítását. □

A 15. Tétel szerint Hansen algoritmusára alkalmazva a (30) megállási feltételt, a megállás pillanatában elfoglalt iterációs szinttől függően felső korlát adható az optimum befoglalásának minőségére. A következő lemma [21] állítása megmutatja, hogy ez a kifejezés éppen a listaelemek alapján megadható, a globális optimum értékére vonatkozó befoglalás szélessége.

6. Lemma. Alkalmazzuk a 4. Algoritmus jelöléseit, és legyen A az 5. lépésben kiválasztott aktuális intervallum. Ekkor igaz, hogy

$$\min_{(Y,y)\in L\cup\{A\}} y \le \mathrm{lb}(f(X)) \le \mathrm{ub}(F(A)). \tag{37}$$

Bizonyítás. Tekintsük először (37) második egyenlőtlenségét. Belátjuk, hogy ez tetszőleges $Y\in\mathbf{I}(X)$-re igaz, így speciálisan az A intervallumra is. Legyen ugyanis F befoglalófüggvénye f-nek. Ekkor definíció szerint $x\in Y$-ból következik, hogy $f(x)\in F(Y)$, tehát $f(x)\le\mathrm{ub}(F(Y))$. De $\mathrm{lb}(f(X))\le f(x)$, amiből következik a (37) második egyenlőtlensége.

Legyen most y a listabeli elemeken számított befoglalófüggvény-értékek alsó korlátjának minimuma. Ebből következik, hogy $\exists Y\in\mathbf{I}(X)$, melyre $\mathrm{lb}(F(Y))=y$, és tetszőleges Z, listabeli intervallumra $y\le\mathrm{lb}(F(Z))$. Válasszuk most Z-t úgy, hogy az tartalmazzon globális minimumhelyet. Ezt mindig megtehetjük, mivel a listaelemek uniója az összes optimumhelyet tartalmazza. Ekkor mivel F befoglalófüggvény, valamint Z választása miatt $\mathrm{lb}(F(Z))\le\mathrm{lb}(f(Z))=\mathrm{lb}(f(X))$, amiből következik (37) első egyenlőtlensége. □

Így a 15. Tétel az eredményként kapott globális befoglalás pontosságára ad felső korlátot. Ezt az eredményt a következőképpen fogalmazhatjuk meg.

2. Következmény. A 4. Algoritmusban megállási feltételként alkalmazva a (30) összefüggést, a listán maradt intervallumok alapján a globális optimum értékére adható befoglalás hibája nem nagyobb, mint

$$c2^{-\alpha l}\,\mathrm{w}^{\alpha}(X) + K2^{-l}\,\mathrm{w}(X) + \varepsilon,$$

ahol a 15. Tétel feltételeit és jelöléseit használtuk.

Amennyiben pontszerű becslést kell adnunk a globális optimum értékére, úgy nyilván a kapott intervallum középpontját választva, az így kapott becslés hibája a befoglalás szélességének felénél nem lesz nagyobb.

A 2. Következmény becslésekor l nyilván nem független ε értékétől, azaz a célfüggvény bizonyos tulajdonságainak ismeretében előre megjósolható, hogy ε-tól függően melyik iterációs szinten áll meg az eljárás. Ezt azonban csak akkor tehetjük meg, ha előre tudjuk, hogy X minden olyan Y részintervallumán, mely az algoritmus végrehajtása során egyáltalán előállhat a felezések eredményeként, $\mathrm{w}(F(Y))$ értékének korlátozása $\mathrm{w}(Y)$-ra milyen korlátot eredményez.

$w(f(Y))\leq w(F(Y))$ miatt azonban ez képletesen azt a feltételt valósítaná meg, hogy a célfüggvényünk ezeken a részintervallumokon „nem túl lapos", azaz a vizsgált értelmezési tartomány szélességét egy bizonyos konstans fölé növelve szükségképpen az értékkészlet szélessége is egy konstans fölé nő. Ilyen feltételt megadni a célfüggvényre általában nem célszerű. Ha azonban előre rögzített alsó korlátot adunk az elérendő becslés $w(F(A))$ pontosságára, akkor megadhatjuk azon iterációs szintek számát, melyeket legalább el kell végeznünk, amint azt a következő, 16. Tételben látni fogjuk. Az eredmény az $\alpha=1$ és $\alpha=2$ esetekben explicit megadható. Az előbbi a természetes kiterjesztés, míg az utóbbi a középponti formulák konvergenciájának felel meg.

16. Tétel. Ha az f célfüggvény Lipschitz-folytonos a K állandó mellett a maximum normára nézve, valamint a befoglalófüggvénye α–konvergens a c konstanssal, akkor a 4. Algoritmus (30) feltétel alapján való megállásakor az aktuális l iterációs szintre igaz, hogy

$$\delta\leq c2^{-\alpha l}w^{\alpha}(X)+K2^{-l}w(X), \tag{38}$$

amennyiben a befoglalás $w(F(A))$ pontossága nem kell, hogy jobb legyen, mint egy adott $\delta>0$ ($\delta\leq\varepsilon$) érték.

1. Ha $\alpha=1$, akkor l-re felső korlát adható a következőképpen:

$$l\leq\log_2\frac{(c+K)w(X)}{\delta}. \tag{39}$$

2. Az $\alpha=2$ esetben az l-re adható felső korlát

$$l\leq\log_2\frac{\left(\sqrt{K^2+4\delta c}+K\right)w(X)}{2\delta}. \tag{40}$$

Bizonyítás. Rendezzük át a 6. Definíció (6) egyenlőtlenségét és helyettesítsük be a 15. Tétel bizonyításabeli (35) összefüggést. Mivel az eredményül kapott képlet igaz minden $Y\in\mathbf{I}(X)$ intervallumra, így speciálisan $Y=A$-ra, az algoritmus 6. lépésének aktuális intervallumára is:

$$w(F(A))\leq cw^{\alpha}(A)+Kw(A). \tag{41}$$

Az iterációs szintek definíciójából és a Hansen eljárások intervallum kiválasztási és vágási tulajdonságaiból következik, hogy

$$w(A)=2^{-l}w(X) \tag{42}$$

Írjuk vissza a (42)-beli $w(A)$ értéket (41)-be. Ha felhasználjuk a $\delta\leq w(F(A))$ egyenlőtlenséget, akkor azt kapjuk, hogy

$$\delta\leq w(F(A))\leq c2^{-\alpha l}w^{\alpha}(X)+K2^{-l}w(X). \tag{43}$$

Ha $\alpha=1$, akkor (43) átírható

$$\delta \le (c+K)2^{-l}\mathrm{w}(X)$$

alakra, ami szolgáltatja (39)-et.

Legyen most $\alpha=2$. Ekkor (43) a következő alakot ölti:

$$\delta \le \frac{c\mathrm{w}^2(X)}{2^{2l}} + \frac{K\mathrm{w}(X)}{2^l}. \tag{44}$$

Átrendezve (44)-et egy 2^l-ben kvadratikus egyenlőtlenséget kapunk:

$$\delta(2^l)^2 - K\mathrm{w}(X)(2^l) - c\mathrm{w}^2(X) \le 0. \tag{45}$$

Ha megoldjuk (45)-öt 2^l-re, akkor az eredmény

$$\frac{\mathrm{w}(X)}{2\delta}\left(K - \sqrt{K^2+4\delta c}\right) \le 2^l \le \frac{\mathrm{w}(X)}{2\delta}\left(K + \sqrt{K^2+4\delta c}\right), \tag{46}$$

mivel δ pozitív. (46) első egyenlőtlensége egy triviális negatív alsó korlátot szolgáltat, míg a második a tételbeli (40)-et. □

Hansen algoritmusa esetén is megfogalmazható egy, a (30)-hoz hasonló feltétel, mely garantálja, hogy a megálláskor a befoglalás hibája nem halad meg egy előre megadott ε értéket. Ez a feltétel a következő:

$$\min_{(Y,y)\in L\cup\{A\}} \mathrm{ub}(F(Y)) - \min_{(Y,y)\in L\cup\{A\}} \mathrm{lb}(F(Y)) < \varepsilon. \tag{47}$$

A feltétel baloldalán álló két tag nyilván felső, illetve alsó korlát a globális optimumra nézve, így ezen értékek alapján ε pontosságú befoglalás adható az optimum értékére. Megvalósításához minden iterációban frissíteni kell mindkét tag értékét.

Tegyük most fel, hogy a 3. Algoritmus 1. és 5. lépését helyettesítjük a 4. Algoritmus megfelelő lépéseivel, így a Moore-Skelboe eljárásban is minden iteráció 5. lépésében kiszámítjuk, illetve frissítjük az $\tilde{f}$ értékét. Amennyiben a 4. lépést is átvesszük, az így keletkező eljárás az irodalomban *Ichida-Fujii algoritmus* [39] néven szerepel. Ekkor a (30)-hoz és a (47)-hez hasonló, szintén a befoglalás hibáját korlátozó megállási feltétel adható a következő módon:

$$\tilde{f} - \mathrm{lb}(F(A)) < \varepsilon. \tag{48}$$

Ez a feltétel a (30)-hoz hasonlóan szintén a Moore-Skelboe algoritmussal, illetve annak intervallum-kiválasztási szabályával együtt alkalmazva biztosítja, hogy az eredményként kapott befoglalás ε pontosságú legyen. Ebben az esetben $\tilde{f}$ felső, míg $\mathrm{lb}(F(A))$ alsó korlátot ad a globális optimumra. Mivel (30) tulajdonképpen az $\mathrm{ub}(F(A)) - \mathrm{lb}(F(A)) < \varepsilon$ feltételt fogalmazza meg, a nyilvánvaló $\tilde{f} \le \mathrm{ub}(F(A))$ egyenlőtlenség miatt (48) jobb feltételt ad meg (30)-hez képest abban az értelemben, hogy azonos iterációszám mellett jobb befoglalást szolgáltat az optimum értékére. Ha feltételezzük, hogy bármely A, legalább egy globális optimumhelyet tartalmazó aktuális intervallumon a vizsgált függvényosztálybeli célfüggvény értékei

azonos valószínűséggel esnek az $F(A)$ intervallum tetszőleges pontjába, akkor $\tilde{f}$ várható értéke ezen az intervallumon $\mathrm{m}(F(A))$. Átlagos esetben kijelenthetjük tehát:

5. Megjegyzés. A Moore-Skelboe algoritmusban a (30) vagy a (48) megállási feltételeket választva az $\varepsilon=\varepsilon_1$, illetve $\varepsilon=\varepsilon_2$ értékekkel, átlagos esetben $\varepsilon_2=\varepsilon_1/2$ választás mellett a megálláshoz a kétféle eljárás azonos számú iterációt vesz igénybe.

Hansen algoritmusára ez így nem feltétlenül igaz, mivel a (31) baloldalán álló különbség átlagos esetben nem csökken a felére. Igaz azonban a következő állítás, mely a 15. Tétel egyenes következménye.

3. Következmény. Alkalmazzuk a 15. Tétel feltételeit és jelölésrendszerét. Ekkor átlagos esetben teljesül, hogy

$$\tilde{f} - \min_{(Y,y)\in L\cup\{A\}} y < c2^{-\alpha l}\,\mathrm{w}^{\alpha}(X) + K2^{-l}\,\mathrm{w}(X) + \frac{\varepsilon}{2}. \tag{49}$$

(49) nyilván jóval gyengébb eredmény, mint a Moore-Skelboe algoritmusra az 5. Megjegyzésben megfogalmazott állítás, ráadásul ez is csupán statisztikus értelemben teljesül.

Általánosságban kimondható, hogy a befoglalás pontosságára vonatkozó megállási feltételek a Moore-Skelboe algoritmusra alkalmazhatók hatékonyan.

3.5.2. Az optimumhely befoglalása

A megállási feltételek másik nagy osztályát alkotják azok, melyek az optimumhelyek befoglalása pontosságára vonatkoznak. Attól függően, hogy minden globális optimumhelyet meg kívánunk-e határozni, vagy megelégszünk egyetlen optimumhely előre meghatározott pontossággal való befoglalásával, itt is többféle módszer létezik. A megállási feltételek ezen osztályának egy része Moore és Skelboe algoritmusára nem működik, sőt, bizonyítható [55], hogy arra át sem vihető.

Az egyik ilyen megállási feltétel, mely a befoglalást eredményező, a listán maradt összes elem befoglalásának együttes pontosságát szabja meg, a következő [34, 35]:

$$\sum_{(Y,y)\in L\cup\{A\}} \mathrm{w}(Y) < \varepsilon. \tag{50}$$

Ez a feltétel a listaelemek V össztérfogatára is felső korlátot szolgáltat:

$$V \le \sum_{(Y,y)\in L\cup\{A\}} \mathrm{w}^n(Y) \le \left(\sum_{(Y,y)\in L\cup\{A\}} \mathrm{w}(Y)\right)^n < \varepsilon^n.$$

Az (50) megállási feltétel felhasználhatósága eleve feltételezi valamilyen, a lista hosszát csökkentő gyorsító eljárás létezését, ellenkező esetben $\Sigma_{(Y,y)\in L\cup\{A\}} w(Y)=w(X)$ nyilván állandó. Másrészt (50) csak akkor teljesülhet, ha a globális optimumhelyek halmazára igaz, hogy izolált részhalmazainak intervallumburkai szélességének összege kisebb, mint ε. Ez utóbbi feltétel gyengíthető az egyes részhalmazok intervallumburkaira adott ε korlátozásra, ha a következő megállási feltételt alkalmazzuk:

$$\forall (Y,y)\in L\cup\{A\} \quad w(Y)<\varepsilon. \tag{51}$$

Mivel Hansen algoritmusánál mindig a listán található legszélesebb intervallum kerül továbbdarabolásra, ezért ebben az esetben (51) ekvivalens a következő megállási feltétellel:

$$w(A)<\varepsilon, \tag{52}$$

ahol A szokás szerint az aktuális intervallumot jelöli. Az (52) megállási feltételt használva a listán található (Y,y) elemek közül azok, melyekre y minimális, $F(Y)$ a 6. Lemma miatt nyilván tartalmaz globális minimumértéket. Ily módon az optimum értékére is becslést adhatunk a következőképpen [21]:

17. Tétel. Alkalmazzuk az (52) megállási feltételt a 4. Algoritmusra, tegyük fel továbbá, hogy az f célfüggvény Lipschitz-folytonos, és F befoglalófüggvénye α-konvergens. Igaz ekkor, hogy

$$w(F(Y))<c\varepsilon^{\alpha}+K\varepsilon, \tag{53}$$

ahol Y a tetszőleges tárolt elem (listaelem, vagy maga az aktuális intervallum), c a 6. Definícióból származó együttható, K pedig a célfüggvény Lipschitz-konstansa a maximum normára nézve.

Bizonyítás: Az F α–konvergenciájából származó (6) összefüggést átrendezve kapjuk, hogy

$$w(F(Y))\le c w^{\alpha}(Y)+w(f(Y)). \tag{54}$$

Helyettesítsük be ebbe az f Lipschitz-folytonosságából adódó (35) összefüggést:

$$w(F(Y))\le c w^{\alpha}(Y)+K w(Y).$$

Mivel a Hansen eljárások esetében (52)-ből következik (51), azaz

$$w(A)<\varepsilon \Rightarrow w(Y)<\varepsilon, \tag{55}$$

így folytatva a (54) egyenlőtlenséget, és behelyettesítve (51)-at adódik az állítás

$$c w^{\alpha}(Y)+K w(Y)<c\varepsilon^{\alpha}+K\varepsilon. \qquad \square$$

4. Következmény. A 4. Algoritmusra alkalmazva a (52) megállási feltételt, a globális minimum értékére adható befoglalás hibája kisebb, mint

$$c\varepsilon^{\alpha}+K\varepsilon. \tag{56}$$

Az (50) és (51) megállási feltételek nem célszerűk a Moore-Skelboe algoritmushoz. Ezzel szemben előfordulhat, hogy csupán egyetlen globális minimumhely megfelelő korlátok közötti meghatározására van szükség, illetve ha az nem egyetlen pontból áll, akkor annak valamely tetszőleges pontjára. Ebben az esetben (52) Moore és Skelboe módszerében is alkalmazható. A globális minimum értékére adható befoglalás ilyenkor pontosan megegyezik a Hansen-féle algoritmus esetében a 17. Tételben kimondottakkal, azzal a különbséggel, hogy ebben az esetben a Moore-Skelboe-féle intervallum-kiválasztási szabály miatt a legkisebb befoglalófüggvény alsó korláttal rendelkező listaelem pontosan az aktuális intervallum. Így a bizonyításbeli (55) következtetés, mely a Moore-Skelboe eljárásnál téves lenne, szükségtelenné válik, és igazolást nyer a következő állítás:

5. Következmény. A 3. Algoritmusra alkalmazva a (52) megállási feltételt teljesül, hogy

$$\mathrm{w}(F(A))<c\varepsilon^{\alpha}+K\varepsilon, \tag{57}$$

ahol f célfüggvény Lipschitz-folytonos a K Lipschitz-konstanssal, befoglalófüggvénye, F α-konvergens, c pedig a 6. Definícióból származó együttható.

Az 5. Következmény állítása nagyon hasonlít a 17. Tételben kimondottakra, lényeges különbség azonban, hogy az állítás ebben az esetben Moore-Skelboe eljárásra vonatkozik, amely nem uniform darabolást valósít meg, a megállási feltétel mégis az aktuális intervallumnak a szélességére vonatkozó, nem pedig a befoglalására. Ennek ellenére igaz, hogy (57) valóban az optimum érték befoglalására ad felső korlátot, mivel a Moore-Skelboe eljárások esetében mindig teljesül az $\mathrm{lb}(f(X))\in F(A)$ tartalmazás.

Az egyes intervallum-kiválasztási szabályokat a velük leghatékonyabban működő megállási feltételekkel párosítva bonyolult gyorsító eljárások nélkül is hatékonyan megoldhatóvá válnak olyan optimalizálási problémák is, melyek egyébként nehezen kezelhetők. Példa erre a műszaki kémia területéről ismert szeparációs rendszerek optimális tervezése, melyek esetében elméleti szempontból is fontos a következőkben ismertetésre kerülő két probléma.

3.6. Egy gyakorlati felhasználás

3.6.1. A szeparációs probléma

A műszaki kémia területén a szeparációs rendszerek minden olyan folyamat során előkerülnek, melyeknél keverékek vagy vegyületek valamilyen összetevőkre való bontására van szükség. Ez az egyszerű kiszitálástól kezdve a közismert desztilláción át egészen bonyolult vegyi eljárásokig számos folyamatot magába foglal. A szeparáció a vegyiparban leggyakrabban összetett eljárások részeként fordul elő, más esetekben azonban a legfontosabb folyamat maga a szeparáció. Az utóbbihoz tartozó terület a gazdaságilag még napjainkban is igen fontos petrolkémiai ágazat.

A gyakorlati problémától függően számos különböző megoldás létezhet egy szeparációs hálózat tervezését illetően. Lényeges különbségek jelentkeznek a hálózatot alkotó egyes szeparátorok tekintetében is. Előfordulhat, hogy a szeparáció során az összetevők élesen nem választhatók el egymástól, vagy külön eljárásokra van szükség a továbbontáshoz, melyeknél eltérő fizikai feltételek szükségeltetnek. A következőkben csupán az egyszerű, éles szeparációval foglalkozunk, annak is a speciális, szekvenciális esetével, mely a következőképpen írható le. Jelöljük a bemeneti anyagáramok azon összetevőit, melyeket a szeparáció során tovább nem bontunk, M_1, $M_2,\ldots$, M_q-val. A szeparációs hálózatot olyan S^i (i=1, 2,..., q-1) típusú szeparátorokból építjük fel, melyek tetszőleges, a fenti összetevőkből álló befolyó anyagáramot az i-dik és i+1-dik összetevő „között" élesen szeparálják. Más szóval egy S^i szeparátor az $M_1M_2\ldots M_q$ anyagáramot olyan $M_1^1M_2^1\ldots M_q^1$ és $M_1^2M_2^2\ldots M_q^2$ kimenő anyagáramokra bontja, melyekre teljesül, hogy

$$M_j^1=\begin{cases}M_j & \text{ha } j\le i,\\ 0 & \text{különben,}\end{cases} \quad \text{és} \quad M_j^2=\begin{cases}M_j & \text{ha } j>i,\\ 0 & \text{különben.}\end{cases}$$

Feltesszük, hogy ahhoz, hogy valamely $f_1,\ldots, f_f$ bemenő anyagáramokból a megfelelő $p_1,\ldots, p_p$ kimenő anyagáramokat megkapjuk, rendelkezésünkre áll minden fenti S^i szeparátorból tetszőleges számú darab, valamint keverő egységek, melyek egyes belső anyagáramok egyesítésére szolgálnak, illetve szétosztó egységek, melyek egy anyagáramot tetszőleges arányban több áramra osztanak úgy, hogy a keletkezett anyagáramok összetétele megegyezik a beléjük táplált anyag összetételével. Nyilvánvaló, hogy tetszőleges szeparációs probléma megoldható ezen a módon. A legegyszerűbb megoldás, hogy az $f_1,\ldots, f_f$ áramokat összekeverjük, majd a kapott áramot összetevőire bontjuk, és azokból szétosztó egységek alkalmazását követően kikeverjük a megfelelő $p_1,\ldots, p_p$ anyagokat. A szeparáci-

ónak azonban jelentős költségvonzata van, melyet figyelembe kell venni. A költségfüggvény ebben az esetben az egész hálózatra a következőképpen írható fel:

$$C = \sum_{i=1}^{m} (D_i \cdot S_i)^d , \tag{58}$$

ahol $D_i>0$ és $0<d<1$ állandók, S_i ($i=1,\ldots, m$) a hálózatban található i-dik szeparátoron átfolyó anyag mennyisége (függetlenül attól, hogy a szeparátor mely összetevők között szeparál), m pedig a hálózatban található szeparátorok száma [7, 31, 43].

3.6.2. A szeparációs probléma egy megoldása

A szeparációs probléma itt ismertetésre kerülő megoldása két részből tevődik össze. Az egyik egy olyan, ún. szuperstruktúra létrehozásából áll, amely egy, a kérdéses feladatot megoldó hálózat váza. A szuperstruktúra nem tartalmazza az egyes anyagáramok osztási arányait, így az átfolyó áramok nagyságát sem, másrészt feltesszük, hogy részhálózatként tartalmaz egy optimális megvalósítást eredményező hálózatot (ld. pl. [43]). Ezek után az osztási arányok mint változók értékeinek beállításával megadható az optimális hálózat.

A szeparációs feladatok megoldásában a szakértők hosszú ideig úgy gondolták, hogy a szuperstruktúra a C költségfüggvény alakjánál fogva egy adott típusú szeparátorból csupán egyet kell tartalmazzon, mivel egy adott típusú szeparátorból több példányt tartalmazó hálózat költsége csökkenthető, ha ezeket egyetlen, nagyobb kapacitású szeparátorral helyettesítjük, így a globálisan optimális hálózat minden típusú szeparátorból csak egyet alkalmaz [31]. Ez a *redundancia* feltevés. Egy másik, a szuperstruktúra egyszerűsítését célzó feltevés szerint egyetlen optimális hálózat sem tartalmazhat *recirkulációt* (anyag-visszaáramoltatást), mivel egy szeparátoron átfolyt anyagot vagy annak egy részét ugyanabba a szeparátorba visszaáramoltatva a költség növekedik. Különböző meggondolások alapján olyan ellenpéldák születtek [43], melyek mindkét feltételezést cáfolják. Ahhoz azonban, hogy ezek globális optimalitását bizonyítani lehessen, olyan módszerre volt szükség, mely globális megoldást garantál. Itt tehát a közelítően optimális megoldásnak nem volt értelme. Ez ösztönzött annak idején az intervallum-felosztási módszerek felkutatásában és felhasználásában.

3.6.3. Két numerikus példa

A következőkben ismertetésre kerülő két szeparációs feladat ellenpélda a fentebb leírt redundanciára és recirkulációra vonatkozó feltételekre [43] vonatkozóan. Mindkét példa a következő típusú szeparációs probléma megoldása: adott két be-

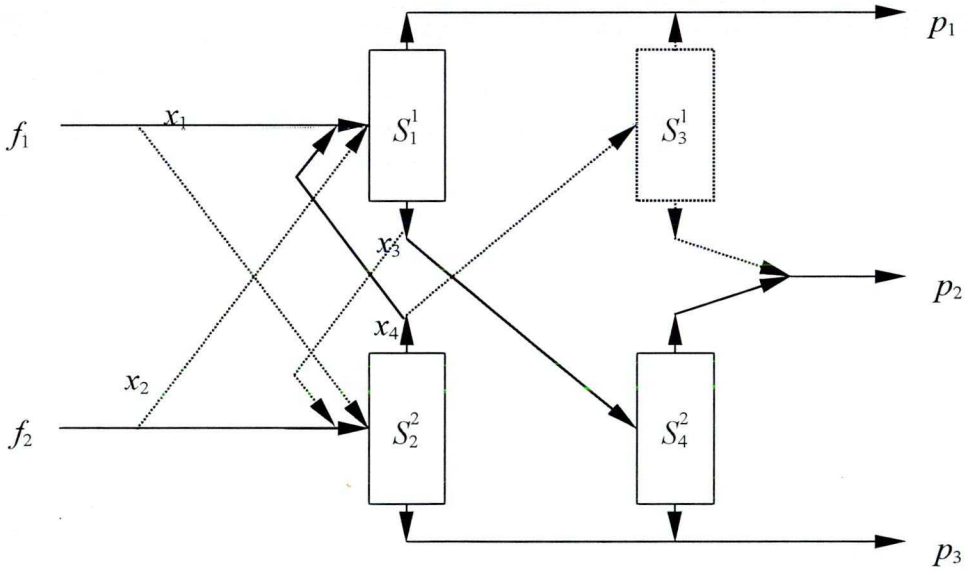
menő anyagáram f_1 és f_2, melyek A, B és C komponensekből állnak. Ezekből kell előállítani három kimenő anyagáramot, a p_1, p_2 és p_3 anyagokat. A két példa be- és kimenő anyagáramainak komponensenkénti értékeit a 4. és az 5. Táblázat tartalmazza.

	A	B	C
f_1	100	1	1
f_2	100	1	200
p_1	200	0	0
p_2	0	2	0
p_3	0	0	201

4. Táblázat. SEPNET1 be- és kimenő anyagáramai komponensenként

	A	B	C
f_1	80	9	12
f_2	3	3	90
p_1	81	0	0
p_2	2	12	0
p_3	0	0	102

5. Táblázat. SEPNET2 be- és kimenő anyagáramai komponensenként

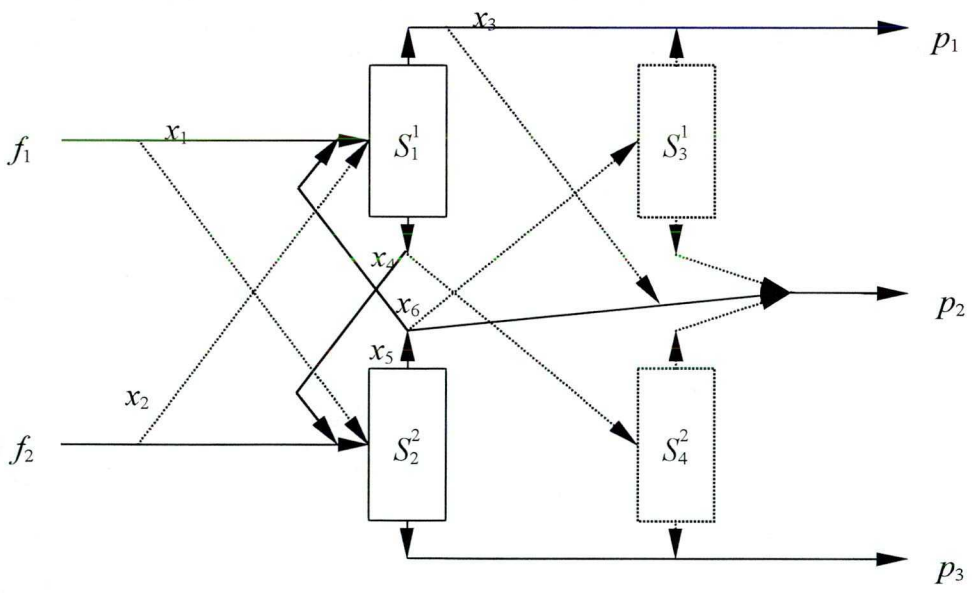


5. Ábra. A SEPNET1 szeparációs probléma szuperstruktúrája.

Az első, SEPNET1 jelű feladat szuperstruktúráját a 5. Ábra mutatja. Az egyes szeparátorok elnevezésében az alsó index egyszerű sorszám, míg a felső annak típusát jelzi. Nem jelöltem külön a szétosztó és keverő egységeket, azokat az anyagáramokat jelölő szakaszok elágazásai érzékeltetik. Az ábrán szaggatott vonal különbözteti meg azokat az áramokat és szeparátorokat, melyeket az (egyetlen) optimális megoldás nem tartalmaz. Látszik, hogy az optimális hálózatban a 2 típusú szeparátorból kettő darab is található, megdöntve ezzel a fentebb leírt feltételezést.

A probléma megoldásához elegendő kiszámítani a szétosztó egységekben előállítandó osztási arányokat, mivel ezek egyértelműen meghatározzák a feladat megoldását. Ezeket az ábrán az x_i változók jelölik úgy, hogy egy osztási pontban valamely f anyagáramból $x_i f$ folyik tovább az egyik és $(1-x_i)f$ a másik ágon $(0 \le x_i \le 1)$. Az optimalizálási probléma az egyes szeparátorokon átfolyó anyagáramok függvényében minimalizálni az (58)-ban felírt költségfüggvényt, ahol $m=4$, $D_i=1$ $(i=1,\ldots,4)$ és $d=0,6$. A feladat pontos megadása, a célfüggvény és a korlátozó feltételek megtalálhatók a függelékben.

A SEPNET2 feladat szuperstruktúrája a 6. Ábrán látható. A jelölések megegyeznek az 5. Ábránál leírtakkal. A különbség annyi, hogy ebben az esetben hármas szétosztás is előfordul. Ez az x_5 és x_6 változókra vonatkozik, és annyit változtat a helyzeten, hogy az adott csomópontba befolyó valamely f anyagmennyiség az $x_5 f$, $x_6 f$ és $(1-x_5-x_6)f$ áramokra bomlik. A SEPNET2 feladat pontos leírása szintén megtalálható a függelékben. A 6. Ábrán jól látszik, hogy az optimális hálózatban a B komponens egy része folyamatosan cirkulál. Valóban, az S_1^1 szeparátor alsó kimenetén keresztül az S_2^2 szeparátorba jutva és annak felső kimenetén keresztül ismét az S_1^1 szeparátor bemenetére adva egy anyagáram ebbe a körbe visszakerül, megcáfolva ezzel a recirkuláció kizárhatóságának feltételét.



6. Ábra. A SEPNET2 szeparációs probléma szuperstruktúrája.

A fentebb megadott két globális optimalizálási probléma megoldásához azok elvi jelentősége miatt olyan módszer szükséges, mely bizonyíthatóan optimális megoldást talál. Ezekre a feladatokra garantált pontosságú és bizonyíthatóan helyes megoldást először intervallum-felosztási módszerek kombinációjával sikerült adni [6, 7, 8]. A megoldás során egy hibrid eljárást használtam, mely két lépcsőből áll. Az első lépcsőben a 3. Algoritmust alkalmaztam a (30) megállási feltétellel a glo-

bális minimum értékének bizonyos pontosságú befoglalására. Ezután az így kapott felső korlátot a 4. Algoritmus 1. lépésében $\tilde{f}$ inicializálásához felhasználva, automatikusan végrehajtottam az (52) megállási feltétellel ellátott Hansen algoritmust. A megvalósításhoz saját, C++ alatt fejlesztett intervallum aritmetikai csomagot alkalmaztam. A két feladaton kapott eredmények a 6. Táblázatban találhatók. Az önállóan elkészített C++ függvénykönyvtár magában foglalja az összes elemi függvény intervallumos és szimpla és dupla pontosságú valós/intervallumos kevert változatát. A megvalósítás során az intervallumokat objektumként valósítottam meg, az elemi műveleteket és függvényeket pedig az „operator overloading" segítségével implementáltam.

Ugyanakkor a más, nem intervallumos algoritmusokkal való próbálkozások során a klasszikus optimalizálási módszerek a nagy számú lokális minimum miatt nem voltak képesek elfogadható eredményt produkálni. Végül sikerült az intervallumos módszerrel való összehasonlításhoz egy klaszterező sztochasztikus eljárással [23] is megoldani mindkét feladatot. A két feladaton a sztochasztikus eljárás futtatásai által szolgáltatott eredmények a 7. és 8. Táblázatokban találhatók. Látható, hogy sem az optimum helyét, sem pedig annak értékét nem volt képes pontosan meghatározni. A megoldások hibáját jól tükrözi, hogy az optimum értékére kapott eredmények még az intervallumos módszer által megadott elég tág befoglaló intervallumokon is kívül esnek.

Feladat jele:		SEPNET1	SEPNET2
Függvényhívások száma:	Első lépcső	3269	19423
	Második lépcső	2384	6746
$\tilde{f}$ kezdeti értéke a második lépcsőben:		56,8714	34,5429
Optimumhely befoglalása:		x_1=[0,9999, 1], x_2= [0, 1,78e-7], x_3= [0, 0], x_4= [0,9999, 1].	x_1=[0,9999, 1], x_2=[0, 7,62e-6], x_3=[1, 1], x_4=[0,9999, 1], x_5=[0,3333, 0,3334], x_6=[0, 3,814e-6]).
Optimumérték befoglalása:		[56,8713, 56,8714]	[34,5429, 34,5430]

6. Táblázat. A SEPNET1 és SEPNET2 szeparációs probléma kétlépcsős hibrid intervallum-felező globális optimalizáló módszerrel való megoldásának eredménye.

Lokális minimum sorszáma	x_1	x_2	x_3	x_4	$f(x)$
1	0,99999	8,551e-9	0	0,99999	56,8715
2	0,94381	4,44e-11	0	0,99999	57,1950
3	0,73212	2,97e-11	0	0,99999	58,3897
4	1	7,78e-10	0	0,96981	58,5924
5	1	2,39e-11	0	0,86762	60,5878
6	1	1	0	0,22640	60,8134
7	0,68203	6,181e-9	0	0,89439	62,4715
8	0,81546	4,796e-9	0	0,79238	62,8698
9	0,80633	5,917e-9	0	0,65647	64,0435
10	0,90103	6,569e-9	0	0,51874	64,0447
11	0,94809	4,526e-9	0	0,38760	64,1371
12	0,62187	2,160e-9	0	0,67845	65,2263
13	0,79207	0	0	0,37366	65,2377

7. Táblázat. A SEPNET1 szeparációs probléma klaszterező sztochasztikus globális optimalizáló módszerrel való megoldásának eredménye. A szükséges függvényhívások száma 4876.

Lok. min. ssz.	x_1	x_2	x_3	x_4	x_5	x_6	$f(x)$
1	0,99999	3,137e-8	1	0,99999	0,33333	2,756e-8	34,5432
2	0,99999	3,418e-8	1	0,99999	0,39161	6,569e-9	34,8682
3	0,99999	0,29961	1	1	0,06994	1,824e-7	36,0358
4	0,99999	0,20634	1	0,99999	0,33728	2,22e-15	36,2553
5	1	1,109e-9	1	0,74917	0,33333	4,069e-9	36,7088
6	0,99999	6,409e-9	1	2,64e-8	0,33333	8,767e-9	37,9873
7	0,99999	6,656e-9	1	2,21e-8	0,64190	4,572e-8	38,3207
8	1	0,333333	1	0,99999	3,0e-10	0,307474	38,5326
9	0,99999	7,227e-9	1	0,33997	0,77028	7,304e-9	38,7660
10	1	0,255642	1	0,08632	0,59043	4,538e-9	41,3403
11	0,99999	0,518744	1	4,62e-9	0,04110	6,079e-9	42,9793
12	0,98575	0,732907	1	1,11e-9	0,45013	0,047801	44,2313
13	0,86645	0,908857	1	0,06354	0,85968	3,308e-9	46,1922
14	0,64823	0,928572	1	0,58503	0,98813	4,106e-9	51,6088

8. Táblázat. A SEPNET2 szeparációs probléma klaszterező sztochasztikus globális optimalizáló módszerrel való megoldásának eredménye. A szükséges függvényhívások száma 21686.

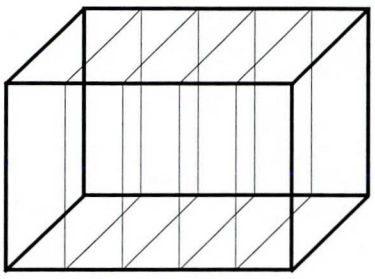
A megoldások pontosságát illetően az intervallumos kétlépcsős módszer esetében az eljárás tetszőlegesen kicsiny befoglalási szélesség eléréséig folytatható lett volna. Emellett a módszerek felépítésénél fogva a kapott intervallumok garantáltan

tartalmaznak globális optimumhelyet, illetve optimumértéket, míg egyéb módszernél a szokásos konvergencia-feltételek és hibaszámítás adhat becslést az eredmény pontosságára és megbízhatóságára vonatkozóan. A hibrid eljárás második lépcsőjének végén a listán csupán egy-egy elem maradt, ami azt jelzi, hogy a megadott pontosságot figyelembe véve az optimális megoldás helye mindkét esetben egyértelmű, azaz a megadott intervallumokon kívül más optimumhely nem létezik.

4. Általános felosztó eljárások

Bár az eddigiekben a 2. Algoritmus egyes lépéseit vizsgálva többnyire nem szorítottuk meg az általános, tetszőleges s darabra vágás esetét, a jelenleg a szakirodalomból ismert eljárások szinte mindegyike az intervallumokat két részre vágó módszereket vizsgál, azon belül is az intervallumfelező eljárásokat. Az első, intervallumokat alkalmazó optimalizálási feladatok megoldására felhasznált módszerek valószínűleg olyan eljárások voltak, melyek a lehetséges megoldások halmazának azon részét, melyből nem volt kizárható globális optimumhely létezése, megfelelően kicsiny intervallumokra vágták. A kérdés tehát felmerül, hogy a korszerű intervallum-felosztási módszerek esetében is alkalmazható-e ez az ötlet; az eljárás során darabolásra kerülő intervallumot vágjuk több részre, így finomabb felbontást kapunk, valamint a 2.2. fejezetből ismert befogalaló függvények α–konvergenciája miatt ezeken a befoglalás is pontosabbá válik. Ez egyrészt egy bizonyos határon túl várhatóan nem fog jobb eredményt szolgáltatni, mivel a továbbdarabolás folyamatos felezések útján úgyis elérhető, ráadásul minden felezés után az új információ birtokában további részintervallumok zárhatók ki a további vizsgálatból. Másrészt azonban ez a többletinformáció egy-egy felező lépést tekintve általában nem túl jelentős, így az egyes iterációs ciklusokhoz fűződő szükséges függvényhívások és adminisztratív lépések — ellenőrzés, listakezelés, megállási feltételek vizsgálata — szükségtelenül gyakran kerülnek végrehajtásra. A sejtés az, hogy megadható s-nek olyan, 2-től eltérő értéke, melyre a többi algoritmikus lépéstől függően esetleg az intervallum-felosztási eljárások gyorsíthatók.

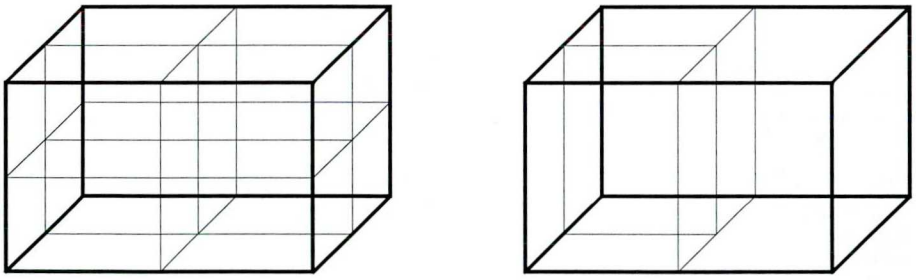
Már a két részre vágás esetében is felmerült néhány módosítási lehetőség a vágás egyes paramétereivel kapcsolatban. Ezek közül a keletkező két részintervallum aránya az egyetlen dolog, melyet a célfüggvény és befoglalófüggvényének részletes ismeretének hiányában kénytelenek vagyunk úgy rögzíteni, hogy átlagos esetben megfelelően működjenek az algoritmusok. Ezért ebben az esetben az intervallumfelező eljárások terjedtek el. Több darabra való vágáskor azonban (több dimenzióban) további kérdések merülhetnek fel. Az egyik, a felezéstől átvehető ötlet, hogy s darab egybevágó részintervallumra bontjuk az aktuális intervallumot. Ennek egyik megvalósítási formája, hogy kiválasztunk egy koordináta-irányt, és arra merőlegesen s darabra „szeleteljük" az intervallumot (7. Ábra).



7. Ábra. 5 darabra való szeletelés 3 dimenzióban.

Az ilyen eljárásokat a továbbiakban *szeletelő eljárás*oknak fogjuk nevezni.

Egy másik módszert eredményez, ha minden egyes koordináta-irányra merőlegesen $\sqrt[n]{s}$ darabra ($\sqrt[n]{s} \geq 2$ egész) „szeljük" az intervallumot, így az végül s egybevágó darabra esik szét. Ugyanúgy, mint a két darabra való vágásnál, a vágási arányokról itt sem tudunk előre jobbat mondani, minthogy ezeket a darabolásokat is egyenlő részekre szeléssel célszerű elvégeznünk, egybevágó részintervallumokat kapva a darabolás elvégzése után. Speciális esetben minden irányra felezést végezve egyetlen iteráció során 2^n darab részintervallum keletkezik. Ezekkel az eljárásokkal már többen is foglalkoztak (pl. [3, 29, 36]), és a teszteredmények tanulsága szerint a 3.1. fejezet B szabálya által meghatározott irányra merőleges felezések közül három egymást követő vágás hozta a legjobb eredményeket [36], azaz amikor az aktuális intervallumot egyetlen iteráción belül $2^3=8$ darabra vágták a többszörös szelés segítségével (8.a Ábra). Hasonlóképpen az egyszerű felezéshez képest javulás tapasztalható az A szabály alkalmazásakor is, amennyiben többszörös szelést alkalmazunk [29]. Ennél kicsit általánosabb módszer az, ha ezt a szelést az eljárás közben megszakítjuk, amint elértük a kívánt számú részintervallumot. A vágási irányok ekkor is a 3.1. fejezetben leírtak szerint választhatók meg. Az így kapott módszerek az úgynevezett *többszörös vágás*t alkalmazó algoritmusok (8.b Ábra).



a b

8. Ábra. a: Többszörös szelés minden koordináta-irányra, b: többszörös vágás három darabra.

A következőkben a szeletelő és a többszörös vágást megvalósító algoritmusok tulajdonságaival foglalkozunk.

4.1. Szeletelő eljárások

A következőkben a Hansen algoritmust módosítjuk oly módon, hogy szeletelő eljárást alkalmazzon az aktuális intervallum továbbdarabolására. Mivel a 3.4. fejezetben említett okok miatt a legrosszabb eset vizsgálatakor mindig feltehetjük, hogy a gyorsító eljárások nem törölnek listaelemet, így ezt a lépést a módszerből az elméleti vizsgálatok első körében eleve elhagyjuk. Ennek érdekében definiáljuk a következő algoritmust:

5. Algoritmus. Hansen intervallum-kiválasztási szabályát alkalmazó intervallumfelosztási algoritmus szeletelő eljárással.

1. Legyen L egy üres lista, $A:=X$ az aktuális intervallum! Állítsuk a k iteráció-számlálót $k:=1$ értékre!
2. Daraboljuk az A intervallumot s darab A_i $(i=1,2,\ldots,s)$ egybevágó részintervallumra oly módon, hogy azt a leghosszabb élére felvett ekvidisztáns beosztás mentén arra merőleges hipersíkokkal felszeleteljük!
3. Helyezzük a lista végére az újonnan keletkezett intervallumokat, azaz legyen $L:=L\cup\{A_1\}\cup\{A_2\}\cup\ldots\cup\{A_s\}$!
4. —
5. Válasszuk ki a listáról az $A\in L$ legelső intervallumot, és emeljük le onnan, azaz legyen $L:=L\setminus\{A\}$!
6. Ha a megállási feltétel nem teljesül, növeljük az iteráció-számlálót, $k:=k+1$, és térjünk vissza a 2. lépéshez!
7. Nyomtassuk ki az eredményt, és STOP!

A szeletelő eljárás egyes iterációs ciklusainak vizsgálata érdekében ebben az esetben is bevezetjük az iterációs szintek 11. Definíciójában leírt fogalmát, ezúttal úgy, hogy az általánosságban s tetszőleges értékére értelmezhető legyen. A szeletelő eljárásoknak az $s=2$ speciális esete nyilván éppen az intervallumfelező módszereket adja. A továbbiakban az általánosság megsértése nélkül feltehetjük, hogy tetszőleges listabeli intervallum egyik éle sem legalább s-szer hosszabb bármely másik élnél. Ez teljesül, ha kikötjük, hogy az X kiindulási intervallum *s-kvázi hiperkocka*, azaz ha X_i jelöli az X intervallum i-dik vetületét, tehát $X=(X_1,\ldots,X_n)$, akkor $w(X_i)<s\,w(X_j)$ $(\forall i,j\in\{1,\ldots,n\})$, mivel a darabolási irány megválasztása később már ezt a tulajdonságot nem rontja el. Ezek után értelmes a következő definíció.

12. Definíció. Azt mondjuk, hogy az 5. Algoritmus k-dik iterációs ciklusa az *l-dik (iterációs) szinten* van, ha az L listán található elemek szélességének w_{max} maximumára teljesül, hogy

$$s^{-(l+1)}\mathrm{w}(X)<\mathrm{w}_{\max}\leq s^{-l}\mathrm{w}(X). \tag{59}$$

Az iterációs szintek ezen kibővített értelemben vett fogalmára [22] ugyanúgy érvényes a 4. Megjegyzés, tehát az iterációs szintek fogalma osztályozást határoz meg az iterációs ciklusok halmazán. Hasonlóan a speciális esethez, a következőkben megadjuk az s-szeletelő eljárások iterációs szintjeinek tulajdonságait [22].

7. Lemma. Az 5. Algoritmus l-dik iterációs szintjére érve a tárolt elemek (az L lista elemei és az A aktuális intervallum) száma $N_l=s^{nl}$.

Bizonyítás. Egy új, l-dik iterációs szintre érve a listán levő minden elem egybevágó, így szélességük azonos, jelöljük $\mathrm{w}_{\max}$-szal. Akkor hagyjuk el az l-dik iterációs szintet, ha az utolsó $\mathrm{w}_{\max}$ szélességű intervallumot is feldaraboltuk. Ekkor a listaelemek szélességének új maximuma $\mathrm{w}_{\max}:=\mathrm{w}_{\max}/s$. Eközben nyilván minden egyes intervallumot s^n részintervallumra darabolunk, így minden $l\in N$ értékre $N_{l+1}=s^nN_l$ teljesül. Mivel $N_0=1$, azaz a 0-dik szint iterációit megelőzően csupán az $A=X$ intervallum van jelen, a rekurzív egyenlőséget explicit módon kifejezve következik a lemma állítása. □

Ennek alapján felállíthatjuk azt az összefüggést, amely megmutatja, hogy hány iterációs ciklus tartozik egy-egy iterációs szinthez [22].

18. Tétel. Az 5. Algoritmus l-dik iterációs szintjéhez tartozó iterációs ciklusok száma

$$k_l=\frac{s^{nl}(s^n-1)}{s-1}. \tag{60}$$

Bizonyítás: Minden, az L listán található intervallumot egybevágó részintervallumokra kell darabolnunk. Mivel az adott szintre lépéskor szintén egybevágó intervallumok találhatók a listán, vizsgáljunk először csak egyet ezek közül. Mivel ez feltevésünk szerint olyan, hogy egyik éle sem legalább s-szerese bármely másiknak, az intervallumot először az egyik koordináta-irányra (a leghosszabb él irányára) merőlegesen, majd a keletkező részintervallumokat egy másik, erre az irányra merőleges irányra merőlegesen, stb. szeleteljük egészen addig, míg mind az n irányt ki nem merítettük. A szeletelések minden ilyen lépcsőjénél —azaz irányánál — a továbbdarabolandó részintervallumok száma éppen s-szer akkora, mint az előző lépcsőjénél volt. A szeletelés lépcsőinek száma tehát n, az X dimenzióinak száma. Egyetlen intervallum uniform darabolásához így összesen

$$\sum_{i=1}^{n}s^{i-1}=\frac{s^n-1}{s-1} \tag{61}$$

iterációs ciklusra van szükség.

Kezdetben s^{nl} egybevágó intervallumunk lévén (7. Lemma), megszorozva ezt a (61) eredményével, éppen a k_l értékét adó formulát kapjuk. □

Annak megállapításához, hogy X egy bizonyos finomságú felbontásának eléréséig hány iterációs ciklust kell végrehajtanunk, elegendő kiszámítani, pontosan mely iterációs ciklusok tartoznak az l-dik iterációs szinthez. A következő tétel [22] erre a kérdésre ad választ.

19. Tétel. Az 5. Algoritmus k-dik iterációs ciklusa az l-dik iterációs szinten van akkor és csak akkor, ha teljesül, hogy

$$\frac{s^{nl}-1}{s-1}+1 \le k \le \frac{s^{n(l+1)}-1}{s-1}. \tag{62}$$

Bizonyítás. (62) közvetlenül következik (60)-ból. A 18. Tétel azt mondja ki, hogy az l-dik szinten pontosan k_l darab iterációs ciklus található (ld. (60)). A k_l értékeit összegezve kapjuk, hogy

$$\sum_{i=0}^{l-1} k_i = \sum_{i=0}^{l-1} \frac{s^{ni}(s^n-1)}{s-1} = \frac{s^n-1}{s-1}\sum_{i=0}^{l-1} s^{ni} = \frac{s^{nl}-1}{s-1}. \tag{63}$$

Így az l-dik szintig elvégzett iterációk számát (63) szolgáltatja. A következő iteráció már az l-dik szinthez tartozik, amiből következik (62) első egyenlőtlensége.

Az l-dik szint utolsó iterációs ciklusát (63)-hoz hasonlóan megkapjuk, ha az összegzést $l-1$ helyett l-ig végezzük. Ebből megkapjuk a második egyenlőtlenséget is. □

Ezek után kimondhatjuk az 5. Algoritmus konvergenciájának sebességére vonatkozó tételünket [22].

20. Tétel. Ha F az f egy α-konvergens befoglalófüggvénye X felett, akkor

$$\mathrm{lb}(f(X)) - \min_{Y \in L \cup \{A\}} \mathrm{lb}(F(Y)) \le c\,\mathrm{w}^{\alpha}(X)\, s^{\alpha}(k(s-1)+1)^{-\alpha/n} \tag{64}$$

igaz, ahol c a legkisebb olyan pozitív konstans, melyre az α–konvergencia teljesül, A pedig a k-dik iteráció aktuális intervalluma.

Bizonyítás. Mivel az 5. Lemma (26) egyenlőtlensége igaz speciálisan minden listaelemre is, így igaz, hogy

$$\mathrm{lb}(f(X)) - \min_{Y \in L \cup \{A\}} \mathrm{lb}(F(Y)) \le c\,\mathrm{w}^{\alpha}(Y^*), \tag{65}$$

ahol Y^* jelöli azt az intervallumot, melyen a $\min_{Y \in L \cup \{A\}} \mathrm{lb}(F(Y))$ minimum felvétetik.

Amennyiben a k-dik iteráció az l-dik szinten van, a 12. Definícióban szereplő (59) második egyenlőtlenségéből következik, hogy

$$(\forall Y \in L \cup \{A\}) \quad w(Y) \leq s^{-l} w(X), \tag{66}$$

mivel $w(Y) \leq w_{max}$ mindig igaz.

Másfelől alakítsuk át (62) második egyenlőtlenségét. Az átalakítás lépései:

$$k \leq \frac{s^{n(l+1)} - 1}{s - 1}$$
$$k(s-1) \leq s^{n(l+1)} - 1 = s^{nl} s^n - 1$$
$$s^{-n}(k(s-1)+1) \leq s^{nl}$$
$$s^{-1}(k(s-1)+1)^{1/n} \leq s^l,$$

azaz

$$s^{-l} \leq s(k(s-1)+1)^{-1/n}. \tag{67}$$

Helyettesítsük most be (67)-et (66)-ba, majd az így kapott összefüggést (65)-be, így megkapjuk a bizonyítandó összefüggést. □

A fenti tétel (64) eredménye részletesebb vizsgálatra szorul. Először is érdemes az egyes tényezőktől való függést kifejezni. Csupán a nagyságrendre gyakorolt hatást tekintve (64) átírható úgy, hogy az alsó becslés globális optimumértéktől való távolságára s-szeletelés esetén a k-dik iterációs ciklusban $O(s^{\alpha(n-1)/n} k^{-\alpha/n})$ értéket kapunk. Mivel $\alpha > 0$ mindig igaz, azonos iterációnál megállva ez a felső korlát s növelésével az $n \geq 2$ esetben nyilván szigorúan monoton módon nő, azaz lassabb konvergenciára utal. Ez a valóságban nem mindig igaz, legrosszabb esetben viszont előfordulhat. Nyilván minél nagyobb az s értéke, annál kevesebb iterációs ciklus alatt érhetjük el az X kiindulási intervallum azonos felbontását. Speciálisan, ha például $s=4$, és így az első szint eléréséhez szükséges iterációs ciklusok száma k, akkor a felező eljárásnak ugyanezen felbontás eléréséhez még egy szintre és éppen $3k$ darab iterációs ciklusra van szüksége, azaz háromszor többre, mint a 4-szeletelő eljárásnak. Vegyük észre, hogy egydimenziós problémák esetében a szeletelés nem változtat a konvergencia-sebességen (vö. 20. Tétel)! Általánosságban a következő állítás [22] igaz a legjobb esetben.

21. Tétel. Alkalmazzuk a 20. Tétel jelöléseit és feltételrendszerét, valamint tegyük fel, hogy a legjobb eset teljesül, azaz a k-dik iterációs ciklus valamely l-dik iterációs szint kezdő lépése. Ekkor igaz a következő egyenlőtlenség:

$$\mathrm{lb}(f(X)) - \min_{Y \in L \cup \{A\}} \mathrm{lb}(F(Y)) \leq c\, w^{\alpha}(X)((k-1)(s-1)+1)^{-\alpha/n} \tag{68}$$

Bizonyítás. A 19. Tétel valamint az állítás feltételei miatt (62) első egyenlőtlenségében az egyenlőség teljesül, azaz

$$k = \frac{s^{nl} - 1}{s - 1} + 1,$$

amit átrendezve kapjuk, hogy

$$s^{-l}=((k-1)(s-1)+1)^{-1/n}. \tag{69}$$

Behelyettesítve ezek után (69)-et (66)-ba, a kapott egyenlőtlenséget pedig (65)-be, az állítást igazoltuk. □

(68) már jól mutatja, hogy amennyiben az adminisztrációs lépések műveletigényétől eltekintünk, az alsó közelítés pontosságára adható becslés ugyanolyan módon javul s növelésének hatására, mint ahogyan azt az iteráció-számláló növekedése teszi.

Vizsgáljuk most meg azt, hogy milyen mértékben változik az azonos felbontás eléréséhez szükséges iterációs szintek és lépések száma s növelésének hatására abban az esetben, ha teljes iterációs szinteket tekintünk [22]!

22. Tétel. Ha az 5. Algoritmus valamely l-dik szintig való végrehajtásához s értékét a p-szeresére növeljük ($p>1$, ps egész), akkor a kiindulási intervallum legalább ugyanolyan finomságú felbontásának eléréséhez szükséges iterációs szintek száma a β-szorosára csökken, ahol

$$\beta \geq \left(1+\frac{\log p}{\log s}\right)^{-1}. \tag{70}$$

Ugyanakkor a szükséges iterációszám a γ-szorosára változik, ahol

$$\gamma \geq \frac{s-1}{ps-1}. \tag{71}$$

Bizonyítás. Mivel az 5. Algoritmus nem tartalmaz gyorsító eljárást, valamint uniform darabolást hajt végre, a szeletelések során keletkező tárolt részintervallumok száma egyben meghatározza a felbontás finomságát, azaz a részintervallumok szélességét. Az elemszám azonban a 7. Lemma szerint egyszerűen meghatározható. Az s-szeletelés esetén az l-dik szintre érve az elemszám s^{nl}, míg ps-szeleteléskor $(ps)^{n\beta l}$, mivel l is nyilván valami β-szorosára változik. A feltétel szerint az utóbbi érték legalább akkora kell legyen, mint az előbbi, amiből a következő adódik:

$$s^{nl} \leq (ps)^{n\beta l}, \tag{72}$$

$$n\beta l \geq \log_{ps} s^{nl} = nl\frac{\log s}{\log ps} = \frac{nl}{1+\frac{\log p}{\log s}},$$

ami ekvivalens a (70) összefüggéssel. Ugyanezt az eredményt kapjuk, ha a listahosszak helyett a listaelemek maximális szélességének 12. Definíciójának felhasználásával írjuk fel az összefüggést.

A 19. Tételbeli (62) első egyenlőtlensége, pontosabban a 19. Tétel bizonyításában szereplő (63) miatt az s-szeletelés l-dik szintjének megkezdéséig

$$k = \frac{s^{nl} - 1}{s - 1} \tag{73}$$

darab iterációs ciklust tettünk meg. A ps-szeletelést választva hasonlóképpen

$$\gamma k = \frac{(ps)^{n\beta l} - 1}{ps - 1} \tag{74}$$

iterációs ciklus szükségeltetik az azonos finomságú felbontást eredményező βl-dik szint eléréséhez. Írjuk át a (73) egyenlőséget a (72) egyenlőtlenség segítségével:

$$k \le \frac{(ps)^{n\beta l} - 1}{s - 1} \tag{75}$$

A γk értékét a (74) egyenletből behelyettesítjük, majd felhasználjuk a k értékére a (75)-ben kapott becslést:

$$\gamma = \frac{\gamma k}{k} = \frac{\dfrac{(ps)^{n\beta l} - 1}{ps - 1}}{k} \ge \frac{\dfrac{(ps)^{n\beta l} - 1}{ps - 1}}{\dfrac{(ps)^{n\beta l} - 1}{s - 1}} = \frac{s - 1}{ps - 1},$$

ami éppen a tétel (71) állítása. □

A 22. Tétel azt az esetet vizsgálja, amikor a 5. Algoritmus egy-egy teljes iterációs szintet elvégzett. A tétel (70) állítása nyilván általános esetben is éles abban az értelemben, hogy

$$\beta l = \left\lceil l \left(1 + \frac{\log p}{\log s} \right)^{-1} \right\rceil .$$

A tétel másik állítása, mely a (71) egyenlőtlenségben van megfogalmazva, sajnos durva alsó becslés, és bizonyos esetekben γ értéke akár 1 fölé is nőhet, ami lassulást jelent az eredeti s-szeleteléshez képest. Ennek oka, hogy az éppen alkalmazott megállási feltétel mellett az s-szeletelés egy szint első lépése után megállhat, míg ps-szeletelés mellett a (70) segítségével kiszámítható szint utolsó vágása után kapja csak meg a megfelelő eredményt a megálláshoz. Ez azt vonhatja maga után, hogy a ps-szeletelés egy teljes szinttel több iterációs ciklust kénytelen végrehajtani, ami a 18. Tétel szerint jelentős műveletigénybeli növekedést eredményezhet. A következő tétel azt pontosítja, hogy milyen korlátok között változhat a 22. Tételbeli γ értéke, amennyiben az eljárások tetszőleges iterációs ciklus esetén megállíthatók [22].

23. Tétel. Hajtsuk végre az 5. Algoritmust először s-szeleteléssel, majd ps-szeleteléssel ($p>1$, ps egész) úgy, hogy a megállási feltétel az aktuális intervallum szélességétől függjön közvetlen vagy közvetett módon. Ekkor a végrehajtás-

hoz szükséges iterációs ciklusok száma a második esetben a γ-szorosára változik, ahol γ értékére igaz, hogy

$$\frac{(s-1)((ps)^{n\beta l}+ps-2)}{(ps-1)(s^{n(l+1)}-1)} \le \gamma \le \frac{(s-1)((ps)^{n(\beta l+1)}-1)}{(ps-1)(s^{nl}+s-2)}. \tag{76}$$

A képletekben szereplő l és βl azokat az iterációs szinteket jelzik, amelyeken a két végrehajtás során az algoritmus megállt.

Bizonyítás. A (76) összefüggés két egyenlőtlensége a legjobb, illetve a legrosszabb esetet mutatja, ami a *ps*-szeleteléssel elérhető javulást illeti.

A legjobb esetben az *s*-szeletelés egy l-dik szint utolsó iterációs ciklusában képes csak megállni, míg a *ps*-szeletelés egy bizonyos βl-dik szintre érve azonnal megáll. Ezeket a 19. Tétel segítségével képlettel felírva kapjuk, hogy

$$k = \frac{s^{n(l+1)}-1}{s-1}, \tag{77}$$

illetve

$$\gamma k = \frac{(ps)^{n\beta l}-1}{ps-1}+1. \tag{78}$$

Mivel (77) felső korlát k értékére, (78) pedig alsó korlát γk-ra, a két egyenlőséget elosztva egymással alsó korlátot kapunk γ értékére, így megkapjuk (76) első egyenlőtlenségét.

Vizsgáljuk most meg a legrosszabb esetet. Ekkor az *s*-szeletelés rögtön megáll, amint egy bizonyos l iterációs szintre ér, míg a *ps*-szeletelés végigdarabolja az egész βl-dik szintet. Az iteráció-számláló így a két esetben ismét a 19. Tételt alkalmazva a következő lesz:

$$k = \frac{s^{nl}-1}{s-1}+1, \tag{79}$$

valamint

$$\gamma k = \frac{(ps)^{n(\beta l+1)}-1}{ps-1}. \tag{80}$$

Ebben az esetben (79) alsó korlát k lehetséges értékeire, míg (80) felső korlát γk-ra. A két egyenlőség hányadosa így felső korlátot szolgáltat k értékére, amivel a tételt bizonyítottuk □

Mindazonáltal egyes esetekben a szeletelő eljárások valóban képesek gyorsítani az intervallum-felosztási módszereket. Ez több tényezőből tevődik össze, a két fő ok azonban az, hogy az iterációs ciklusonként csupán egyszer végrehajtandó szá-

mításokra fordított idő lényegesen csökken, másrészt gyorsító eljárások alkalmazásakor hamarabb kizárhatók globális optimumhelyet nem tartalmazó részintervallumok a vizsgálatokból.

A szeletelő eljárások hatását tesztfüggvényeken konkrét programfuttatásokkal is megvizsgáltuk [13], melyek az elméleti eredmények által adottaknál szélesebb körben nyújtottak információt a szeletelések tulajdonságairól (9. és 10. Táblázat). A futtatások során a Karlsruhei Egyetem kutatói által kifejlesztett C-XSC könyvtárakat felhasználó Toolbox for Validated Numerics programcsomagot alakítottuk át, hogy alkalmas legyen a szeletelő eljárások alkalmazására. A szakirodalomban széles körben alkalmazott kilenc felhasznált tesztfüggvény pontos leírása megtalálható a függelékben. Az algoritmus megvalósítása során a vágási irány meghatározásához a 3.1. fejezetben ismertetett A, B, illetve C szabályokat alkalmaztuk. A futtatást az $s=2,3,4,5,7$ értékekre végeztük el. Megállási feltételként a (30) összefüggést használtuk az $\varepsilon=10$ értékkel, melyhez a befoglalófüggvény értékeinek kiszámítása nem vett igénybe külön függvényhívást, mivel azt a listára helyezésük előtt kiszámítottuk, és a befoglalás szélességét a részintervallumokkal együtt tároltuk. A vágási irány kiválasztásához a B és C szabályt alkalmazva szükség volt a célfüggvény gradiense befoglalófüggvényének kiszámítására is. Ezek műveletigényét külön nem tüntettük fel, mivel az az implementációnál fogva megegyezik a befoglalófüggvény hívásainak számával.

A szándékosan sokféle tesztfüggvényen az eljárások különböző módon viselkednek. Az eredményeket két külön táblázatba foglaltuk, melyek közül a 9. Táblázat a szükséges iterációk, míg a 10. Táblázat a szükséges függvényhívások számát jelzi.

A 9. Táblázat eredményei jól mutatják, hogy a Schwefel2.7 függvényt kivéve minden egyes tesztfüggvényen csökkent a szükséges iterációs ciklusok száma s értékének növelésével. Mivel a legnagyobb műveletigényű lépések a függvényhívások az eljárásokon belül, érdemes külön megvizsgálni a 10. Táblázat eredményeit is. Ebben a tekintetben a javulás nem olyan szembetűnő, sőt, az esetek egy részében határozottan nőtt a szükséges függvényhívások száma. Mindamellett a kevesebb iterációs ciklus miatt az egyéb műveletek számának csökkenését nem szabad figyelmen kívül hagynunk.

Átlagolva a kapott eredményeket adódik, hogy még a Schwefel2.7 tesztfüggvény esetén adódó eltérő viselkedés ellenére is az $s=7$ esetet kivéve a szükséges iterációs ciklusok száma jelentősen csökken, míg a szükséges függvényhívások száma kismértékben nő.

További vizsgálatok szükségesek annak kiderítésére, hogy milyen hatással lehet a vágások számának valamilyen paraméterek alapján való futás közbeni módosítása az algoritmusok viselkedésére.

Szabály	Tesztfüggvény	s=2	s=3	s=4	s=5	s=7
A	Shekel5	48	15	17	21	14
	Hartman3	64	20	22	32	57
	Branin	31	13	21	6	8
	Rosenbrock	15	13	5	6	8
	THCB	46	67	39	16	28
	Levy8	223	121	183	91	85
	Schwefel2.1	68	49	27	39	10
	Schwefel2.7	14	27	43	64	168
	Griewank5	127	121	5	6	1
	Kerekített átlag	**71**	**50**	**40**	**31**	**42**
B	Shekel5	36	15	17	21	14
	Hartman3	26	14	18	11	15
	Branin	19	13	21	6	8
	Rosenbrock	15	13	5	6	8
	THCB	46	42	39	16	8
	Levy8	128	40	61	81	63
	Schwefel2.1	67	39	23	33	54
	Schwefel2.7	14	27	43	64	64
	Griewank5	127	121	5	6	1
	Kerekített átlag	**53**	**36**	**26**	**27**	**26**
C	Shekel5	40	19	8	9	12
	Hartman3	26	14	18	11	15
	Branin	19	13	21	6	8
	Rosenbrock	15	13	5	6	8
	THCB	46	42	21	16	8
	Levy8	128	40	61	81	63
	Schwefel2.1	67	39	23	33	54
	Schwefel2.7	14	27	43	64	64
	Griewank5	127	121	5	6	1
	Kerekített átlag	**54**	**36**	**23**	**26**	**26**

9. Táblázat. A szeletelő algoritmus által végrehajtott iterációs ciklusok száma az s=2, 3, 4, 5, 7 értékekben.

Szabály	Tesztfüggvény	s=2	s=3	s=4	s=5	s=7
A	Shekel5	96	45	68	105	98
	Hartman3	128	60	88	160	399
	Branin	62	39	84	30	56
	Rosenbrock	30	39	20	30	56
	THCB	92	201	156	80	196
	Levy8	446	363	732	455	595
	Schwefel2.1	136	147	108	195	70
	Schwefel2.7	28	81	172	320	1176
	Griewank5	254	363	20	30	7
	Kerekített átlag	**141**	**149**	**161**	**156**	**295**
B	Shekel5	73	46	69	106	99
	Hartman3	53	43	73	56	106
	Branin	39	40	85	31	57
	Rosenbrock	31	40	21	31	57
	THCB	93	127	157	81	57
	Levy8	257	121	245	406	442
	Schwefel2.1	135	118	93	166	379
	Schwefel2.7	29	82	173	321	449
	Griewank5	225	364	21	31	8
	Kerekített átlag	**104**	**109**	**104**	**137**	**184**
C	Shekel5	81	58	33	46	85
	Hartman3	53	43	73	56	106
	Branin	39	40	85	31	57
	Rosenbrock	31	40	21	31	57
	THCB	93	127	85	81	57
	Levy8	257	121	245	406	442
	Schwefel2.1	135	118	93	166	379
	Schwefel2.7	29	82	173	321	449
	Griewank5	255	364	21	31	8
	Kerekített átlag	**108**	**110**	**92**	**130**	**182**

10. Táblázat. A szeletelő algoritmus által végrehajtott függvényhívások száma az s=2, 3, 4, 5, 7 értékekben.

4.2. Többszörös vágás

A többszörös vágási eljárások az utóbbi néhány évben komoly javulást ígérnek az intervallum-felosztási algoritmusok terén [26]. Ezen módszerek — a szeletelő eljárásokkal együtt — az egyszerre keletkező részintervallumok nagyobb számánál fogva természetes módon kínálják fel a párhuzamosítás lehetőségét (pl. [3]). Mivel a technikai újításokon túlmutatva a számítógépes programok gyorsítása a párhuzamosításban rejlik, a szeletelő és többszörös vágási algoritmusok ezen tulajdonsága önmagában is figyelemre méltó.

A többszörös vágást megvalósító algoritmusok általános elméleti vizsgálata nehezebb, mint a szeletelő algoritmusoké. Már a felező módszerek esetében is igen nehéz betekintést nyerni az uniform darabolástól eltérő módon felépített módszerek működésének szerkezetébe, mivel a kezelt listák felépítése feladatról feladatra jelentősen eltérhet egymástól (pl. Moore-Skelboe algoritmus). Többszörös vágás esetében az egyszerű felező algoritmusoktól eltérően ráadásul semmit sem tudunk arról, hogy a listánkon várakozó különböző szélességű intervallumok milyen vágás során kerültek fel a listára. Uniform darabolás esetén biztosak lehetünk abban, hogy minden vágás során keletkező részintervallum az iterációs ciklusok következő szintjén továbbdarabolásra kerül. Arra vonatkozó vizsgálataink, hogy ezek a tapasztalatok hogyan vihetők át a többszörös vágást megvalósító algoritmusokra, jelenleg folynak, így ezen dolgozatból szükségképpen még hiányoznak.

Speciális esetben azonban felhasználhatjuk a 4.1. fejezetben nyert ismereteinket. Mivel uniform darabolásnak minősül az is, amikor egy iterációs cikluson belül minden koordináta-irányra merőlegesen elvégzünk egy vágást, ebben az esetben a listaelemek szükségszerűen időről időre egybevágóak, azaz az iterációs ciklusok egyfajta szintenkénti felosztása itt is lehetséges. A iterációs szintekre osztás 12. Definíciójának elméleti jelentősége abban rejlett, hogy az iterációs ciklusokat osztályokba zártuk. Egy ilyen osztály lépéseit végrehajtva az adott iterációs szint kezdő listájához képest annak végére a listán csupa, a kezdő elemekhez geometriai értelemben hasonló, de s-ed élhosszúságú intervallumok állnak. Amennyiben a többszörös vágást minden koordináta-irányra elvégezzük egy-egy iterációs cikluson belül, úgy ebből a szempontból egyetlen iterációs ciklus felel meg egy egész szintnek. A 5. Algoritmus egyszerűen módosítható úgy, hogy ennek megfelelően működjön. Ennek érdekében elég a 2. iterációs lépést megváltoztatni, így kapjuk az alábbi módszert:

6. Algoritmus. Hansen intervallum-kiválasztási szabályát alkalmazó intervallum-felosztási algoritmus többszörös vágási eljárással minden koordináta-irányra.

1. Legyen L egy üres lista, $A:=X$ az aktuális intervallum! Állítsuk a k iteráció-számlálót $k:=1$ értékre!

2. Daraboljuk az A intervallumot s darab A_i ($i=1,2,\ldots,s$) egybevágó részintervallumra oly módon, hogy azt minden koordináta-irányára merőlegesen egy-egy $\sqrt[n]{s}$ -ekvidisztáns beosztás mentén arra merőleges hipersíkokkal felszeleteljük!
3. Helyezzük a lista végére az újonnan keletkezett intervallumokat, azaz legyen $L:=L\cup\{A_1\}\cup\{A_2\}\cup\ldots\cup\{A_s\}$!
4. —
5. Válasszuk ki a listáról az $A\in L$ legelső intervallumot, és emeljük le onnan, azaz legyen $L:=L\setminus\{A\}$!
6. Ha a megállási feltétel nem teljesül, növeljük az iteráció-számlálót, $k:=k+1$, és térjünk vissza a 2. lépéshez!
7. Nyomtassuk ki az eredményt, és STOP!

A fenti algoritmus implicit módon nyilván tartalmazza azt a feltételt, hogy $\sqrt[n]{s}$ egész szám. Ezek után kimondható a következő állítás:

24. Tétel. Amennyiben F az f-nek α-konvergens befoglalófüggvénye X felett, akkor a 6. Algoritmusra

$$\mathrm{lb}(f(X))-\min_{Y\in L\cup\{A\}}\mathrm{lb}(F(Y))\le c\,\mathrm{w}^{\alpha}(X)s^{\alpha}(k(s^n-1)+1)^{-\alpha/n} \tag{81}$$

mindig teljesül, ha c a 6. Definícióbeli konstans, L a 6. Algoritmus k-dik iterációs ciklusának listája, A pedig aktuális intervalluma.

Bizonyítás. Mivel az iterációs szintek 12. Definíciója alkalmazható a 6. Algoritmusra is, vizsgáljuk meg ezen terminológiát felhasználva először azt, hogy mely k_l iterációs ciklusok tartoznak az l-dik szinthez. Mivel a 7. Lemma itt változatlan formában igaz, azt felhasználva kapjuk, hogy

$$k_l=s^{nl}. \tag{82}$$

Valóban, a 18. Tétel bizonyításabeli (61)-ben számított érték, azaz az egyetlen intervallum uniform darabolásához szükséges iterációs ciklusok száma ugyanis ebben az esetben 1.

A 19. Tétel bizonyításához hasonlóan összegezve az egyes iterációs szintek iterációs ciklusait (82) segítségével kapjuk, hogy a 6. Algoritmus l-dik szintjének iterációira

$$k\le\sum_{i=0}^{l}k_i=\sum_{i=0}^{l}s^{nl}=\frac{s^{n(l+1)}-1}{s^n-1} \tag{83}$$

teljesül.

Alakítsuk át a (83) egyenlőtlenséget:

$$k \le \frac{s^{n(l+1)}-1}{s^n-1}$$
$$k(s^n-1) \le s^{n(l+1)}-1 = s^{nl}s^n-1$$
$$s^{-n}(k(s^n-1)+1) \le s^{nl}$$
$$s^{-1}(k(s^n-1)+1)^{1/n} \le s^l$$
$$s^{-l} \le s(k(s^n-1)+1)^{-1/n}.$$

Ebből az iterációs szintek definícióját felhasználva kapjuk, hogy

$$(\forall Y \in L \cup \{A\}) \quad w(Y) \le s(k(s^n-1)+1)^{-1/n} w(X),$$

amiből az 5. Lemma alapján adódik az állítás. □

A 24. Tétel speciális esete a többszörös vágási eljárásoknak. Összevetve (81)-at a 20. Tétel (64) eredményével látszik, hogy míg ott legrosszabb esetben az s növelése ronthatta a pontosságra adható becslést, addig itt kismértékben javít rajta. A (81) korlát további érdekessége, hogy a becslés nagyságrendje legrosszabb esetben független s-től. Az egyes szintek első iterációs ciklusaira adva meg a korlátot (szeletelés esetében ld. 21. Tétel) az s-től való függés nagyságrendje $\mathcal{O}(s^{-\alpha})$.

Szabály		Processzor idő			Maximális listahossz			Függvényhívások száma		
		% átlag	Össz.	/(A/2)	% átlag	Össz.	/(A/2)	% átlag	Össz.	/(A/2)
A	/2		4 180			228			502 360	
	/3	101%	4 134	99%	125%	276	121%	103%	411 439	82%
	/4	110%	5 613	134%	147%	360	158%	113%	434 574	87%
B	/2	82%	675	16%	89%	71	31%	83%	131 822	26%
	/3	84%	551	13%	104%	78	34%	86%	107 046	21%
	/4	100%	562	13%	130%	92	40%	104%	105 306	21%
C	/2	81%	666	16%	88%	71	31%	82%	132 395	26%
	/3	80%	518	12%	103%	73	32%	84%	102 674	20%
	/4	97%	521	12%	128%	84	37%	101%	98 950	20%
D	/2	136%	5 369	128%	142%	299	131%	129%	664 474	132%
	/3	112%	8 830	211%	141%	409	179%	112%	741 433	146%
	/4	110%	12 871	308%	148%	579	254%	111%	801 293	160%

11. Táblázat. A megoldáshoz szükséges processzor idő, maximális listahossz és a függvényhívások száma 37 tesztfeladaton a négy különböző intervallum-felosztási szabályra 2, 3, és 4 darabra való többszörös vágás esetén.

További vizsgálatok szükségesek ahhoz, hogy ennek a tulajdonságnak az általános esetre való átvihetőségét megvizsgáljuk, illetve általánosságban megbecsüljük ennek az adminisztrációs lépések számának növekedésére gyakorolt hatását egy-egy iterációs cikluson belül. Mindazonáltal néhány biztató gyakorlati eredmény már jelenleg is rendelkezésre áll [22, 27]. A 11-12. Táblázatokban ezek eredmé-

nyeit foglaltuk össze. A két táblázat 37 tesztfeladat megoldásánál szerzett tapasztalatokat összegzi, külön-külön megadva a futtatásokhoz szükséges processzor időt másodpercekben, az egyes futtatások során a tároláshoz szükséges legnagyobb listahosszat, valamint az iterációs ciklusok számán kívül a megfelelő függvényhívások számát. Utóbbiak esetében a gradiens és Hesse mátrix számításokra a Newton módszer alkalmazása miatt volt minden esetben szükség. A futtatási eredmények megtalálhatók a felezéses, valamint a három, illetve négy darabra való többszörös vágás esetére is. Az eredmények az egyes felosztási szabályokra különkülön meg vannak adva. A különböző mérési adatokra három-három mérőszám is utal, ezek közül a második és harmadik oszlopokban levők a mérőszámok összegeit, illetve az A felosztási módszer felező eljáráshoz viszonyított százalékos arányait adják meg. Az első oszlopok az A felező eljáráshoz képest az egyes tesztfeladatokon tapasztalt százalékos eltérések átlagát mutatják.

		Gradiens hívások száma			Hesse hívások száma			Iterációs ciklusok száma		
Szabály		% átlag	Össz.	/(A/2)	% átlag	Össz.	/(A/2)	% átlag	Össz.	/(A/2)
	/2		289 016			39 243			79 422	
A	/3	108%	254 050	88%	96%	24 809	63%	88%	57 964	73%
	/4	123%	283 938	98%	86%	15 852	40%	88%	58 545	74%
	/2	84%	76 234	26%	82%	10 185	26%	81%	19 151	24%
B	/3	92%	66 296	23%	76%	7 071	18%	71%	13 511	17%
	/4	114%	69 046	24%	75%	5 344	14%	80%	12 501	16%
	/2	84%	76 794	27%	82%	10 500	27%	81%	19 053	24%
C	/3	89%	63 629	22%	74%	6 860	17%	70%	12 863	16%
	/4	110%	64 918	22%	70%	5 123	13%	78%	11 628	15%
	/2	126%	373 084	129%	119%	44 789	114%	137%	112 829	142%
D	/3	118%	440 675	152%	101%	41 943	107%	98%	104 188	131%
	/4	121%	522 237	181%	78%	21 592	55%	87%	114 472	144%

12. Táblázat. A megoldáshoz szükséges gradiens és Hesse mátrix függvényhívások száma, valamint iterációs ciklusok száma 37 tesztfeladaton a négy különböző intervallum-felosztási szabályra 2, 3, és 4 darabra való többszörös vágás esetén.

A következő fejezetben áttérünk egy másik, az intervallum-felosztási módszerek esetében esszenciális kérdéskörre, a gyorsító eljárások elméleti vizsgálatára.

5. A gyorsító eljárások hatásvizsgálata

A 4. fejezetben olyan módszerekkel foglalkoztunk, melyekben nem vettük figyelembe a széles körben elterjedt gyorsító eljárások hatását. Az algoritmusok legrosszabb működésének leírásakor így is kell tenni, a gyakorlatban felhasznált eljárások tényleges működéséhez képest azonban ezek az eredmények nem feltétlenül irányadóak. A különböző gyorsító eljárások más-más problémakörökben működnek hatékonyan, illetve bizonyos feladatosztályokon némelyek nem is alkalmazhatók. Áttekintő vizsgálatra így csak akkor van lehetőség, ha olyan közös tulajdonságot találunk a gyorsító eljárások legalább egy meghatározott halmazán, mely lehetővé teszi az egységes, elméletileg megfogalmazható vizsgálatokat. A következőkben erre teszünk kísérletet (ld. [15, 16, 22]) a szeletelő eljárások esetében.

5.1. Az η feltevés

A gyorsító eljárások alatt továbbra is csak azokat a módszereket fogjuk érteni, amelyek képesek a darabolás során keletkezett részintervallumok közül olyanokat, melyek nem tartalmaznak globális optimumhelyet, kizárni a további vizsgálatokból. A gyorsító eljárások közül így nem tekintjük azokat, melyek egyéb műveleteket (pl. továbbdarabolás az intervallumos Newton módszer esetében) is elvégeznek a listaelemeken.

A következőkben bevezetünk egy egységes fogalmat ezen gyorsító eljárások egy halmazára, mely semmi egyéb specialitást nem követel meg.

13. Definíció. Gyorsító eljárások egy halmazát akkor nevezzük *η-gyorsító eljárás*nak, ha elemei összességének segítségével minden egyes szint során a listaelemek legalább $1-\eta$ hányada törölhető ($0 \le \eta \le 1$).

Az 13. Definíció nyilván csak uniform darabolást megvalósító eljárásokra vonatkozik, egyébként az iterációs szint fogalma nem is érvényes.

A következőkben ismertetésre kerülő 13–15. Táblázatokban néhány tesztfüggvényen megvizsgáljuk, értelmes-e a 13. Definíció, azaz elméletileg megadható-e egyáltalán ilyen η érték. A futtatásokhoz az 5. Algoritmust alakítottuk át oly módon, hogy az a 4. lépésében valamilyen gyorsító eljárást, vagy több ilyen eljárás kombinációját tartalmazza. A felhasznált tesztfüggvények megegyeznek a 9. és 10. Táblázatok eredményeihez felhasználtakkal, leírásuk a függelékben megtalálható.

A megvalósításkor s értékét mindig 2-re állítottuk be, és a futtatást 10 szint megtétele után félbeszakítottuk, ahol a lista túlnövekedése hamarabb meg nem állította az algoritmust. Az utóbbi esetben „–" jellel jelöltük a további adatok hiányát.

Az 13–15. Táblázatokban szereplő értékek azt jelzik, hogy adott iterációs szintre érve az előző szint során előálló részintervallumok hány százalékát nem volt képes egy adott η-gyorsító eljárás törölni. Például a 13. Táblázat **S5** oszlopában a 3. szinten álló 15,3% érték azt jelzi, hogy a monotonitási teszt alkalmazásának következtében az algoritmusnak a Shekel5 tesztfüggvényen való végrehajtása során az l=2 iterációs szint kezdetén a listán szereplő 91 darab intervallum darabolása során keletkező összesen $N_{l+1}=s^n N_l=2^4 \cdot 91=1456$ darab részintervallumból a 3. szint kezdetére csupán 223 elem maradt.

A 13. és 14. Táblázatokban a monotonitási és a vágási tesztek hatását különkülön vizsgáltuk.

Szint	S5	H3	Br	Rb	THCB	L8	Sch2.1	Sch2.7	G5
1	100,0%	50,0%	100,0%	75,0%	100,0%	100,0%	100,0%	75,0%	100,0%
2	35,5%	50,0%	56,2%	58,3%	50,0%	100,0%	62,5%	68,8%	3,1%
3	15,3%	31,2%	63,9%	53,6%	46,9%	82,8%	50,0%	40,2%	6,2%
4	12,0%	16,9%	32,6%	50,0%	43,3%	13,5%	47,5%	41,9%	–
5	11,6%	14,4%	26,7%	48,3%	34,6%	12,2%	47,4%	41,3%	–
6	–	11,9%	25,8%	45,7%	37,5%	12,8%	42,7%	–	–
7	–	12,9%	24,2%	40,3%	31,5%	8,4%	43,9%	–	–
8	–	10,9%	27,3%	38,7%	21,3%	7,5%	46,3%	–	–
9	–	12,5%	25,0%	31,0%	24,1%	11,5%	43,3%	–	–
10	–	12,0%	22,9%	35,3%	25,0%	13,0%	38,1%	–	–
11	–	13,7%	23,4%	38,9%	24,1%	10,8%	–	–	–

13. Táblázat. A monotonitási teszt által a listán hagyott elemek számaránya (η) százalékosan, szintenként megállva.

A monotonitási teszt alkalmazásakor (13. Táblázat) általában a 3-4. iterációs szinttől kezdve megadható olyan η érték, mely minden későbbi szintre érvényes, sőt, az ott szereplő η értékektől általában nem különbözik lényegesen. Ezek az értékek az egyes függvények esetében az utolsó oszlopot kivéve a 4. szinttől kezdve 0,12, 0,169, 0,326, 0,5, 0,433, 0,135, 0,475 és 0,419, a táblázat szerint balról jobbra haladva a tesztfüggvényeken.

Hasonlóképpen a vágási teszt (14. Táblázat) is lehetővé teszi ilyen η értékek rögzítését, és ez az érték eltér a monotonitási teszt által elért értékektől. Említésre méltó az egyes szintek nagymértékű eltérése.

A konkavitási teszttel végrehajtott eredmények ugyan szintén alátámasztják az η feltevés jogosságát, a vizsgált tesztfeladatokon azonban ez az érték alig csök-

kent 1 alá, aminek következtében a programfutás többnyire az első két-három szint után a túl hosszú lista miatt leállt, így ezeket az eredményeket táblázatszerűen nem is közöljük.

Szint	S5	H3	Br	Rb	THCB	L8	Sch2.1	Sch2.7	G5
1	100,0%	100,0%	100,0%	100,0%	100,0%	100,0%	100,0%	100,0%	100,0%
2	25,0%	64,1%	100,0%	37,5%	100,0%	31,2%	62,5%	70,3%	3,1%
3	38,2%	47,9%	65,6%	54,2%	81,2%	39,4%	30,0%	41,1%	3,1%
4	4,7%	31,6%	32,7%	51,9%	80,8%	33,7%	35,4%	39,7%	3,1%
5	6,6%	–	35,5%	56,5%	49,9%	3,2%	27,9%	–	3,1%
6	1,6%	–	26,9%	34,8%	44,6%	22,2%	36,8%	–	3,1%
7	12,4%	–	19,9%	34,4%	38,5%	8,0%	21,4%	–	3,1%
8	3,1%	–	34,0%	31,0%	–	19,8%	11,5%	–	3,1%
9	12,4%	–	20,1%	33,6%	–	8,2%	20,5%	–	3,1%
10	3,1%	–	25,7%	24,9%	–	19,5%	25,0%	–	3,1%
11	13,8%	–	24,0%	25,1%	–	8,0%	30,6%	–	3,1%

14. Táblázat. A vágási teszt által a listán hagyott elemek számaránya (η) százalékosan, szintenként megállva.

Szint	S5	H3	Br	Rb	THCB	L8	Sch2.1	Sch2.7	G5
1	100,0%	50,0%	100,0%	75,0%	100,0%	100,0%	100,0%	75,0%	100,0%
2	19,1%	46,9%	56,2%	41,7%	50,0%	31,2%	62,5%	68,8%	3,1%
3	17,7%	14,2%	63,9%	45,0%	46,9%	39,4%	30,0%	34,8%	3,1%
4	0,8%	15,4%	20,7%	44,4%	43,3%	9,9%	35,4%	36,4%	3,1%
5	1,1%	3,6%	28,9%	43,8%	28,8%	0,2%	27,9%	38,5%	3,1%
6	2,1%	2,1%	28,4%	39,3%	31,7%	12,5%	36,8%	–	3,1%
7	6,2%	75,0%	22,0%	33,0%	25,0%	12,5%	21,4%	–	3,1%
8	6,2%	10,4%	29,5%	31,9%	22,4%	12,5%	11,5%	–	3,1%
9	6,2%	2,5%	25,0%	35,1%	2,9%	12,5%	20,5%	–	3,1%
10	6,2%	75,0%	22,1%	24,8%	25,0%	12,5%	25,0%	–	3,1%
11	6,2%	8,3%	22,8%	25,0%	25,0%	12,5%	30,6%	–	3,1%

15. Táblázat. A monotonitási és vágási teszt együttes alkalmazása mellett a listán maradt elemek számaránya (η) százalékosan, szintenként megállva.

A 15. Táblázatban több gyorsító eljárás együttes viselkedését figyelhetjük meg. Ennek során együttesen alkalmaztuk a monotonitási és a vágási tesztet. A három teszt együttes alkalmazásának hatásvizsgálatára tett kísérlet, melyben az előbbi két gyorsító eljárás mellett a konkavitási tesztet is felhasználtuk, a fentebb említett okok miatt nem adott ettől lényegesen eltérő adatokat, így ezen futási eredmények

külön itt nem szerepelnek. Megállapítható, hogy a törlési arányok többnyire kiegyenlítődtek.

A vizsgált esetekben általában működik tehát az η feltevés, azaz minden η-gyorsító eljáráshoz és tesztfeladathoz megadható egy szintektől független η, de csupán akkor, ha az első néhány szinttől eltekintünk. Ez a további, elméleti vizsgálatokat nem zavarja, mivel úgy is tekinthetjük, hogy a kiindulási L listánk már eleve tartalmazza az első néhány szinten történt darabolás eredményeképpen keletkezett részintervallumokat. Ennek érdekében az 5. Algoritmust úgy módosítjuk, hogy ezeknek az elvárásoknak megfeleljen.

7. Algoritmus. Hansen intervallum-kiválasztási szabályát alkalmazó szeletelő algoritmus η-gyorsító eljárással.

1. Legyen L egy N_0 darab egybevágó n-dimenziós intervallumból álló lista, melyek uniója része X-nek és tartalmazza az f összes globális optimumhelyét! Legyen a lista első eleme A! Állítsuk a k iteráció-számlálót k:=1 értékre, és legyen $L:=L\setminus\{A\}$!
2. Daraboljuk az A intervallumot s darab A_i ($i=1,2,\ldots,s$) egybevágó részintervallumra oly módon, hogy azt a leghosszabb élére felvett ekvidisztáns beosztás mentén arra merőleges hipersíkokkal felszeleteljük!
3. Helyezzük a lista végére az újonnan keletkezett intervallumokat, azaz legyen $L:=L\cup\{A_1\}\cup\{A_2\}\cup\ldots\cup\{A_s\}$!
4. Töröljük az L listáról azokat az elemeket, melyek az η-gyorsító eljárás értelmében nem tartalmazhatnak globális minimumhelyet.
5. Válasszuk ki a listáról az $A\in L$ legelső intervallumot, és emeljük le onnan, azaz legyen $L:=L\setminus\{A\}$!
6. Ha a megállási feltétel nem teljesül, növeljük az iteráció-számlálót, $k:=k+1$, és térjünk vissza a 2. lépéshez!
7. Nyomtassuk ki az eredményt, és STOP!

Az η-gyorsító eljárást alkotó gyorsító eljárások az algoritmus közben úgy működnek, hogy segítségükkel az L lista a várhatónál legalább $1-\eta$-szorosával rövidül minden szint esetén. Legyen például az l-dik szintre érve a listaelemek száma N_l. Ahogyan azt a 7. Lemma bizonyításában láttuk, az l-dik szint iterációs ciklusait elvégezve a lista az s^n-szeresére duzzad η-gyorsító eljárás nélkül. A 13. Definíció szerint azonban a listaelemek száma ekkorra legfeljebb $\eta s^n N_l$ lehet. Általánosságban ez a következőket jelenti.

8. Lemma. A 7. Algoritmus iterációs ciklusainak l-dik szintjére érve az L lista elemeinek N_l számára mindig teljesül, hogy

$$N_l \le \lfloor (\eta s^n)^l N_0 \rfloor. \tag{84}$$

Bizonyítás. Egyetlen iterációs szint során minden, az adott szintre érkezéskor az L listán tárolt elemet s^n darab egybevágó részintervallumra vágunk. A szint minden lépésében néhány részintervallum törlésre kerül úgy, hogy a szint végére legfeljebb η hányaduk marad a listán. Így a szint összes lépésének elvégzését követően a kezdeti N_{l-1} listaelemből keletkezők közül csupán legfeljebb $\lfloor \eta s^n N_{l-1} \rfloor$ marad a listán. Képlettel felírva $N_l \le \lfloor \eta s^n N_{l-1} \rfloor \le \lfloor (\eta s^n)^l N_0 \rfloor$, amit bizonyítani kellett. □

Az 8. Lemma bizonyításából látszik, hogy a listaelemek számára (84)-nél jobb közelítés is adható. Ez azonban nem ad zárt képletet a kerekítések miatt, így a direkt módon kapható becslés nehezen kezelhető. A közvetlenül a 13. Definícióból adódó korlát a következőképpen írható fel:

6. Megjegyzés. A 7. Algoritmus iterációs ciklusainak l-dik szintjére érve az L lista elemeinek N_l számára mindig teljesül, hogy

$$N_l \le \left\lfloor \eta s^n \ldots \left\lfloor \eta s^n \left\lfloor \eta s^n N_0 \right\rfloor \right\rfloor \ldots \right\rfloor \tag{85}$$

ahol az ηs^n tényező l-szer szerepel.

Mindemellett a (84) becslés elég jó, és elméletileg éles, azaz egyenlőség is előfordulhat. Megbízhatóságát vizsgáljuk meg a 15. Táblázat adatain. Hagyjuk el minden tesztfüggvény esetében az első két-három szintet, a keletkező intervallumokat helyezzük fel a 7. Algoritmus kiindulási listájára, és a fennmaradó szintek η értékeinek maximumával számoljuk ki a táblázatban szereplő utolsó szinteken (84) alapján várható listahosszak felső korlátjait. Azt kapjuk, hogy a kilenc tesztfeladaton a tényleges listahosszak a (84) alapján számított értékeknek átlagosan körülbelül a negyedére csökkennek, azaz a tesztfeladatokon a (84) becslés nagyságrendben jó eredményt szolgáltat. Ezzel szemben az algoritmusokat ugyanonnan indítva, az η feltevés nélkül — legrosszabb esetre — a 7. Lemma felhasználásával kiszámítva a listák növekedését a valós listahosszak a felső becslésnek csupán ezredrészére nőnek. Nyilvánvaló tehát, hogy a gyorsító eljárásokkal kiegészített szeletelő algoritmusok, és így az uniform intervallumfelező módszerek vizsgálatakor is az η feltevés nagyságrendekkel pontosabb eredményeket szolgáltat. A következőkben megvizsgáljuk, hogy milyen hatással van ez a konvergencia sebességére adható eredményekre.

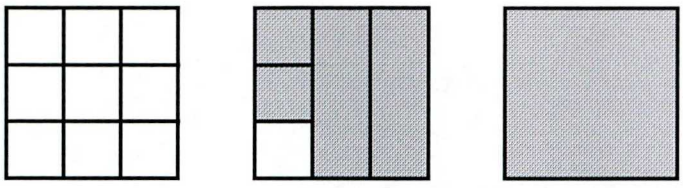
5.2. Konvergencia-sebesség η-gyorsítással

Az uniform darabolást végző intervallum-felosztási algoritmusok egyes szintjein végrehajtandó iterációs ciklusok száma nyilván közvetlenül függ a szint megkezdésekor a listán található elemek számától. Mivel ez utóbbi érték az η-gyorsító eljárások hatására jelentős mértékben csökken, várhatóan ugyanazon iterációs szint is kevesebb iterációs ciklus árán elérhető. A listaelemek törlése a 7. Algoritmus 4. lépésében egy adott szint során bármely iterációs ciklusban bekövetkezhet, melynek szintbeli elhelyezkedését előre megmondani nem tudjuk. Az adott szint elvégzéséhez szükséges iterációs ciklusok száma nagymértékben függ attól, hogy a törlések még a szint első néhány lépésében megtörténnek-e, vagy csupán a szint vége felé. Míg a szint első lépéseiben végezve a törléseket az azonos η hányad eléréséhez fizikailag kevesebb listaelem törlése szükséges, valamint a törölt elemek nagyobb térfogata miatt több iterációs ciklus takarítható meg a szint fennmaradó részére, addig a törléseket a szint utolsó lépéseiben megvalósítva ezek ellenkezője igaz. A két esetet érdemes külön kezelni, mivel jelentős különbség tapasztalható közöttük. Először a legjobb esetet vizsgáljuk.

9. Lemma. Legjobb esetben az 7. Algoritmus l-dik szintjéhez tartozó iterációs ciklusok k_l számára teljesül, hogy

$$k_l \le \frac{s^n - 1}{s - 1} \left\lceil \eta \left\lfloor (\eta s^n)^l N_0 \right\rfloor \right\rceil. \tag{86}$$

Bizonyítás. A legjobb eset akkor következik be, ha a listaelemek legalább $1-\eta$ hányadának törlését az l-dik szint első lépéseiben sikerül elvégezni. Ez azt jelenti, hogy $\lfloor (1-\eta) N_l \rfloor$ intervallumot egészben sikerül kizárni a további vizsgálatokból, további $\lfloor s \cdot \mathrm{frac}((1-\eta) N_l) \rfloor$ részintervallumot az első irányban végzett szeletelése után ($\mathrm{frac}(\cdot)$ függvény a törtrész függvényt jelöli), és így tovább. A 9. Ábrán erre láthatunk példát, ahol a szint kezdetén 3 elem van a listán, és a szint során törlésre kerülő részintervallumokat szürkítés jelöli. A szint egyes vágásait az intervallumokba húzott egyenes szakaszok érzékeltetik. Ebben az esetben a gyorsító eljárások nélküli, összesen 12 iterációs ciklus helyett annak csupán felére, azaz 6 lépésre van szükség.



9. Ábra. Listaelemek törlése η-gyorsítással a legjobb esetben $N_l=3$ elem esetén, $s=3$, $n=2$ és $\eta=0{,}4$ értékek mellett.

Ezek a törlések a szint további iterációs ciklusaiból

$$\frac{s^n-1}{s-1}\lfloor(1-\eta)N_l\rfloor$$

darabot megtakarítanak azzal, hogy a kiindulási intervallumokból egészben törölnek $\lfloor(1-\eta)N_l\rfloor$ darabot, majd ezután az első irányú vágást követően

$$\frac{s^{n-1}-1}{s-1}\lfloor s\cdot\mathrm{frac}\big((1-\eta)N_l\big)\rfloor$$

további iterációt takarítanak meg azzal, hogy $\lfloor s\cdot\mathrm{frac}((1-\eta)N_l)\rfloor$ intervallumot már az első vágás után törölnek, és így tovább. A szint összesen elvégzendő iterációs ciklusainak számára ez azt jelenti, hogy az l-dik szint k_l összes iterációs ciklusainak száma legfeljebb

$$\frac{s^n-1}{s-1}-\frac{s^n-1}{s-1}\lfloor(1-\eta)N_l\rfloor-\frac{s^{n-1}-1}{s-1}\lfloor s\cdot\mathrm{frac}\big((1-\eta)N_l\big)\rfloor, \tag{87}$$

ami átrendezve a következőképpen írható fel:

$$\begin{aligned} k_l &\le \frac{s^n-1}{s-1}\lfloor\eta N_l\rfloor+\frac{s^{n-1}-1}{s-1}\lceil s\cdot\mathrm{frac}\big((1-\eta)N_l\big)\rceil\le \\ &\le \frac{s^n-1}{s-1}\lceil\eta N_l\rceil. \end{aligned} \tag{88}$$

Behelyettesítve (84)-et (88)-ba adódik (86). □

Az η feltevés mellett a 9. Lemmabeli (86)-nál jobb becslés az $n=1$ esetben nem adható, amennyiben a (84) által nyújtott korlát pontos. A becslés igaz ugyan magasabb dimenzióban is, sőt, a (86)-ban egyenlőség is előfordulhat tetszőleges magas n érték esetén. A 9. Ábrabeli példán azonban $k_l=6$ a pontos érték, míg a (86) által szolgáltatott korlát 8. Pontosabb eredményt is kaphatunk a 9. Lemma bizonyításában előforduló további tagok felhasználásával. Így speciálisan $n\le 2$ esetben igaz a következő:

7. Megjegyzés. Legjobb esetben a 7. Algoritmus l-dik szintjéhez tartozó iterációs ciklusok számára teljesül, hogy

$$k_l \le \frac{s^n-1}{s-1} - \frac{s^n-1}{s-1}\left\lfloor(1-\eta)\left\lfloor\eta s^n \ldots\left\lfloor\eta s^n\left\lfloor\eta s^n N_0\right\rfloor\ldots\right\rfloor\right\rfloor - \\ - \frac{s^{n-1}-1}{s-1}\left\lfloor s\cdot \mathrm{frac}\left((1-\eta)\left\lfloor\eta s^n \ldots\left\lfloor\eta s^n\left\lfloor\eta s^n N_0\right\rfloor\ldots\right\rfloor\right\rfloor\right)\right\rfloor, \tag{89}$$

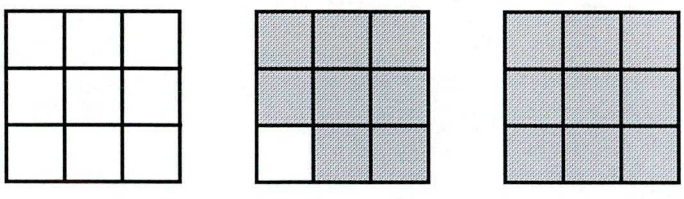
és kizárólag az η feltevésre támaszkodva $n \le 2$ esetben ennél jobb becslés nem adható.

Az 9. Ábrabeli példára felhasználva ezt az eredményt, már valóban pontos értéket kapunk k_l-re. Az $n>2$ esetben a pontosabb becsléshez további tagok szükségesek, melyek kiszámítása a (89) tagjaihoz hasonlóan történhet, és melyekkel a továbbiakban nem foglalkozunk. Vizsgáljuk most meg a legrosszabb esetet! Ezt a 9. Lemma által megfogalmazott legjobb esettel szemben a következő állítás írja le.

10. Lemma. Az 7. Algoritmus l-dik szintjéhez tartozó iterációs ciklusok k_l számára legrosszabb esetben is teljesül, hogy

$$k_l \le \frac{s^n-1}{s-1}\left\lfloor(\eta s^n)^l N_0\right\rfloor. \tag{90}$$

Bizonyítás. A legrosszabb eset akkor fordul elő, amikor az adott l-dik szint iterációs ciklusai során minden törlendő elem szélessége $s^{-(l+1)}\mathrm{w}(X)$, azaz a törlésekre az iterációs szint végrehajtásának utolsó szakaszában kerül sor. A 10. Ábrán a kezdőlista elemei láthatók, a vágások helyét egyenes szakaszok, míg az adott szinten törlésre kerülő elemeket szürkítés jelöli. Más szóval a szint kezdetekor a listán szereplő minden elemre az összes koordináta-irányra merőlegesen minden vágást el kell végezni, így a törlésekből származó gyorsítást csak a következő szint lecsökkent kezdőlistája jelenti.

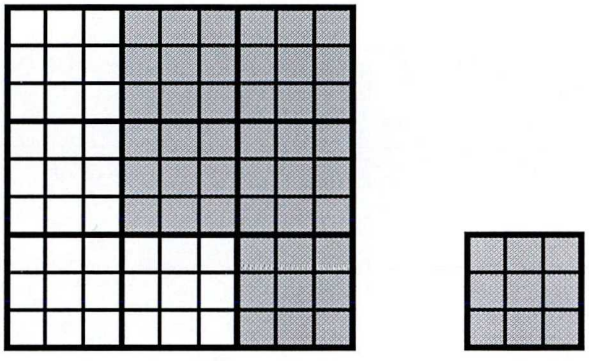


10. Ábra. Listaelemek törlése η-gyorsítással a legrosszabb esetben N_l=3 elem esetén, s=3, n=2 és η=0,4 értékek mellett.

Az l-dik iterációs szintre érve (84) szerint legfeljebb $\lfloor(\eta s^n)^l N_0\rfloor$ darab elem várakozik a listán, melynek mindegyike a 18. Tétel (60) egyenlősége alapján kiszá-

mítható számú iterációs ciklust igényel a feldarabolásához. Ezen két tényező összeszorzásával kapjuk a lemma állítását. □

A legrosszabb eset elemzésekor egyetlen szint elszigetelt vizsgálata nem mutat eltérést az η-gyorsított esethez képest: a 10. Ábrán szereplő példán képleteink általános (a 18. Tétel (60) képlete alapján) és η-gyorsított (a 10. Lemma (90) képlete egyenlőség teljesülésekor) legrosszabb esetben is k_l=12 értéket szolgáltatnak, ami egyébként megfelel a pontos értéknek. Megvizsgálva azonban az innen következő szintre (ld. 11. Ábra) az egyes képletek pontosságát már szembetűnő az eltérés. Míg (60) alapján az eredmény k_{l+1}=108, addig a pontos és a (90) alapján kalkulált éles érték is k_{l+1}=40.



11. Ábra. Listaelemek törlése η-gyorsítással a legrosszabb esetben a 10. Ábrán vázolt szintet követő iterációs szinten (s=3, n=2 és η=0,4 értékek mellett).

Mivel a konvergencia-sebesség kiszámításakor általános esetben a legrosszabb kimenetelű működést kell figyelembe vennünk, a továbbiakban a 10. Lemma felhasználásával jellemezzük az 7. Algoritmus adott szintjéig elvégzendő legfeljebb szükséges iterációs ciklusokat.

25. Tétel. Ha a k-dik iterációs ciklus az 7. Algoritmus l-dik szintjén található és $\lceil \eta s^n \rceil > 1$, akkor

$$k \le \frac{\left(\lceil \eta s^n \rceil^{l+1} - 1\right)\left(s^n - 1\right)}{\left(\lceil \eta s^n \rceil - 1\right)(s-1)} N_0 . \tag{91}$$

Az $\lceil \eta s^n \rceil$=1 esetben

$$k \le \frac{s^n - 1}{s-1}(l+1)N_0 . \tag{92}$$

Bizonyítás. Az 10. Lemma felső korlátot ad az l-dik szinten elvégzendő k_l iterációszámra. Összegezzük tehát ezt az eredményt az l-dik szint végéig:

$$\sum_{i=0}^{l} k_i \le \frac{s^n-1}{s-1}\sum_{i=0}^{l}\left\lfloor(\eta s^n)^i N_0\right\rfloor \le \frac{s^n-1}{s-1}N_0\sum_{i=0}^{l}\left\lceil \eta s^n\right\rceil^i. \tag{93}$$

Amennyiben $\lceil \eta s^n \rceil > 1$, akkor az összegzés eredménye (91)-et szolgáltatja. Az $\lceil \eta s^n \rceil = 1$ esetben az összegzés eredménye egyszerűen $l+1$. Ezzel az állítás mindkét esetét bebizonyítottuk. □

Az l-dik szint iterációs ciklusainak jellemzésekor nem tudunk a 19. Tétel (62) eredményében szereplő alsó korlátnál jobbat megadni, mivel az egyes iterációs szintek végeztével a listán maradó elemek arányára a feltevés alapján csak felső becslés áll rendelkezésre. Másfelől egy alsó becslés rögzítése az η-gyorsító eljárás fogalmának alkalmazhatóságát korlátozná, így ilyen kikötést tenni értelmetlen. A 25. Tétel (91) egyenlőtlensége azonban elegendő ahhoz, hogy a gyorsító eljárással kiegészített uniform intervallum-felosztási eljárások konvergencia-sebességének nagyságrendjét megbecsülhessük.

26. Tétel. Legyen F az f valós függvénynek α-konvergens befoglalófüggvénye az X intervallum felett, továbbá jelölje L az aktuális iterációs ciklus listáját, A pedig az aktuális intervallumát. Ekkor a 7. Algoritmus l-dik szintjén lévő bármely k-dik iterációs ciklusában teljesül, hogy

$$\mathrm{lb}(f(X)) - \min_{Y\in L\cup\{A\}}(\mathrm{lb}\,F(Y)) = \mathcal{O}(\eta^{\alpha l/n} s^{\alpha(n-1)/n} k^{-\alpha/n}), \tag{94}$$

ha $\lceil \eta s^n \rceil > 1$.
Amennyiben $\lceil \eta s^n \rceil = 1$, a műveletigény a következő:

$$\mathrm{lb}(f(X)) - \min_{Y\in L\cup\{A\}} \mathrm{lb}(F(Y)) = \mathcal{O}(s^{-\alpha l/n} k^{-\alpha l/n} l^{\alpha l/n}) = \mathcal{O}\left(\left(\frac{l}{sk}\right)^{\frac{\alpha l}{n}}\right) \tag{95}$$

Bizonyítás. Tegyük fel először, hogy

$$\lceil \eta s^n \rceil \ge 2. \tag{96}$$

Mivel $\lceil \eta s^n \rceil$ pozitív és az $\lceil \eta s^n \rceil = 1$ esetet külön vizsgáljuk, a (96) feltétel mindig teljesül. Mivel $s>1$ és $N_0>0$, így a 25. Tétel (91) egyenlőtlenségét átalakíthatjuk a következő módon:⌈⌉

$$\frac{k(s-1)(\eta s^n-1)}{N_0(s^n-1)}+1 \le \frac{k(s-1)\left(\lceil \eta s^n \rceil - 1\right)}{N_0(s^n-1)}+1 \le \lceil \eta s^n \rceil^{l+1}. \tag{97}$$

A továbbiakban felső korlátot adunk $\lceil \eta s^n \rceil^{l+1}$ értékére. Nyilván $\lceil \eta s^n \rceil < \eta s^n + 1$, és a (96) feltétel miatt $\eta s^n > 1$. Ezekből következik, hogy $\eta s^n + 1 < 2\eta s^n$. Folytatva a gondolatsort

$$\lceil \eta s^n \rceil^{l+1} < (2\eta s^n)^{l+1} = (2\eta)^{l+1}(s^l s)^n. \tag{98}$$

A (97) és (98) egyenlőtlenségek együttesen felső korlátot szolgáltatnak s^{-l} értékére:

$$s^{-l} \le s\left(\frac{\frac{k(s-1)(\eta s^n-1)}{N_0(s^n-1)}+1}{(2\eta)^{l+1}}\right)^{-\frac{1}{n}}. \tag{99}$$

Mivel a 7. Algoritmus is Hansen intervallum-kiválasztási szabályát követi, így értelmes rá a 12. Definíció, mely alapján

$$w(A)=w_{max}\le s^{-l}w(X)$$

teljesül az l-dik szint minden egyes A aktuális intervallumára. Mivel az L listában tárolt minden egyes Y elemnek a szélessége legfeljebb akkora, mint az aktuális intervallumé, azon Y^* intervallum szélességére, melyen a $\min_{Y\in L\cup A} \mathrm{lb}F(Y)$ minimum felvétetik, szintén igaz a következő egyenlőtlenség:

$$w(Y^*)\le s^{-l}w(X). \tag{100}$$

Alkalmazva (100)-at az 5. Lemmára kapjuk, hogy speciálisan

$$\mathrm{lb}(f(X))-\min_{Y\in L\cup A}\mathrm{lb}(F(Y))\le c w^{\alpha}(Y^*)\le c(s^{-l}w(X))^{\alpha}. \tag{101}$$

Folytassuk most a (101) egyenlőtlenségsort (99) behelyettesítésével:

$$c(s^{-l}w(X))^{\alpha} < c\left(s\,w(X)\left(\frac{\frac{k(s-1)(\eta s^n-1)}{N_0(s^n-1)}+1}{(2\eta)^{l+1}}\right)^{-\frac{1}{n}}\right)^{\alpha}.$$

Az egyes tényezők nagyságrendje alapján ezzel bizonyítottuk a tétel első állítását, azaz a (94) nagyságrendet.

Amennyiben $\lceil \eta s^n \rceil=1$, a 25. Tétel (92) egyenlőtlensége érvényes, melynek átrendezésével azt kapjuk, hogy

$$s^{-l} \le \left(\frac{k(s-1)}{(l+1)N_0}+1\right)^{-\frac{1}{n}}. \tag{102}$$

Tételünk első állításának bizonyításában taglaltak szerint (102) (101)-be való behelyettesítésével a tétel bizonyítható. Ebben az esetben a behelyettesítés eredménye a következő lesz:

$$\mathrm{lb}(f(X)) - \min_{Y \in L \cup A} \mathrm{lb}(F(Y)) \le c\left(\frac{k(s-1)}{(l+1)N_0} + 1\right)^{-\frac{\alpha l}{n}} \mathrm{w}(X)^{\alpha},$$

mellyel (95)-öt, és ezzel a tételt bizonyítottuk. □

Vegyük észre, hogy a gyorsítás nélküli szeletelő algoritmus konvergencia-sebességét vizsgáló 20. Tétel (64) egyenlőtlenségében szereplő felső korlátban az s és k tényezők nagyságrendje megegyezik a gyorsító eljárásokat is figyelembe vevő 26. Tétel (94) egyenlőségében megadottakkal. A nem állandó tagokban való egyetlen különbözőség az η tényező jelenléte. Ez az együttható az η-gyorsítás „gyengeségét" adja meg, azaz minél nagyobb η értéke, annál gyengébb a globális optimum közelítése adott rögzített számú iterációs ciklust követően. Az $\lceil \eta s^n \rceil = 1$ eset igen speciális, és csak ritkán fordul elő. Másrészt a 26. Tételbeli felső korlátok közül a (95)-beli mindig élesebb, mint a (94)-beli, így minden esetben alkalmazható az utóbbi. A speciális esetet csupán a teljesség kedvéért vizsgáltuk meg.

Összegezve a 26. Tétel eredményeit, az η-gyorsított szeletelő eljárás konvergencia-sebességének η-tól való függését az $\eta^{\alpha l/n}$ tényezővel jellemezhetjük. Ez azt jelenti, hogy rögzített állandójú α-konvergencia és változószám (dimenzió) esetén a gyorsító eljárásaink javítása magán az algoritmuson polinomiális gyorsulást eredményez. Minél magasabb iterációs szinten jár az algoritmus a feladat megoldásában, annál erősebb a gyorsító eljárások hatása.

A gyorsított eset vizsgálatakor a legjobb esettel egyáltalában nem foglalkoztunk az η-gyorsítás természetére való tekintettel (vö. a 25. Tétel utáni megjegyzés).

6. Összefoglalás

Jelen dolgozat a globális optimalizálásban, azon belül is az intervallum-felosztási eljárások terén az utóbbi néhány évben végzett kutatásaim során elért főbb eredményeket tartalmazza. Az 1. fejezetben a globális optimalizálás néhány hagyományos módszerét ismertettem röviden, a 2. fejezetben az intervallum analízis leírását közöltem.

A 3. fejezetben a korlátozás és szétválasztás elvén alapuló intervallum-felosztási globális optimalizáló eljárásokat taglaltam. Ehhez megadtam egy modellalgoritmust (2. Algoritmus), melynek egyes lépéseit pontosan definiálva megkapható minden, a vizsgált eljárás-osztálybeli algoritmus. A modell iteratív része 5 lépésből áll (2-6. lépés), melyeket az egyes alfejezetekben külön-külön megvizsgáltam, utalva az egyes lépések közötti kölcsönhatásokra. A 3.1. alfejezetben ismertettem az irodalomban fellelhető felosztási szabályokat.

A 3.2. alfejezetben egy eddig kevéssé vizsgált területet, az intervallum-felosztási módszerek listakezelését tekintettem át ([17, 20]). Ezen belül megadtam a rendezetlen (5. Tétel), a rendezett (6. Tétel) és a hibrid (7. Tétel) listakezelő módszerek műveletigényét legrosszabb esetben. Utóbbi egy olyan általam definiált új listakezelési módszer, mely az eddigi vegyes listakezelő módszereknél hatékonyabb. Igazoltam továbbá (8. Tétel), hogy a rendezett és a hibrid listakezelés műveletigényére kapott eredmény legrosszabb esetben optimális. A 3.2.4. és 3.2.5. alfejezetekben röviden bemutattam a Hansen algoritmus listakezelését, valamint a fenti listakezelési módszerek legjobb esetben való viselkedését.

A 3.3. alfejezetben foglaltam össze a gyorsító eljárásokat, melyben megadtam az intervallum-felosztási modell egy iterációs ciklusában szükséges függvényhívások számát is (1. Lemma). A gyorsító eljárások új megközelítése, mely a szintén új, szeletelő technikát is tartalmazza, külön, az 5. fejezetben került ismertetésre.

A 3.4. alfejezetben az intervallum-kiválasztási szabályok közül legelterjedtebb Moore-Skelboe és Hansen kiválasztási szabályt vizsgáltam ([9, 10,11, 12, 14]). Míg a Moore-Skelboe szabály esetében az algoritmusok konvergencia-sebessége ismert, addig az iterációs szintek fogalmának bevezetésével (11. Definíció, 2-4. Lemma) sikerült hasonló eredményt bizonyítanom Hansen algoritmusára is (13. és 14. Tétel). A legjobb eset vizsgálatok során tapasztalt különbségeket numerikus eredményekkel is alátámasztottam.

A modellalgoritmus iterációs része utolsó lépését, a megállási feltételt vizsgáltam a 3.5. alfejezetben ([18, 19, 21]). Ebben az irányban végzett kutatásaim alapját a Lipschitz-folytonos célfüggvények optimalizálása adta. Az optimumérték és optimumhely befoglalására különböző megállási feltételeket szokás alkalmazni. A

3.5.1. alfejezetben bizonyítottam, hogy Hansen algoritmusát alkalmazva is adható korlát az optimumérték befoglalására vonatkozóan, és egy ilyen felső korlátot közöltem is (15. Tétel, 6. Lemma). Továbbá, megadtam egy felső korlátot az adott pontosságú befoglalás eléréséhez szükséges iterációs szintek minimális számára vonatkozóan (16. Tétel). A 3.5.2. alfejezetben megmutattam, hogy az optimumhely befoglalására törekvő megállási feltételek alkalmazásakor is adható becslés az optimum értékére mind Hansen (17. Tétel), mind Moore és Skelboe (5. Következmény) algoritmusa esetében.

A 3.6. alfejezetben példát adtam az intervallum-felosztási eljárások egy lényeges felhasználási területére ([6, 7, 8]). Más globális optimalizáló módszerektől eltérően az intervallum-felosztási eljárások jelen dolgozatban vizsgált osztálya bizonyító erejű megoldást szolgáltat. Az alfejezetben egy szeparációs feladat kapcsán vegyipari rendszerek tervezésénél felmerült elvi problémák megoldását adtam meg. Az úgynevezett recirkuláció és redundancia feltevés cáfolatát optimális ellenpéldák létezésével bizonyítottam, ahol az optimalitást a megoldásokhoz felhasznált intervallum-felosztási módszerek biztosították.

A 4. fejezetben egy új trend továbbfejlesztését írtam le, mely során az intervallum-felosztás kettőnél több darabra történik egy-egy iterációs cikluson belül ([15, 16, 22, 26, 27]). Ez előállítható egyirányú szeleteléssel vagy többszörös vágással (ill. többszörös szeleteléssel). A 4.1. alfejezetben definiáltam az eddig nem vizsgált szeletelő algoritmust és kiterjesztettem rá az iterációs szintek fogalmát (12. Definíció, 7. Lemma, 18. és 19. Tétel). Részletesen megvizsgáltam a szeletelő algoritmus konvergencia tulajdonságait (20-23. Tételek), és numerikus teszteredményekkel alá is támasztottam a kapott elméleti eredményeket. A 4.2. alfejezetben a többszörös szelést alkalmazó algoritmusok konvergencia-sebességét határoztam meg (24. Tétel).

Az 5. fejezetben a gyorsító eljárások egységes kezelésére definiáltam az η-gyorsító eljárás fogalmát. Az η feltevést, mely szerint megadható a szeletelő eljárások esetében olyan arány, melynél kevesebb listaelemet a gyorsító eljárások nem törölnek — egy bizonyos küszöbön túl — egyetlen iterációs szinten sem, numerikus eredményekkel támasztottam alá. Az η feltevés felhasználásával az eddigi legrosszabb eset vizsgálatokhoz képest esetenként több nagyságrenddel pontosabb becslést sikerült megadnom a szeletelő és speciális esetben az intervallumfelező eljárások konvergencia-sebességére vonatkozóan (26. Tétel).

A távlati kutatások közül a legfontosabb a gyorsító eljárások további elemzése és az algoritmusok egyes részei kölcsönhatásának részletes vizsgálata. Az algoritmusok hatékonyságának további növelése a befoglaló függvények javításával fokozható.

Summary

The present work consists of the main results I have achieved in the last few years in the field of global optimization, especially in the interval subdivision methods. The core of this work falls into five sections. Chapter 1 is a brief summary of some standard methods for global optimization, and chapter 2 describes interval analysis.

Chapter 3 discusses the interval subdivision methods for global optimization based on the branch-and-bound principle. In order to do so I give a model algorithm (Algorithm 2). All of the algorithms of the investigated class of methods can be obtained by defining the individual steps of this model rigorously. The iterative part of the model consists of five steps (step 2 to step 6) which I investigate in certain sections separately, also referring to their interactions. In section 3.1 I review the subdivision rules known from the literature.

Section 3.2 analyses the storage handling of the interval subdivision methods ([17, 20]), a less investigated field up to the present. There I prove the worst case performance of the unordered (Theorem 5), the ordered (Theorem 6), and the hybrid (Theorem 7) storage handling methods. The latter method is a new one defined here which is more efficient than other mixed storage handling methods. Furthermore, I prove (Theorem 8) that the results for the ordered and the hybrid methods are optimal in the worst case. In subsections 3.2.4 and 3.2.5 I briefly discuss the storage handling of Hansen's algorithm, and the best case behavior of the methods listed above.

Section 3.3 gives a description on the accelerating devices where I prove the number of function evaluations needed to complete a single iteration cycle (Lemma 1). A new theoretical approach for considering accelerating devices containing the also new multisplitting technique is given in chapter 5.

Section 3.4 investigates the two most wide-spread interval selection rules, the Moore-Skelboe and the Hansen rule ([9, 10, 11, 12, 14]). While for the Moore-Skelboe rule the convergence speed of the algorithms is known, introducing the notion of iteration levels (Definition 11, Lemmas 2-4) I could prove similar results for the algorithm of Hansen (Theorems 13 and 14). The differences occurring in the best case are analyzed also using numerical examples.

The last step of the iterative part of the model algorithm, the termination criterion is investigated in section 3.5 ([18, 19, 21]). The basis of my investigations in this direction is the optimization of Lipschitz continuous functions. It is common to apply different termination criteria for inclusions of the optimum value and the optimum points. In subsection 3.5.1 I prove that a bound for the inclusion of the optimum value can also be given for Hansen's algorithm, and I also provide such a bound (Theorem 15, Lemma 6). Furthermore, I give an upper bound on the minimal number of iteration levels that are necessary to achieve a given accuracy

(Theorem 16). In subsection 3.5.2 I show that using termination criteria to determine the inclusion of the optimum points it is also possible to give inclusions for the optimum value applying Hansen's (Theorem 17) or Moore and Skelboe's (Corollary 5) algorithm, as well.

Section 3.6 gives an example for an important application of interval subdivision methods ([6, 7, 8]). In contrast to other global optimization methods the class of interval subdivision algorithms investigated in the present work provide reliable computational proofs. In the section I give the solution for a problem of principle in connection with separation network synthesis arising when planning chemical systems. I prove the denial of the so-called recycling and redundancy assumptions by proving the existence of optimal counter examples, where the optimality is assured by the interval subdivision methods.

Chapter 4 consists of the development of a new trend, i.e., the intervals are subdivided into more than two subintervals within a single iteration cycle ([15, 16, 22, 26, 27]). This can be obtained by slicing in one direction or using multisection. In section 4.1 I define the new multisplitting algorithm and extend the notion of iteration levels for it (Definition 12, Lemma 7, Theorems 18 and 19). I investigate the convergence properties of the multisplitting algorithm in details (Theorems 20 to 23), and support the theoretical results with numerical tests. In section 4.2 I determine the convergence rate of algorithms using uniform multisection (Theorem 24).

In chapter 5 I define the notion of the η-accelerating device in order to give a uniform way for handling acceleration devices. The η assumption — supposing the existence of a rate for at least deleted list elements at each iteration level from a certain level on — is supported by numerical tests. With the aid of the η assumption I provide new bounds for the convergence speed of multisplitting and in the special case for bisection methods, which bounds are better by orders of magnitude in certain cases (Theorem 26).

The most important parts in future investigations are the additional analysis of accelerating devices and the interaction of the different steps of the model algorithm. Further increase of the algorithms' efficiency can be achieved by improving the inclusion functions.

Irodalomjegyzék

[1] Alefeld, G., J. Herzberger, *Einführung in die Intervallrechnung,* Bibliographisches Institut, Mannheim, 1974.

[2] Baumann, E., *Globale Optimierung stetig differenzierbarer Funktionen einer Vaiablen,* Freiburger Intervall-Berichte 86/6, Institut für Angewandte Mathematik, Universität Freiburg, 1986.

[3] Berner, S., *Ein paralleles Verfahren zur verifizierten globalen Optimierung,* Dissertation, Universität Wuppertal, 1995.

[4] Bomze, I.M., T. Csendes, R. Horst, and P.M. Pardalos, (eds.), *Developments in Global Optimization,* Kluwer Academic Publishers, Dordrecht, 1997.

[5] Cormen, T.H., Ch.E. Leiserson, R.L. Rivest, *Introduction to Algorithms,* MIT Press – McGraw-Hill, Cambridge, Massachusetts, 1990.

[6] Csallner, A.E., *On the Global Optimization in Chemical Process Design,* Proceedings of the Fourth Bulgarian-Hungarian Workshop on Chemical Engineering, pp. 10-12, Varna, Bulgaria, 1992.

[7] Csallner, A.E., *Global Optimization in Separation Network Synthesis,* Hungarian Journal of Industrial Chemistry, **21**, pp. 303-308, 1993.

[8] Csallner, A.E., *Global Optimization in Separation Network Synthesis,* Second International Workshop on Mathematical Modelling in Chemical Engineering, Budapest, p. 3, 1993.

[9] Csallner, A.E., *The Role of the Interval Selection Rule in the Interval Bisection Methods,* Scientific Computation and Mathematical Modelling, pp. 7-10, Datecs Publishing, Sofia, 1993.

[10] Csallner, A.E., *Intervallum-felező globális optimalizáló módszerek összehasonlítása,* XXI. Magyar Operációkutatási Konferencia, 8. o., Szeged, 1993.

[11] Csallner, A.E., *Large Interval Selection Rule and the Convergence Speed in Interval Methods,* XIIth International Conference on Mathematical Programming, p. 6, Mátrafüred, 1994.

[12] Csallner, A.E., T. Csendes, *Convergence Speed of Interval Methods for Global Optimization,* 6th International Conference on Numerical Methods, p. 14, Miskolc, 1994.

[13] Csallner, A.E., T. Csendes, *Convergence Speed of Interval Methods for Global Optimization and the Joint Effects of Algorithmic Modifications,* SCAN-95 IMACS/GAMM Conference, p. 33, Wuppertal, 1995.

[14] Csallner, A.E., T. Csendes, *The Convergence Speed of Interval Methods for Global Optimization,* Computers and Mathematics with Applications, **31**, pp. 173-178, 1996.

[15] Csallner, A.E., T. Csendes, M.Cs. Markót, *Convergence Properties for Multisection Interval Methods for Global Optimization,* SCAN-97 IMACS/GAMM Conference, pp. V5-V8, Lyon, 1997.

[16] Csallner, A.E., T. Csendes, M.Cs. Markót, *Szeletelő intervallum-felosztási globális optimalizáló eljárások konvergenciatulajdonságai,* XXIII. Magyar Operációkutatási Konferencia, p. 8, Pécs, 1997.

[17] Csallner, A.E., M.Cs. Markót, *Improving Interval Methods for Global Optimization,* CSCS Conference, p. 29, Szeged, 1998.

[18] Csallner, A.E., T. Csendes, *A Study on the Termination of Interval Subdivision Methods for Global Optimization,* Conference on Numerical Methods and Computational Mechanics, pp. 20-21, Miskolc, 1998.

[19] Csallner, A.E., M.Cs. Markót, *Termination Criteria and the Objective Function's Lipschitz Continuity by Interval Subdivision Methods,* IMACS/GAMM SCAN-98 Conference, pp. 25-26, Budapest, 1998.

[20] Csallner, A.E., *Improving Storage Handling of Interval Methods for Global Optimization,* Acta Cybernetica, **13**, pp. 413-421, 1998.

[21] Csallner, A.E., *Lipschitz Continuity and the Termination of Interval Methods for Global Optimization*, 14 oldal, közlésre elfogadva a Computers and Mathematics with Applications c. folyóiratba.

[22] Csallner, A.E., T. Csendes, M.Cs. Markót, *Multisection in Interval Branch-and-Bound Methods for Global Optimization,* 32 oldal, közlésre benyújtva.

[23] Csendes, T., *Nonlinear Parameter Estimation by Global Optimization — Efficiency and Reliabilility,* Acta Cybernetica, **8**, pp. 361-370, 1988.

[24] Csendes, T., J. Pintér, *The Impact of Accelerating Tools on the Interval Subdivision Algorithm for Global Optimization,* European Journal on Operational Research, **65**, pp. 314-320, 1993.

[25] Csendes, T., D. Ratz, *Subdivision Direction Selection in Interval Methods for Global Optimization,* SIAM Journal on Numerical Analysis, **34**, pp. 922-938, 1997.

[26] Csendes, T., M.Cs. Markót, A.E. Csallner, *Multisection in Interval Methods for Global Optimization,* Abstracts of the ismp97, p. 69, Lausanne, Switzerland, 1997.

[27] Csendes, T., A.E. Csallner, M.Cs. Markót, *Multisection in Interval Methods for Global Optimization,* SCAN-97 IMACS/GAMM Conference, pp. V9-V12, Lyon, 1997.

[28] Dixon, L.C.W., G.P. Szegő, *Towards Global Optimisation,* North-Holland, Amsterdam, 1975.

[29] Eriksson, J., *Parallel Global Optimization Using Interval Analysis*, Ph.D. Thesis, University of Umeå, Sweden, 1991.

[30] Fischer, H.-C., *Schnelle automatische Differentiation, Einschließungs-methoden und Anwendungen,* Dissertation, Universität Karlsruhe, 1990.

[31] Floudas, C., *Separation Synthesis of Multicomponent Feed Streams into Multicomponent Product Streams*, AIChE Journal, **33**, pp. 540-556, 1987.

[32] Floudas, C., P.M. Pardalos, (eds.), *Recent Advances in Global Opimization,* Princeton University Press, Princeton, 1992.

[33] Halász, L., F. Kátai, A.E. Csallner, G. Almásy, *On the Solution of Nonlinear Equations in Process Simulation,* Proceedings of the Fourth Bulgarian-Hungarian Workshop on Chemical Engineering, pp. 25-27, Varna, 1992.

[34] Hansen, E.R., *Global Optimization Using Interval Analysis — The One-Dimensional Case,* Journal on Optimization Theory and Applications, **29**, pp. 331-344, 1979.

[35] Hansen, E.R., *Global Optimization Using Interval Analysis — The Multidimensional Case,* Numerische Mathematik, **34**, pp. 247-270, 1980.

[36] Hansen, E.R., *Global Optimization Using Interval Analysis*, Marcel Dekker, New York, 1992.

[37] Hansen, E.R., S. Sengupta, *Global Constrained Optimization Using Interval Analysis*, In: Michel, K. (ed.), Interval Mathematics 1980, Academic Press, New York, pp. 25-47, 1980.

[38] Horst, R., P.M. Pardalos, N.V. Thoai, *Introduction to Global Optimization,* Kluwer Academic Publishers, Dordrecht, 1995.

[39] Ichida, K., J. Fujii, *An Interval Arithmetic Method For Global Optimization*, Computing, **23**, pp. 85-97, 1979.

[40] Kearfott, R.B., *Rigorous global search: continuous problems,* Kluwer Academic Publishers, Dordrecht, 1996.

[41] Krawczyk, R., A. Neumaier, *Interval Slopes for Rational Functions and Associated Centered Forms,* SIAM Journal on Numerical Analysis, **22**, pp. 604-616, 1985.

[42] Krawczyk, R., K. Nickel, *Die zentrische Form in der Intervallarithmetik, ihre quadratische Konvergenz und ihre Inklusionsisotonie,* Computing, **28**, pp. 117-132, 1982.

[43] Kovács, Z., F. Friedler, L.T. Fan, *Recycling in a Separation Process Structure*, AIChE Journal, **39**(6), pp. 1087-1089, 1993.

[44] Levy, A.V., A. Montalvo, S. Gomez, A. Calderon, *Topics in Global Optimization,* Lecture Notes in Mathematics #909, Springer, Berlin, 1981.

[45] Mágoriné Huhn, Á., A.E. Csallner, *Drawing Fractals in Maple*, Proceedings of the 3rd International Conference on Applied Informatics, **1**, pp. 263-275, 1997.

[46] Moore, R.E., *Interval Arithmetic and Automatic Error Analysis in Digital Computation,* Ph.D. Thesis, Stanford University, 1962.

[47] Moore, R.E., *Interval Analysis,* Prentice-Hall, Englewood Cliffs NJ, 1966.

[48] Moré, J.J., B.S. Garbow, K.E. Hillstrom, *Testing Unconstrained Optimization Software,* ACM Trans. Math. Software, **7**, pp. 17-41, 1981.

[49] Pintér, J., *Lipschitzian Global Opimization: Some Prospective Applications,* In: Floudas, C., P.M. Pardalos, (eds.), *Recent Advances in Global Opimization,* Princeton University Press, Princeton, 1992.

[50] Pardalos, P.M., S.A. Vavasis, *Quadratic Programming with One Negative Eigenvalue is NP-Hard,* Journal on Global Optimization, **1**, pp. 15-22, 1991.

[51] Piyavskii, S.A., *An Algorithm for Finding the Absolute Extremum of a Function,* USSR Computational Mathematics and Mathematical Physics, **12**, pp. 57-67, 1972.

[52] Rall, L.B., *Automatic Differentiation, Techniques and Applications,* Lecture Notes in Computer Science, **120**, Springer-Verlag, Berlin, 1981.

[53] Ratschek, H., J. Rokne, *Computer Methods for the Range of Functions,* Ellis Horwood, Chichester, 1984.

[54] Ratschek, H., J. Rokne, *Efficiency of a Global Optimization Algorithm,* SIAM Journal on Numerical Analysis, **24**, pp. 1191-1201, 1987.

[55] Ratschek, H., J. Rokne, *New Computer Methods for Global Optimization,* Ellis Horwood, Chichester, 1988.

[56] Ratschek H., J. Rokne, *Interval Methods*, In: Horst R., P.M. Pardalos (eds.), Handbook of Global Optimization, Kluwer, Dordrecht, pp. 751-828, 1993.

[57] Ratz, D., *Automatische Ergebnisverifikation bei globalen Optimierungsproblemen,* Dissertation, Universität Karlsruhe, 1992.

[58] Schwefel, H., *Numerical Optmization of Computer Models,* Wiley, New York, 1991.

[59] Skelboe, S., *Computation of Rational Interval Functions,* BIT, **14**, pp. 87-95, 1974.

[60] Sunaga, T., *Theory of an Interval Algebra and its Application to Numerical Analysis,* RAAG Memoirs, **2**, pp. 547-564, 1958.

[61] Törn, A., A. Žilinskas, *Global Optimization,* Springer-Verlag, Berlin, 1989.

[62] Walster, G., E. Hansen, S. Sengupta, *Test Results for a Global Optimization Algorithm,* In: Boggs, P., R. Byrd, R. Schnabel (eds.), *Numerical Optimization 1984*, SIAM, Philadelphia, pp. 272-287, 1985.

[63] Warmus, M., *Calculus of Approximations,* Bull. Acad. Polon. Sci. Cl. III, **4**, pp. 253-259, 1956.

Tárgymutató

Függelék

A dolgozatbeli táblázatokban szereplő futtatási teszteredményekhez felhasznált függvények szinte kivétel nélkül a szakirodalomból jól ismert, más megoldók teszteléséhez is széleskörben alkalmazott problémákból állnak. A következőkben megadjuk a függvények alakját és az irodalomban egy-egy jellemző előfordulásukat.

Booth [44]

$f(x)=(x_1+2x_2-7)^2+(2x_1+x_2-5)^2$, $x\in\mathbf{R}^2$.
Kiindulási intervallum (X): $[-5\cdot10^6,\ 5\cdot10^6]^2$.

Branin (Br) [61]

$$f(x)=\left(\frac{5}{\pi}x_1-\frac{5{,}1}{4\pi^2}x_1^2+x_2-6\right)^2+10\left(1-\frac{1}{8\pi}\right)\cos x_1+10,\ x\in\mathbf{R}^2.$$

Kiindulási intervallum (X): [-5, 10]×[0, 15].

Dripstone Cave (DSC*n*) [14]

$$f(x)=\left(\frac{1}{n}\sum_{i=1}^{n}\sin(x_i)\right)\left(1-\frac{4}{441\pi^2 n}\sum_{i=1}^{n}\left(x_i-19\frac{\pi}{2}\right)^2\right),\ x\in\mathbf{R}^n.$$

Kiindulási intervallum (X): $[0,\ 63]^n$.

Griewank5 (G5) [61]

$$f(x)=\sum_{i=1}^{5}\frac{x_i^2}{400}-\prod_{i=1}^{5}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1,\ x\in\mathbf{R}^5.$$

Kiindulási intervallum (X): $[-500,\ 600]^5$.

Hartman3 (H3) [61]

$$f(x)=-\sum_{i=1}^{4}c_i\exp\left(-\sum_{j=1}^{3}\alpha_{ij}(x_j-p_{ij})^2\right),\ x\in\mathbf{R}^3,\ \text{ahol}$$

i	α_i	c_i	p_i
1	(3,0, 10,0, 30,0)	1,0	(0,36890, 0,11700, 0,26730)
2	(0,1, 10,0, 35,0)	1,2	(0,46990, 0,43870, 0,74700)
3	(3,0, 10,0, 30,0)	3,0	(0,10910, 0,87320, 0,55470)
4	(0,1, 10,0, 35,0)	3,2	(0,03815, 0,57430, 0,88280)

Kiindulási intervallum (X): $[0, 1]^3$.

Levy1 [48]

$f(x)=x_1^6-15x_1^4+27x_1^2+250$, $x\in\mathbf{R}^1$.
Kiindulási intervallum (X): $[-4, 4]$.

Levy2 [48]

$$f(x)=\sum_{i=1}^{5} i\cdot\cos\left((i+1)x_1+i\right),\ x\in\mathbf{R}^1.$$
Kiindulási intervallum (X): $[-10, 10]$.

Levy8 (L8) [44]

$$f(x)=\sum_{i=1}^{n-1}(y_i-1)^2\left(1+10\sin^2(\pi y_{i+1})\right)+\sin^2(\pi y_1)+(y_n-1)^2,$$
ahol $y_i=1+(x_i-1)/4$, $i=1,\ldots,n$, $n=3$, $x\in\mathbf{R}^n$.
Kiindulási intervallum (X): $[-10, 10]^n$.

Matyas [44]

$f(x)=0{,}26(x_1^2+x_2^2)-0{,}48x_1x_2$, $x\in\mathbf{R}^2$.
Kiindulási intervallum (X): $[-10, 10]^2$.

Powell [44]

$f(x)=(x_1+10x_2)^2+5(x_3-x_4)^2+(x_2-2x_3)^4+10(x_1-x_4)^4$, $x\in\mathbf{R}^4$.
Kiindulási intervallum (X): $[-4, 6]^4$.

Rosenbrock (Rb) [44]

$f(x)=100(x_2-x_1^2)^2+(x_1-1)^2$, $x\in\mathbf{R}^2$.
Kiindulási intervallum (X): $[-600, 400]^2$.

Schwefel2.1 (Sch2.1) [58]

$f(x)=(1{,}5-x_1+x_1x_2)^2+(2{,}25-x_1+x_1x_2^2)^2+(2{,}625-x_1+x_1x_2^3)^2$, $x\in\mathbf{R}^2$.
Kiindulási intervallum (X): $[-1{,}5, 7{,}5]\times[-4, 5]$.

Schwefel2.7 (Sch2.7) [62]

$$f(x) = \sum_{i=1}^{10}\left(\exp(-0{,}1ix_1) - \exp(-0{,}1ix_2) - \left(\exp(-0{,}1i) - \exp(-i)\right)x_3\right)^2$$

$x \in \mathbf{R}^3$.

Kiindulási intervallum (X): [0, 5]×[8, 11]×[0,5, 3]

SEPNET1 [7]

$\min C(x)=(S_{1A}+S_{1B}+S_{1C})^{0,6}+(S_{2A}+S_{2B}+S_{2C})^{0,6}+(S_{3A}+S_{3B}+S_{3C})^{0,6}+(S_{4A}+S_{4B}+S_{4C})^{0,6}$,

ahol $S_{1A}=x_1f_{1A}+x_2f_{2A}+x_4((1-x_1)f_{1A}+(1-x_2)f_{2A})$

$S_{1B}=x_1f_{1B}+x_2f_{2B}+x_4((1-x_1)f_{1B}+(1-x_2)f_{2B}+x_3S_{1B})$

$S_{1C}=x_1f_{1C}+x_2f_{2C}$

$S_{2A}=(1-x_1)f_{1A}+(1-x_2)f_{2A}$

$S_{2B}=(1-x_1)f_{1B}+(1-x_2)f_{2B}+x_3S_{1B}$

$S_{2C}=(1-x_1)f_{1C}+(1-x_2)f_{2C}+x_3S_{1C}$

$S_{3A}=(1-x_4)S_{2A}$

$S_{3B}=(1-x_4)S_{2B}$

$S_{3C}=0$

$S_{4A}=0$

$S_{4B}=(1-x_3)S_{1B}$

$S_{4C}=(1-x_3)S_{1C}$

valamint

f_{iM}	M=A	M=B	M=C
i=1	100	1	1
i=2	100	1	200

a következő korlátozási feltételek mellett:

$S_{1A}+S_{3A}=200$

$S_{3B}+S_{4B}=2$

$S_{2C}+S_{4C}=201$

$x_i \in [0, 1]$ $i=1,\ldots,4$.

SEPNET2 [7]

$\min C(x)=(S_{1A}+S_{1B}+S_{1C})^{0,6}+(S_{2A}+S_{2B}+S_{2C})^{0,6}+(S_{3A}+S_{3B}+S_{3C})^{0,6}+(S_{4A}+S_{4B}+S_{4C})^{0,6}$,

ahol $S_{1A}=x_1f_{1A}+x_2f_{2A}+x_5((1-x_1)f_{1A}+(1-x_2)f_{2A})$

$S_{1B}=x_1f_{1B}+x_2f_{2B}+x_5((1-x_1)f_{1B}+(1-x_2)f_{2B}+x_4S_{1B})$

$S_{1C}=x_1f_{1C}+x_2f_{2C}$

$S_{2A}=(1-x_1)f_{1A}+(1-x_2)f_{2A}$

$S_{2B}=(1-x_1)f_{1B}+(1-x_2)f_{2B}+x_3S_{1B}$

$S_{2C}=(1-x_1)f_{1C}+(1-x_2)f_{2C}+x_3S_{1C}$

$S_{3A}=x_6 S_{2A}$
$S_{3B}=x_6 S_{2B}$
$S_{3C}=0$
$S_{4A}=0$
$S_{4B}=(1-x_4)S_{1B}$
$S_{4C}=(1-x_4)S_{1C}$

valamint

f_{iM}	M=A	M=B	M=C
i=1	80	9	12
i=2	3	3	90

a következő korlátozási feltételek mellett:

$x_3 S_{1A}+S_{3A}=81$
$(1-x_3)S_{1A}+(1-x_5-x_6)S_{2A}=2$
$(1-x_5-x_6)S_{2B}+S_{3B}+S_{4B}=12$
$S_{2C}+S_{4C}=102$
$x_5+x_6\leq 1$
$x_i\in[0,\ 1]\quad i=1,\ldots,6.$

Shekel5 (S5) [61]

$$f(x)=-\sum_{i=1}^{t}\frac{1}{(x-a_i)^{\mathrm{T}}(x-a_i)+c_i},\ t=5,\ x\in\mathbf{R}^4,\ \text{ahol}$$

i	a_i	c_i
1	$(4, 4, 4, 4)^{\mathrm{T}}$	0,1
2	$(1, 1, 1, 1)^{\mathrm{T}}$	0,2
3	$(8, 8, 8, 8)^{\mathrm{T}}$	0,2
4	$(6, 6, 6, 6)^{\mathrm{T}}$	0,4
5	$(3, 7, 3, 7)^{\mathrm{T}}$	0,4

Kiindulási intervallum (X): $[0,\ 10]^4$.

Three Hump Camel Back (THCB) [28]

$f(x)=12x_1^2-6{,}3x_1^4+6x_2(x_2-x_1),\ x\in\mathbf{R}^2.$
Kiindulási intervallum (X): $[-1,\ 1]\times[-1,\ 1{,}5]$.