

TDK dolgozat

Stágel Bálint Marcell

A DUÁL SZIMPLEX ALGORITMUS DEGENERÁCIÓ ELLENES STRATÉGIÁINAK ÖSSZEHASONLÍTÁSA

Szerző: Stágel Bálint Marcell

Szak / évfolyam: Mérnökinformatikus BSc. III. évf.

Témavezető: Tar Péter, PhD hallgató



**Pannon Egyetem, Műszaki Informatikai Kar
Rendszer és Számítástudományi Tanszék
Operációkutatási Kutatólaboratórium**

Absztrakt

A szimplex algoritmus fontossága ma már nem megkérdőjelezhető, a kereskedelmi és nyílt forráskódú LP megoldók egyaránt magas szintű implementációval rendelkeznek. Ez az algoritmus szomszédos bázisokon halad az optimum irányába, az ehhez szükséges pivotelem kiválasztása nagyban befolyásolja mind a megoldáshoz szükséges iteráció számot, mind az algoritmus monotonitását, így a pivotáláskor körültekintőnek kell lennünk. Munkánk során a duál szimplex algoritmus hatékonyságával foglalkoztunk, amelyet mind elméleti szinten, mind pedig az implementációt tekintve megvizsgáltunk. Mivel a degeneráció jelensége jelentős mértékben hátráltathatja a megoldást, így ennek a kezelése kritikus az algoritmus szempontjából.

Meglehetősen sokféle technika alkalmazható a numerikusan stabil pivotelem biztosítására, ezek hatékonysága azonban eltérő lehet. Ezen technikák megfelelő kombinálása sem triviális kérdés, hatásaik hol kiegészítik, hol kioltják egymást. Kis változtatások ezekben az algoritmusokban teljesen más bázisokhoz, és ezen keresztül más numerikus kihívásokhoz vezethetnek, így ezeket a technikákat szakszerűen kell tudnunk alkalmazni. Dolgozatunkban ezen módszerek összehasonlítását végeztük el, majd a hatékony implementációra vonatkozóan ajánlást fogalmaztunk meg. Közismert feladatok segítségével vizsgáltuk meg állításainkat, valamint különböző paraméterezésű futtatások összehasonlítására alapozva megfogalmaztuk következtetéseinket.

Kulcsszavak: Lineáris programozás, duál szimplex algoritmus, degeneráció

Tartalomjegyzék

1. Bevezetés	1
2. Irodalmi áttekintés	2
2.1. A lineáris programozási feladat	2
2.2. A duál szimplex algoritmus	5
2.2.1. Első fázis	6
2.2.2. Második fázis	7
2.3. A hányadoseszték szerepe és működése	8
2.3.1. Első fázis	8
2.3.2. Második fázis	11
2.4. A degeneráció	13
2.4.1. Lehetséges okai	14
2.4.2. Harris hányadoseszt	14
2.4.3. EXPAND eljárás	16
2.4.4. Célfüggvény együtthatók perturbálása	17
3. Hatékony megvalósítás	19
3.1. Egy nagyméretű LP megoldó implementációs kihívásai	19
3.2. Megvalósított technikák és algoritmusok	20
3.2.1. A PanOpt által használt matematikai modell	21
3.2.1. Toleranciaértékek használata	22
3.2.2. Stabil pivotálási eljárás	24
3.2.3. Harris hányadoseszt, EXPAND	25
3.2.4. Célfüggvény együtthatók perturbálása	28
3.3. Az implementáció struktúrája	29
3.3.1. A FeasibilityChecker osztály	30
3.3.2. A BreakpointHandler osztály	30
3.3.3. A DualRatiotest osztály	31
4. Mérési eredmények	33
4.1. Tesztfeladatok	33
4.2. Elvégzett mérések	34
4.2.1. Hamis megengedett hányadosok kezelése	34
4.2.2. Numerikusan stabil pivotelem biztosítása	35
4.2.3. Degenerált feladatok kezelése	38
5. Összefoglalás	43
Irodalomjegyzék	44
Köszönetnyilvánítás	45

1. Bevezetés

Manapság számos eszköz létezik a különböző gazdasági, politikai vagy egyéb döntések támogatására, a XX. században egész tudomány fejlődött ki e kérdéskör mélyebb vizsgálatára: az operációkutatás (Operations Research, OR). Az egyes döntések kimenetelét valamilyen szempont szerint (célfüggvény) értékelhetjük, majd a döntési alternatívák közül kiválaszthatjuk a számunkra legjobbat. Az alkalmazott matematikának ez a tudományága rendkívül szerteágazó. Magába foglal különböző ütemezési feladatokat, kombinatorikus módszereket, modellezési technikákat, hálózati, és logisztikai problémákat, egészértékű optimalizálást, útvonaltervezést, szimulációt és még sok minden mást is. Ezen területek közül munkánk során a lineáris programozással (Linear Programming, LP) foglalkoztunk.

Lineáris programozási problémáról akkor beszélünk, ha az optimalizálási feladatot leíró matematikai modellben szereplő összefüggések mind lineárisak egy folytonos értelmezési tartományon. Ezen terület még számos egyéb eset megoldásának alapját képezi, például a gyakorlati életben felmerülő egészértékű optimalizálási feladatok megoldása során akár több százezer relaxált LP feladatot kell megoldani az optimális megoldás megtalálásához. Jellemző, hogy a valós életben felmerülő feladatok rendkívül nagyméretűek, az ezek megoldására kidolgozott szoftverek közel sem triviális implementációt követelnek meg. Ezen megoldók alapja sok esetben a báziscserén alapuló algoritmusok közül a szimplex módszer. Az algoritmus két változata a primál és duál szimplex. Egy mai, korszerű LP megoldótól mindkét implementáció magas színvonalú jelenléte elvárható. Ezen algoritmusokat nagyban hátráltató jelenség a degeneráció, ilyenkor több iteráción keresztül nem változik a célfüggvény-érték.

Munkánk során a duál szimplex algoritmussal foglalkoztunk. Megvizsgáltuk a báziscsere pivotelemének (generálóelemének) meghatározására szolgáló hányadoseszt eljárásokat, melyek a degeneráció kezelésére nyújtanak lehetőséget. Feltártuk továbbá a különböző degenerált eseteket, illetve az ezek kezelésére szolgáló fellelhető technikákat. Ezen vizsgálódásainkat a Pannon Egyetem Operációkutatási Kutatólaboratóriuma által fejlesztett Pannon Optimizer (PanOpt) LP megoldó fejlesztésének keretében végeztük. A szakirodalomban található legfontosabb degeneráció-ellenes stratégiákat megismertük, megvalósítottuk, illetve azok továbbfejlesztésének érdekében hatékonysági vizsgálatok alá vetettük. Ezek alapján kiegészítettük a megvalósított technikákat annak érdekében, hogy a duál algoritmus hatékonyságát, stabilitását a lehető legnagyobb mértékben növeljük.

Dolgozatunkban ismertetjük a duál szimplex algoritmus elméleti hátterét, kiemelt tekintettel a hányadosesztékre és a degenerációra. Kitérünk a megvalósított technikák implementációjára és tesztelésére a PanOpt keretében, végül sort kerítünk az általunk elvégzett mérési eredmények bemutatására és a levont következtetések ismertetésére.

2. Irodalmi áttekintés

Ebben a fejezetben ismertetjük a lineáris programozási feladat standard alakját, jellemzőit, kitérünk a feladatot különböző módon reprezentáló formalizmusokra. Ismertetjük továbbá az LP-hez kapcsolódó alapvető fogalmakat és jelöléseket, amelyek feltétlenül szükségesek ahhoz, hogy érdemben tudjuk tárgyalni a hányadoseszteket.

A második alfejezetben nagy vonalakban bemutatjuk a duál szimplex algoritmust, annak felépítését, sajátosságait. Tárgyaljuk a hányadosesztek melletti főbb építőelemeket, a megoldó két fázisának lényegét és a megoldás menetét.

Az utolsó alfejezetben rátérünk a duál hányadosesztek elméleti hátterére, hiszen ez képezte vizsgálódásunk alapját. Részletesen bemutatjuk a különböző hányados-meghatározási módszerek működését, feltárjuk a bennük rejlő lehetőségeket.

Tekintsük át röviden a dolgozatban alkalmazott jelöléskonvenciót:

- Dőlt kisbetű: v , konstans, változóérték, index, vagy sorszámnév
- Félkövér kisbetű: $\boldsymbol{v}$, vektor,
- Félkövér nagybetű: $\boldsymbol{M}$, mátrix,
- Írott nagybetű: $\mathcal{M}$, halmaz.

2.1. A lineáris programozási feladat

A lineáris programozás széles körben alkalmazható, rengeteg különböző optimalizálási feladat megoldásának alapja lehet. Mint már említettük egy lineáris programozási feladat többféle módon formalizálható, amelyek minden esetben ekvivalensek. Ezek jellemzője, hogy a változókat és feltételeket különböző kategóriákba sorolják, ezáltal azok a megoldás során hatékonyan kezelhetők.

Az LP feladat standard alakja:

$$\begin{aligned} \min \quad & \boldsymbol{c}^T \boldsymbol{x} \\ \text{s. t.} \quad & \boldsymbol{A} \boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \mathbf{0}, \end{aligned} \tag{1}$$

ahol $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ jelöli a feltételekben szereplő együtthatókból származó mátrixot, $\boldsymbol{x} \in \mathbb{R}^n$ a döntési változókat tartalmazó vektort, $\boldsymbol{c} \in \mathbb{R}^n$ a költségegyütthatók vektorát, $\boldsymbol{b} \in \mathbb{R}^m$ pedig a konstansokat tartalmazó vektort. Az egyes változókra vonatkozóan meghatározhatók egyéni korlátok, amelyek általános esetben a következőképpen formalizálhatók:

$$l_j \leq x_j \leq u_j, j = 1 \dots n, \tag{2}$$

ahol l_j a j . változó alsó, u_j pedig a felső korlátját jelenti.

A standard alak mellett, gyakran használt az LP feladat egyes számítási alakja (Computational Form 1, CF-1) [1]:

$$\begin{aligned} \min \quad & c^T x \\ \text{s. t.} \quad & Ax = b \\ & \text{type}(x_j) \in \{0, 1, 2, 3\}; j = 1, 2, \dots, n \end{aligned} \quad (3)$$

A modellben használt $\text{type}(x_j)$ formalizmus a j . változó típusát jelenti, amely a következők valamelyike lehet:

- type 0: fix vagy kötött változók, $l_j = x_j = u_j = 0$,
- type 1: korlátos változók, $0 \leq x_j \leq u_j < \infty$,
- type 2: plusz típusú, vagy alsó korlátos változók, $0 \leq x_j < \infty$,
- type 3: szabad változók, $-\infty < x_j < +\infty$.

Ha az A mátrix teljes rangú, annak m darab lineárisan független oszlopa egy nem szinguláris mátrixot, vagyis bázist alkot, ezt jelölje $B \in \mathbb{R}^{m \times m}$. Az együtthatómátrix többi oszlopából képzett nem-bázis részhez tartozó mátrixot pedig jelölje $R \in \mathbb{R}^{m \times (n-m)}$, feltételezhetjük, hogy az ehhez szükséges $n \geq m$ feltétel teljesül. Az x vektort ennek megfelelően két részre bonthatjuk, így a lineárisan független oszlopok által meghatározott változókat bázisváltozóknak, míg a többi nem-bázis változóknak nevezzük. A változók indexeinek tárolására vezessünk be két halmazt, legyenek a bázisváltozók indexei a $\mathcal{B}$, a nem-bázis-változóké pedig az $\mathcal{R}$ halmazban. Ezek alapján egy permutációt alkalmazva átrendezhetjük az együttható mátrixot, valamint a c és x vektorokat az $A = [B \mid R]$, $x = \begin{bmatrix} x_{\mathcal{B}} \\ x_{\mathcal{R}} \end{bmatrix}$, valamint $c = \begin{bmatrix} c_{\mathcal{B}} \\ c_{\mathcal{R}} \end{bmatrix}$ alakokra. Egy bázisról azt mondjuk, hogy degenerált, ha valamely bázisváltozó a korlátján van. A későbbiekben részletesen tárgyaljuk a degeneráció kérdéseit, hiszen ez képezi vizsgálódásunk alapját.

A CF-1-es alakra jellemző, hogy a változók véges alsó korlátja minden esetben nullába van eltolva, ez minden különösebb nehézség nélkül megtehető. Ez az alak a megoldó algoritmus számára hatékonyan használható, a korlátos változók külön kezelése sok lehetőséget rejt magában (pl. korlátcserés lépés). Éppen ezért előnyös megvizsgálni, hogy milyen LP reprezentációt akarunk használni a megoldás során, hiszen ezek más és más szempontból előnyösek. Például egy bonyolultabb matematikai modell sok esetben jóval megkönnyíti a megoldás menetét, ez által hatékonyabb, mint egy egyszerűbb modell, amit viszont könnyebb felírni.

Egy x vektort megoldásnak tekintünk, ha kielégíti az $Ax = b$ feltétel rendszert, megengedett megoldásról akkor beszélünk, ha mindeközben a változók kielégítik egyéni korlátjaikat. Optimális megoldás pedig az a megengedett megoldás, amelyhez tartozó célfüggvény-érték a lehető legjobb.

A már korábban ismertetett bázis és nem-bázis felbontás alapján a feladat feltételrendszere az alábbi alakot ölti:

$$\mathbf{B}\mathbf{x}_{\mathcal{B}} + \mathbf{R}\mathbf{x}_{\mathcal{R}} = \mathbf{b} \quad (4)$$

Mivel $\mathbf{B}$ bázis, ezért nem szinguláris mátrix, tehát létezik annak inverze, amely alapján a bázisváltozók:

$$\mathbf{x}_{\mathcal{B}} = \mathbf{B}^{-1}(\mathbf{b} - \mathbf{R}\mathbf{x}_{\mathcal{R}}) \quad (5)$$

Bontsuk fel a célfüggvényt is ennek megfelelően:

$$\begin{aligned} z = \mathbf{c}_{\mathcal{B}}^T \mathbf{x}_{\mathcal{B}} + \mathbf{c}_{\mathcal{R}}^T \mathbf{x}_{\mathcal{R}} &= \mathbf{c}_{\mathcal{B}}^T \mathbf{B}^{-1}(\mathbf{b} - \mathbf{R}\mathbf{x}_{\mathcal{R}}) + \mathbf{c}_{\mathcal{R}}^T \mathbf{x}_{\mathcal{R}} = \\ &= \mathbf{c}_{\mathcal{B}}^T \mathbf{B}^{-1} \mathbf{b} + (\mathbf{c}_{\mathcal{R}}^T - \mathbf{c}_{\mathcal{B}}^T \mathbf{B}^{-1} \mathbf{R}) \mathbf{x}_{\mathcal{R}} \end{aligned} \quad (6)$$

Az utolsó alakban, a zárójelben szereplő kifejezést nevezzük redukált költségnek, melynek j . komponense: $d_j = c_j - \mathbf{c}_{\mathcal{B}}^T \mathbf{B}^{-1} \mathbf{a}_j$, ami a változó egységnyi megváltozásának a célfüggvényre gyakorolt hatását írja le. [2] Ha tehát egy nem-bázis változó értékét megváltoztatjuk, akkor az elmozdulás a célfüggvényt d_j -nek megfelelően befolyásolja. Ha nincs olyan változó, amely elmozdulása javítana a célfüggvényen, akkor optimális megoldást találtunk, az erre vonatkozó feltételek CF-1-es alakot feltételezve:

- alsó korlátos nem-bázis változók esetében $d_j \geq 0$,
- felső korlátos lévő nem-bázis változókra $d_j \leq 0$,
- egy szabad változóé $d_j = 0$,
- fix változók redukált költsége bármilyen előjelű lehet, báziscsere során nem léphet be a bázisba.

Megjegyzésként hozzáfűznénk, hogy egy maximum feladatra pont ellentétes előjelű feltételek vonatkoznak. A könnyebb áttekinthetőség érdekében tekintsük át táblázatos formában (**1. táblázat**) a CF-1-es alakra ismertetett optimalitási feltételeket:

1. táblázat: Változókra vonatkozó optimalitási feltételek a CF-1-es alakban

$type(x_j)$	a változó értéke	optimalitási feltétel
0	$x_j = 0$	d_j tetszőleges
1	$x_j = 0$	$d_j \geq 0$
1	$x_j = u_j$	$d_j \leq 0$
2	$x_j = 0$	$d_j \geq 0$
3	$x_j = 0$	$d_j = 0$

Az eddig tárgyalt modell a lineáris programozási feladat primál változata, emellett beszélhetünk a standard alak duál párjáról, vagyis duál feladról:

$$\begin{aligned} \max \mathbf{b}^T \mathbf{y} \\ \text{s. t. } \mathbf{A}^T \mathbf{y} + \mathbf{w} = \mathbf{c} \\ \mathbf{w} \geq \mathbf{0}, \end{aligned} \quad (7)$$

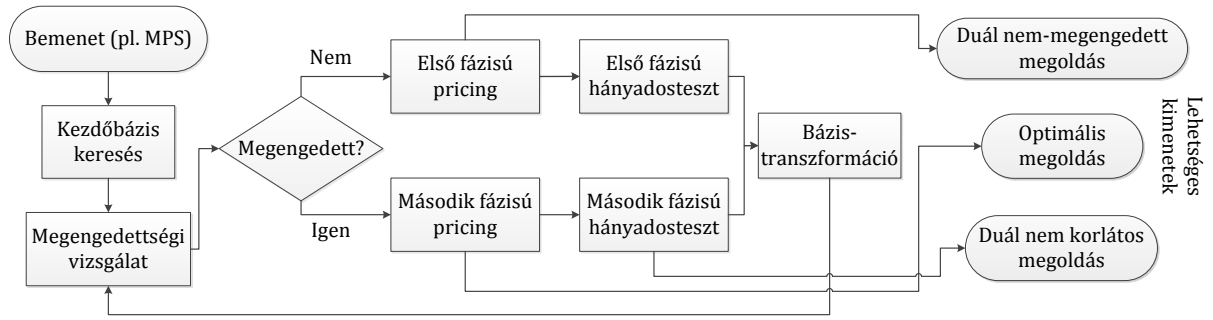
ahol a már bevezetett jelölést használjuk, a két új elem a modellben a primálhoz képest az $\mathbf{y}$, ez a duál változók vektora, és a $\mathbf{w}$, a duál logikai változók vektora. Minden primál feladathoz tartozik egy duál feladat, ezeket gyakran párban tárgyalják, hiszen sok analógia lelhető fel a két alak között. Érdekesség, hogy ha a primál-duál feladatpár bármelyikének van egy megengedett megoldása véges optimummal, akkor ugyanez igaz a másik feladatra, és az optimális célfüggvény-értékek megegyeznek. Ezt a kapcsolat erős dualitásnak nevezzük, ami könnyen belátható, ha a duál változókat $\mathbf{y} = \mathbf{c}_{\mathcal{B}}^T \mathbf{B}^{-1}$ -ként választjuk meg. A továbbiakban a duál változók értékének az így kiszámítható értékeket tekintjük. [3]

Egy másik érdekessége a primál-duál feladatpárnak, hogy a duál megengedettségi feltételek megegyeznek a primál optimalitási feltételekkel, valamint a duál optimalitási feltételek megegyeznek a primál megengedettségi feltételekkel. Ennek megfelelően a primál szimplex első és második fázisának célja rendre megegyezik a duál algoritmus második és első fázisának céljával. A témakörhöz kapcsolódó további vizsgálatokról, illetve az ismertetett analógiák bizonyításáról dolgozatunkban eltekintünk.

2.2. A duál szimplex algoritmus

Miután modelleztük a lineáris programozási problémát, választhatunk, hogy milyen megoldó algoritmust használunk. Az egyik kiemelkedő a szimplex algoritmus, melynek két változatát a primált és a duált Dantzig és Lemke publikálták. [4] [5] Mindkét változatra jellemző, hogy csak egy változóban eltérő, vagyis szomszédos bázisokon halad az n dimenziós térben az optimum irányába. Magához a haladáshoz nincs szükség másra, mint a megfelelő feltételek melletti báziscsere során ki és belépő változóra. Egy adott LP feladatot a primál és duál szimplex algoritmusok bármelyikével megoldhatunk, a már ismertetett párhuzamok alapján. Szeretnénk hangsúlyozni, hogy egyik algoritmus sem domináns a másik felett, mindkettőnek megvannak a maga előnyei, ezért egy korszerű megoldóban mindkettő kiemelt helyet, elméleti szempontból pedig kiemelt figyelmet érdemel.

A duál szimplex algoritmus fő elemeit tekintve az alábbiaképpen működik (1. ábra):



1. ábra: A duál szimplex algoritmus működése

A már ismertetett fogalmak alapján beszélhetünk az algoritmus két külön fázisáról, a következőkben tekintsük át ezeket.

2.2.1. Első fázis

A duál szimplex algoritmus első fázisának célja a változókra ismertetett optimalitási feltételek kielégítése. Míg primál esetben változók értékével és korlátaival számolunk, addig a duál esetben a redukált költségek értékével és korlátaival. A d_j redukált költség megengedettségi tartományának a j . változóra vonatkozó optimalitási feltétel által meghatározott intervallumot tekintjük, amint azt az előző alfejezetben ismertettük (1. táblázat).

Ezen feltételek kielégítése tehát a célunk a duál első fázis során, ehhez definiáljuk a következő megengedettségi halmazokat:

- azon nem-bázis változók indexei, amelyek redukált költsége negatív irányban tér el megengedettségi tartományuktól kerüljenek a $\mathcal{M}$ halmazba,
- a duál megengedettségi tartományuktól pozitív irányban eltérő nem-bázis változók indexei kerüljenek a $\mathcal{P}$ halmazba,
- végül a duál megengedett nem-bázis változók indexei kerüljenek az $\mathcal{F}$ halmazba.

A halmazok alapján definiálhatjuk a nem-megengedettség mértékét, vagyis a duál első fázisú célfüggvényt [2]:

$$\max f = \sum_{j \in \mathcal{M}} d_j - \sum_{j \in \mathcal{P}} d_j \quad (8)$$

Mivel az $\mathcal{M}$ halmazban lévő változók redukált költsége negatív, a $\mathcal{P}$ halmazhoz tartozóké pozitív, ezért $f \leq 0$ mindig teljesül, $f = 0$ esetben pedig duál megengedett bázismegoldásunk van. Az algoritmus szomszédos bázisokon halad mindeközben, így meg kell határozunk a redukált költségek vektorának az új bázisra vonatkozó képét.

Ha a generálóelemet a p . sorból választottuk, és áttérünk a szomszédos bázisra, akkor a redukált költségekből ki kell vonnunk a p . sor valahányszorosát (ez a hányadosesztben kerül

meghatározásra), jelöljük ezt a szorzót t -vel. Vizsgáljuk meg a duál elsőfázisú célfüggvényt a t megváltozás függvényében:

$$f(t) = \sum_{j \in \mathcal{M}} d_j(t) - \sum_{j \in \mathcal{P}} d_j(t) = f(0) - t \left(\sum_{j \in \mathcal{M}} \alpha_j^p - \sum_{j \in \mathcal{P}} \alpha_j^p \right), \quad (9)$$

ahol α_j^p az $\mathbf{A}$ mátrix p . sorának j . elemének az adott bázisra vonatkozó képe, a zárójelben szereplő kifejezés pedig a változó duál első fázisú redukált költsége, amelyet így jelölünk:

$$v_p = \sum_{j \in \mathcal{M}} \alpha_j^p - \sum_{j \in \mathcal{P}} \alpha_j^p \quad (10)$$

Ezen jelölés bevezetésével leegyszerűsödött a célfüggvény értékének megváltozása, ami így a $\Delta f = -tv_p$ alakban írható le. Ha a redukált költségek megengedettek lesznek, akkor az $\mathcal{M}$ és $\mathcal{P}$ halmazok kiürülnek, az első fázisú célfüggvény értéke 0 lesz. Mivel ezt a célfüggvényt maximalizáljuk, ezért, hogy a megváltozás pozitív legyen, a következő feltételek teljesülése szükséges:

- ha $v_p > 0$, akkor $t < 0$,
- ha $v_p < 0$, akkor $t > 0$.

Tehát válasszunk ki egy nem-nulla első fázisú redukált költségű javító változót, ezt az eljárást nevezzük duál első fázisú pricing-nak. A kijelölt javító bázisváltozóhoz az első fázisú hányadosesztben meghatározunk egy bejövő nem-bázis változót, ennek részleteit a 2.3 fejezetben tárgyaljuk.

Ha a célfüggvény értéke 0, akkor duál megengedett megoldást találtunk és folytathatjuk a második fázissal, ellenkező esetben, ha már nincs javító változó, és nem nulla ez az érték, akkor a feladat duál nem-megengedettséget mondhatjuk ki.

2.2.2. Második fázis

Ha a duál megengedettséget biztosítottuk az első fázis során, akkor kezdetét veszi a második fázis, amely ezt a megengedettséget megtartva keres optimális megoldást. Ehhez a célfüggvényünk a már ismertetett $Z = \mathbf{b}^T \mathbf{y}$.

Legyen adott egy duál megengedett $\mathbf{B}$ bázis. Válasszuk ki a k . bázisváltozóhoz tartozó egyenlőségként teljesülő duál feltételt, majd relaxáljuk azt úgy, hogy a célfüggvényérték javuljon, ennek értelmében az $\mathbf{a}_k^T \mathbf{y} = c_k$ feltétel alakja $\mathbf{a}_k^T \mathbf{y} \geq c_k$ vagy $\mathbf{a}_k^T \mathbf{y} \leq c_k$ lesz. Attól függően, hogy az egyenlet bal oldalán szereplő kifejezés értéke nőtt vagy csökkent, beszélhetünk $t \geq 0$ és $t \leq 0$ esetről, ahol t a feltétel eltolásának mértéke. Mindezt a duál változók értékeinek az alábbi módon történő megváltozása kompenzálja ($\mathbf{e}_p$ kanonikus bázisvektor) [2]:

$$\mathbf{B}^T \mathbf{y} = \mathbf{B}^T \mathbf{y}(t) - t \mathbf{e}_p, \quad (11)$$

amit átrendezve megkapjuk a duál változók megváltozását:

$$\mathbf{y}(t) = \mathbf{y} + t\boldsymbol{\rho}_p, \quad (12)$$

ahol $\boldsymbol{\rho}_p$ a $\mathbf{B}^{T-1}$ mátrix p . sora. Ez alapján megvizsgálhatjuk a duál célfüggvény-érték megváltozását t függvényében:

$$Z(t) = \mathbf{b}^T \mathbf{y}(t) = \mathbf{b}^T \mathbf{y} + t\mathbf{b}^T \boldsymbol{\rho}_p = \mathbf{b}^T \mathbf{y} + t\beta_p = Z(0) + t\beta_p, \quad (13)$$

ahol β_p a p . bázisváltozó értékét jelöli. Mivel a célfüggvényt maximalizáljuk, ezért ez a megváltozás pozitív kell, hogy legyen. Ez az alábbi esetekben teljesül:

- ha $t < 0$, akkor $\beta_p < 0$,
- ha $t > 0$, akkor pedig $(\beta_p - u_{\mathcal{B}_p}) > 0$, ahol $u_{\mathcal{B}_p}$ a p . bázisváltozó felső korlátja.

Összegezve tehát, egy olyan sorra van szükségünk, ami nem teljesíti a duál optimalitási feltételt, ezt a javító változót a duál második fázisú pricing határozza meg.

2.3. A hányadoseszték szerepe és működése

Az eddigiekben áttekintettük, hogy a duál szimplex algoritmus során hogyan kapunk javító változót, ehhez milyen feltételeknek kell teljesülnie. A hányadosesztéknek az elsődleges be-menete tehát nem más, mint a bázis-transzformáció során a bázist elhagyó változó. Ehhez a hányadoseszték során meghatározunk egy bejövő változót. Ezt követően elvégezhetők a szükséges transzformációk, a báziscsere, és így haladhatunk iteratívan az optimum felé. [6] A báziscseréhez szükség van egy generálóelemre, amit a pricing és a hányadoseszték együttesen határoz meg.

2.3.1. Első fázis

Ha a duál megengedettség a célunk, tehát első fázisban vagyunk, akkor a következők szerint definiáljuk a hányadosokat. Ahogy azt már bevezettük, a redukált költségek j . komponense az alábbi alakot ölti:

$$d_j = c_j - \mathbf{c}_{\mathcal{B}}^T \mathbf{B}^{-1} \mathbf{a}_j \quad (14)$$

Vizsgáljuk meg ezen értékek megváltozását a t elmozdulás függvényében, felhasználva a duál változók definícióját, illetve azok megváltozását:

$$\begin{aligned} d_j(t) &= c_j - \mathbf{a}_j^T \mathbf{y}(t) = c_j - \mathbf{a}_j^T (\mathbf{y} + t\boldsymbol{\rho}_p) = c_j - \mathbf{c}_{\mathcal{B}}^T \mathbf{B}^{-1} \mathbf{a}_j - t\mathbf{e}_p^T \mathbf{B}^{-1} \mathbf{a}_j \\ &= d_j - t\alpha_j^p. \end{aligned} \quad (15)$$

Ennek alapján, ahol a j . változó duál megengedettségi állapotot vált, egy $t_j = d_j/\alpha_j^p$ hányadost definiál, hiszen annak átlépése esetén másik megengedettségi halmazba kerül. Így

tehát, egy adott változó jelenlegi megengedettségi állapota szerint az alábbi esetekben definiál hányadost (2. táblázat):

2. táblázat: Hányadost definiáló irányok a duál első fázisban

Megengedettségi tartomány	$\mathcal{M}$		$\mathcal{F}$		$\mathcal{P}$	
	$\alpha_j^p < 0$	$\alpha_j^p > 0$	$\alpha_j^p < 0$	$\alpha_j^p > 0$	$\alpha_j^p < 0$	$\alpha_j^p > 0$
$t_j < 0$	-	d_j/α_j^p	d_j/α_j^p	d_j/α_j^p	d_j/α_j^p	-
$t_j > 0$	d_j/α_j^p	-	d_j/α_j^p	d_j/α_j^p	-	d_j/α_j^p

Láthatjuk tehát, hogy egy $\mathcal{M}$ halmazbeli változóhoz akkor definiálunk hányadost, ha redukált költsége pozitív irányba változik, míg $\mathcal{P}$ halmaz esetén akkor, ha ezen megváltozás iránya negatív. Ehhez értelemszerűen szükség van arra, hogy a változónak abban az irányban, amerre halad, legyen korlátja. Most vizsgáljuk meg, hogy a már ismertetett CF-1-es alakban használt változó típusok, mikor definiálnak hányadost.

- Egy fix típusú változóra vonatkozó optimalitási feltétel nem korlátoz, tehát mindegy, hogy milyen a redukált költségének előjele, ha egyszer elhagyta a bázist, akkor többet nem hozzuk be, ezért ezeket a változókat figyelmen kívül hagyhatjuk a hányadostesztben.
- Egy korlátos változóra, attól függően, hogy melyik korlátján van, különböző optimalitási feltétel vonatkozik, de mivel a korlátcserés lépés (boundflip) során nem változik a változó redukált költsége (hiszen ilyenkor nincs transzformáció), ezért ezen változók kezelése egyszerűsíthető. Ha redukált költsége nem megengedett értékű, akkor tegyük a másik korlátjára, és ugyanaz a redukált költség a másik korláton már megengedett lesz. Ezt az eljárást nevezzük megengedettségi korrekciónak (feasibility correction), amelyet a hányadosteszt után kell elvégezni, hiszen ekkor már nem transzformálódnak a redukált költségek. [2]
- Plusz típusú változóra az optimalitási feltétel $d_j \geq 0$. Ennek megfelelően egy ilyen változó az $\mathcal{M}$ és $\mathcal{F}$ halmazok valamelyikében lehet, így ha az egyik halmazból a másik felé mozog, akkor a két halmaz határán definiálhat töréspontot.
- Egy szabad változó duál megengedettségi tartománya $d_j = 0$ -ban van, ennek megfelelően bármelyik megengedettségi halmazban szerepelhet. Ha az adott halmazból kilép, akkor minden esetben hányadost definiál. Ha egy ilyen változó a $\mathcal{P}$ halmazból $\mathcal{M}$ irányába mozog, akkor a 0 helyen 2 töréspontot is definiál, hiszen megengedettségi tartományának ez az egyetlen pontja. Hasonlóképpen két hányadost definiál, ha $\mathcal{M}$ -ből a $\mathcal{P}$ halmaz irányába mozog.

A hányadosok meghatározása után választhatunk, hogy mely módszer alapján szeretnénk meghatározni a bejövő változót.

Hagyományos egylépéses módszer

Egy fontos fenntartásunk van, ha ezzel a módszerrel számoljuk a hányadosokat, ez pedig az, hogy a kimenő változó redukált költsége mindig legyen megengedett, ezáltal a nem-megengedett változók száma sosem növekszik. Válasszuk ki a definiált hányadosok közül abszolút érték szerint a legkisebbet és az ezt definiáló változóval végezzük el a báziscserét. Előjelüket tekintve ezek a hányadosok bármilyenek lehetnek, ezért van szükség az abszolút érték szerinti sorba rendezésre. Megjegyzésként hozzáfűznénk, hogy ez a módszer, amennyiben van 0-ban definiált töréspont, minden esetben azzal fog visszatérni, ez által gyakoriak a degenerált iterációk, viszont folyamatos, monoton haladást eredményez.

Törtvonalfüggvényes hányados-meghatározás

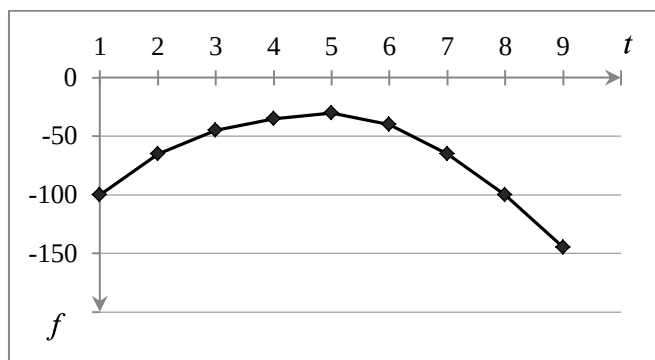
Ez a módszer jóval összetettebb, az adott iterációban elérhető legnagyobb előrehaladást garantálja. Vizsgáljuk meg az első fázisú célfüggvényt az elmozdulás függvényében.

Az első fázisú redukált költségek definiálásából következően a célfüggvény meredeksége kezdetben $s_0 = |v_p|$. [2] Az első hányadosig a függvény lineárisan növekszik, de érdemes megvizsgálni, hogy mi történik ebben a pontban. Tudjuk azt, hogy minden változó, akkor definiál töréspontot, ha megengedettségi állapotot vált. Ennek megfelelően vagy nem-megengedett lesz a változó, és az indexe az $\mathcal{M}$ és $\mathcal{P}$ halmazok valamelyikébe kerül, vagy kikerül az indexe ezekből a halmazokból, mert megengedett állapotba kerül. Mindkét esetben a célfüggvény meredeksége megváltozik, méghozzá pontosan az első hányadoshoz tartozó alfa értékkel csökken. Tehát az második hányadosnál $s_1 = s_0 - |\alpha_j^p|$. Ha ez a meredekség pozitív, akkor érdemes tovább menni a következő hányadosig, mert ebben az esetben a nagyobb lépés még mindig javít a célfüggvényen. Így lehet, hogy az első hányadost definiáló változó már nem-megengedett tartományba kerül, de összességében, ha a meredekség pozitív, akkor a nem-megengedettség összmértéke csökken. A függvényen haladva további töréspontokat találunk, így egy szakaszonként lineáris konkáv függvényt kapunk. [7]

Határozzuk meg ezen törtvonalfüggvény maximumát. Összességében addig megyünk a hányadosokon, ameddig a célfüggvény meredeksége negatív nem lesz. Azon a helyen, ahol a meredekség előjelet vált, maximuma van a függvénynek, ez lesz az elérhető legnagyobb javulás. Rekurzív összefüggésekkel megadva minden töréspontnál a célfüggvény-érték $f_k = f_{k-1} + s_p^{k-1}(t_k - t_{k-1})$, a meredeksége pedig $s_p^k = s_p^{k-1} - |\alpha_{j_k}^p|$ szerint írható le. [2]

Érdemes kiemelni, hogy akár több törésponthoz tartozó célfüggvény érték is jobb lehet a kezdeti pontban lévőnél, ezért ezek közül bármelyiket kiválasztva és a báziscserével járó transzformációkat elvégezve javító iterációt kapunk. Így az előbbi módszerhez képes több

alternatív lehetőségünk van, nagyobb eséllyel elkerülhető a degeneráció. Erre a későbbiekben még visszatérünk. Grafikonon ábrázolva egy törtvonalfüggvényt (2. ábra), tipikusan az alábbihoz hasonló forma jellemző:



2. ábra: Első fázisú törtvonalfüggvény

Ha az első fázisú célfüggvényt 0 értékre hoztuk, akkor végezzük el a megengedettségi korrekciókat a korlátos változókra, ezzel duál megengedett megoldást kapunk és rátérhetünk a második fázisra.

2.3.2. Második fázis

Legyen adott egy duál megengedett B bázis. Amennyiben ebben a bázisban van olyan változó, amely sérti az egyéni korlátait, tehát nem teljesíti a duál optimalitási feltételeket, akkor mindig létezik egy szomszédos bázis, legalább ugyanakkora célfüggvény-értékkel, máskülönben a feladat duál nem-korlátos. [2]

Mint már említettük a második fázis során folyamatosan megköveteljük, hogy teljesüljön az első fázis során biztosított duál megengedettség. Ez formálisan a következőt jelenti:

Alsó korláton lévő nem-bázis változók esetén a redukált költségek t szerinti megváltozása után a megengedettségét a $d_j(t) = d_j - t\alpha_j^p \geq 0$ összefüggés írja le, a felső korláton lévő változókét pedig $d_j(t) = d_j - t\alpha_j^p \leq 0$. Ezek alapján a definiálhatjuk azt a legnagyobb lépést, ahol a változó redukált költsége még megengedett marad.

- Ha $t \leq 0$, vagyis a javító változó alsó korlátja alatti értéken van, akkor $\alpha_j^p < 0$ teljesülése mellett a lépésre adható alsó korlát: $t_j \geq d_j/\alpha_j^p$, minden alsó korláton lévő nem-bázis változóra. Felső korláton lévőkre pedig $\alpha_j^p > 0$ mellett szintén $t_j \geq d_j/\alpha_j^p$ adható. Ellenkező α_j^p előjelek esetén a redukált költségek megengedett tartományban maradnak, ilyenkor nem definiálnak hányadost.
- Ha felső korlát feletti értéken van a javító bázisváltozó, vagyis $t \geq 0$ esetben vagyunk, akkor alsó korlátos változókra $\alpha_j^p > 0$ esetén a lépésre adható felső korlát

$t_j \leq d_j/\alpha_j^p$, felső korláton lévőkre pedig $\alpha_j^p < 0$ teljesülése esetén $t_j \leq d_j/\alpha_j^p$. Az α_j^p előjeleket megfordítva a redukált költségek megengedett irányba haladnak, ebben az esetben nem definiálnak hányadost.

Összegezve tehát elmondhatjuk, hogy az a lépéshossz, amire a j . nem-bázis változó duál megengedettsége még nem sérül, mindkét esetben $t_j = d_j/\alpha_j^p$. [3]

Vizsgáljuk meg, hogy az egyes változótípusokra nézve ez mit jelent. Első fázishoz hasonlóan a fix változók nem játszanak szerepet, hiszen azokra vonatkozó optimalitási feltétel nem korlátoz. Plusz típusú és szabad változókra van duál megengedettségi megkötés, ezért ezeket bevonjuk a hányadostesztbe, hogy megengedettek maradjanak. A korlátos változók nem-megengedettségének kezelését a törtvonalfüggvényes módszer keretében ismertetjük.

Hagyományos egy lépéses módszer

A hányadosok definiálását követően ezeket sorba rendezzük az első fázishoz hasonló módon abszolút érték szerint. Ezt követően kiválasztjuk a legkisebbet, majd annak segítségével áttérünk egy szomszédos bázisra. A módszerről továbbra is állítjuk, hogy nem túl hatékony, gyakoriak a degenerált iterációk, viszont monoton előrehaladást biztosít.

Törtvonalfüggvényes módszer

A második fázisra definiált törtvonalfüggvény alapkoncepcióját tekintve hasonló, mint az első fázisú, néhány szempontból viszont jelentősen különbözik attól. Vizsgáljuk meg a korlátos változókat. Ha egy ilyen változót választunk ki a hányadostesztben, akkor az a bázisba $x_p + \theta_p$ értéken kerül be, ami nem megengedett, ha a lépéshossz nagyobb, mint a változó megengedettségi tartománya, vagyis $|\theta_p| > u_p$ esetén. Viszont, ha nem végezzük el a transzformációt, hanem ehelyett korlátcserét hajtunk végre, akkor a változó redukált költsége megengedett marad, a célfüggvény-érték javul. [2] Persze ettől a pricing által a bázis elhagyására kijelölt változó javító változó marad, és ezért a következő iterációban újra kiválasztásra kerülhet. Így egy javító sor felhasználásával többszörös korlátcserét (boundflip-et) hajthatunk végre, amíg találunk egy változót, amely megengedett értéken lép be a bázisba. Így hatékonyan egy iterációban nagymértékben tudjuk csökkenteni a bázisváltozók nem-megengedettségét, egy sor maximális javító tulajdonságát kihasználjuk.

Vizsgáljuk meg a második fázisú célfüggvényt. Ha a sorba rendezett hányadosok közül az első korlátos változó, akkor ezen változón korlátcserét hajtunk végre és haladunk tovább bejövő változót keresve. A célfüggvény kezdeti meredeksége $s_0 = -\beta_p$, ha a kimenő változó alsó korlátja alatti, és $s_0 = \beta_p - u_p$, amennyiben felső korlátja feletti értéken szerepel. [7] Feltéve tehát, hogy az első hányadosnál korlátcserét hajtunk végre, láthatjuk, hogy a célfüggvény meredeksége $s_1 = s_0 - |\alpha_j^p| u_j$ alapján változik. Így az első fázishoz hasonlóan, ezt addig is-

mételhetjük, ameddig ez a meredekség előjelet vált, hiszen azon a helyen maximuma lesz a célfüggvénynek, az adott lépésben az lesz az elérhető legnagyobb javulás. [8]

Iteratíván tehát mindaddig, ameddig a $|\theta_p| > u_p$ feltétel teljesül, haladunk tovább a törtvonalfüggvényen. Eközben a meredekség és a célfüggvény-érték rekurzívan változnak az $s_k = s_{k-1} - |\alpha_j^p| u_j$ és $f_k = f_{k-1} + s_{k-1}(t_k - t_{k-1})$ összefüggések alapján. [2] Így egy javító sorhoz a törtvonalfüggvényes hányadoseszt kijelölhet néhány boundflip-et, valamint egy bejövő változót. Könnyen látható, hogy vagy egy olyan korlátos változót fogunk behozni a bázisba, aminek megengedettségi tartománya nagyobb, mint a primál lépéshossz, vagy egy plusz típusú változót, hiszen azon túllépve a célfüggvény-érték minden határon túl csökkenne, mivel $u_j = \infty$. Amennyiben szabad változó is definiál töréspontot, akkor nincs is szükség a hányadosok sorba rendezésére, hiszen az minden esetben bekerül a bázisba.

Abban az esetben, ha az összes hányadost felhasználtuk a törtvonalfüggvény meghatározásakor, és annak meredeksége még mindig pozitív előjelű, akkor a feladat duál nemkorlátosságát mondhatjuk ki. [3] Kiemelnénk, hogy ebben a fázisban csak akkor jelent előnyt a törtvonalfüggvényes módszer a hagyományossal szemben, ha a feladatban szerepelnek korlátos változók, máskülönben működése a hagyományossal megegyezik. [9]

Láthatjuk tehát, hogy több alternatív megvalósítás közül választhatunk a hányadoseszték során, így kihasználhatjuk a különböző implementációk nyújtotta előnyöket. A hányadoseszt után már adott a bázist elhagyó és a bázisba belépő változó, így elvégezhető a báziscsere és az ahhoz kapcsolódó transzformációk.

2.4. A degeneráció

A degenerált bázis fogalmát már korábban ismertettük, ilyenkor tehát valamely bázisváltozó a korlátján van. Mint azt már korábban említettük, a pricing és a hányadoseszt együttesen határozzák meg a pivotelemet, a hányadoseszt elvégzése után tudjuk egyértelműen, hogy az aktuális iterációban várható-e javulás. Léteznek a pricing-hoz kapcsolódó heurisztikus degeneráció-ellenes technikák, de azt jellemzően a hányadosesztben lehet hatékonyan kezelni.

Definiálhatjuk a duál degenerált bázist, ez olyan bázismegoldást jelent, amiben valamilyen redukált költség 0, azaz a megengedettségi tartományának határán van: $\mathcal{D} = \{j: d_j = 0\}$. Ennek eredményeként a hányadosesztben 0 értékű hányadosok kerülhetnek definiálásra, melyek kiválasztása duál degenerált iterációhoz vezet, vagyis nem javul a duál célfüggvény értéke. Az első és a második fázisban egyaránt jelentkezhethet ez a probléma, alapvetően tehát mindkét helyen foglalkozni kell a degeneráció kezelésével.

A degeneráció okozhatja az algoritmus ciklizálását (cycling), ilyenkor egy k darab bázisból álló sorozat a megoldás során újra és újra ismétlődik. Emellett jelentkezhethet a helyben járás (stalling) jelensége is, mely során a célfüggvény-érték a folyamatos báziscserék ellenére sem

javul. Összességében a megoldást nagy mértékben hátráltatják ezek a jelenségek, ezért érdemes kiemelt figyelmet fordítani ezen problémás szituációk kezelésére.

2.4.1. Lehetséges okai

Ha megvizsgáljuk az algoritmus működését azt láthatjuk, hogy minden olyan nem-bázis változó, melynek redukált költsége 0-hoz nagyon közeli, degenerált tulajdonságokat mutat a gyakorlatban. [2] Ennek vizsgálatára érdemes bevezetni egy toleranciát, melynek értelmében a degenerált pozíciókat egy tolerancia segítségével definiáljuk: $\mathcal{D} = \{j: |d_j| < \epsilon_{degen}\}$.

Sok olyan feladattal találkozhatunk, amelyek nagyon degeneráltak, ilyen például a megengedettségi probléma, amelyben konstans 0 célfüggvény mellett keresünk megengedett megoldást. Ebben az esetben a c költségegyütthatókat tartalmazó vektor azonosan nulla, ebből következően a redukált költségek is nullák lesznek, az ilyen feladatok tehát nagyon degeneráltak. Másik esetben például, ha az A mátrix oszlopai hasonló értékeket tartalmaznak, és ezek az oszlopok nincsenek bázisban, akkor egy pont körül definiálhatunk hányadosokat, amik kiválasztása szintén degenerációhoz vezethet.

Most pedig tekintsük át, hogy melyek a leggyakrabban alkalmazott degeneráció-ellenes stratégiák. Az egyik ilyen a Harris hányadoseszt, illetve az azon alapuló EXPAND eljárás. [10] [11] Ezen technikákat a primál algoritmus második fázisához találták ki, de mindegyik átvihető a duál algoritmus logikájába, viszont ennek kérdése nem triviális, mint azt a későbbiekben látni fogjuk. Munkánk során a primál második fázisra ismertetett leírás alapján dolgoztunk [2], mivel nem találtunk kimondottan duál algoritmusra vonatkozó pontos leírást ezekre az eljárásokra. Az alábbiakban tekintsük át ezek működését, az eljárások duál megfelelőit a következő fejezetben ismertetjük, hiszen az már saját kutatásunk részét képezi. Ezen módszerek mellett a későbbiekben bemutatjuk a perturbációs eljárásokat, és azok működését is.

2.4.2. Harris hányadoseszt

Ebben a módszerben a hányadosok definiálása két iterációban történik. Az eljárás alapja, hogy egy fix értékű toleranciát használ a hányadosok kiterjesztéséhez, így a pivotelem kiválasztását tekintve rugalmassá válik a hányadoseszt. Ennek értelmében a degeneráció veszélyét hordozó, egy pontban lévő hányadosok egymástól kis mértékben eltávolodnak, aminek segítségével hatékonyan csökkenthető a degenerált iterációk száma. Legyen adott tehát egy rögzített tolerancia, esetünkben az ϵ_{feas} megengedettségi tolerancia.

Tegyük fel, hogy $t \geq 0$ esetben vagyunk a primál második fázisban, a $t \leq 0$ eset analóg módon tárgyalható, annak részletezésétől a módszer ismertetése során eltekintünk. A részletesen bemutatott duál hányadoseszthez hasonlóan a primál második fázisban olyan t értékekre kell hányadost definiálnunk, amellyel a primál változók megengedettségi tartományuk határá-

ra kerülnek. Az alábbi összefüggések alapján definiáljuk a hányadosokat a primál második fázisú hányadosesztben, az LP feladat CF-1-es alakját feltételezve [2]:

$$t_i = \begin{cases} \frac{\beta_i}{\alpha_q^i}, & \text{ha } \alpha_q^i > 0 \\ \frac{\beta_i - u_{\mathcal{B}_i}}{\alpha_q^i}, & \text{ha } \alpha_q^i < 0 \end{cases} \quad (16)$$

Az első esetben a véges alsó korláton definiálunk hányadost (CF-1-es alakban ez a korlát 0), a második esetben pedig felső korláton ($u_{\mathcal{B}_i}$). Ennek értelmében minden változó pontosan a korlátjainak megfelelően definiálja a hányadosokat. A Harris hányadosesztben emellett toleranciával kiterjesztett hányadosokat is definiálunk az alábbiak szerint:

$$t_i = \begin{cases} \frac{\beta_i + \epsilon_{feas}}{\alpha_q^i}, & \text{ha } \alpha_q^i > \epsilon_{pivot} \\ \frac{\beta_i - u_{\mathcal{B}_i} - \epsilon_{feas}}{\alpha_q^i}, & \text{ha } \alpha_q^i < -\epsilon_{pivot} \end{cases} \quad (17)$$

A hányadosokat tehát ϵ_{feas} -nek megfelelően relatívan kiterjesztjük. Fontos megjegyezni, hogy az egyes α_q^i értékek vizsgálatakor egy rögzített pivot tolerancia értéket használunk, ami a kiválasztható generálóelemek minimális nagyságrendjét határozza meg. Ezzel a módszerrel kizárhatjuk azokat a hányadosokat, amelyekhez tartozó generálóelem nagyon kis nagyságrendű, aminek jelentőségére a hatékony implementációt tárgyaló fejezetben még visszatérünk.

Az Harris féle hányados-választás két iterációjának menete a következő:

Első iterációban számítsuk ki a kiterjesztett hányadosokat, majd ezek közül vegyük a legkisebbet, és jelöljük θ_1 -gyel. Feltételezzük továbbá, hogy a bejövő változót felső korlátja θ_1 -nél kisebb elmozdulás esetén nem korlátozza, ellenkező esetben végezzünk boundflip-et. Az így meghatározott érték egy felső korlát, amennyiben ennél kisebb lépéshosszhoz tartozó pivotelemet választunk, minden bázisváltozó a toleranciával kiterjesztett korlátokra vonatkozóan megengedett marad.

Második körben számítsuk ki újra a hányadosokat, ezúttal az egzakt korlátokkal számolva, majd a θ_1 -nél kisebb értékeket tegyük egy $\mathcal{S}$ halmazba. Ezek után pedig vegyük azt a hányadost ebből a halmazból, amelyhez tartozó α_q^i generálóelem abszolút értékben a legnagyobb. Az így definiált hányados legyen θ_p , azaz primál lépéshossz. Abban az esetben, ha θ_p negatív (ez akkor fordulhat elő, ha a kiválasztott változó csak a kiterjesztett korlátján belül volt megengedett, az eredetin már nem), akkor a lépéshossz értékét 0-ra állítjuk, és így megelőzhetünk egy rontó iterációt eredményező változó kiválasztását.

A Harris hányadosesztben tehát a kiválasztott legnagyobb abszolút értékű pivotelemmel végezzük el a báziscserét, a lépéshosszt pedig a hozzá tartozó t hányados, vagy a fixen 0 érték közül választjuk. Ennek eredményeképpen a változók nem megengedettségének mértéke sosem haladja meg az ϵ_{feas} értéket, a degeneráció mértéke pedig csökken.

2.4.3. EXPAND eljárás

Ez az eljárás a Harris hányadoseszt kiterjesztéseként értelmezhető, attól két pontban tér el lényegesen. [12] Az első jelentős eltérés, hogy ebben az esetben nem egy rögzített konstans toleranciával számolunk, hanem minden iterációban kis mértékben növeljük a tolerancia értékét. Ennek értelmében a tolerancia a k . iterációban $\epsilon_{feas_k} = \epsilon_{feas_{k-1}} + \tau$ lesz, ahol τ az inkrementálás mértéke, ez minden iterációban állandó. Az iterációk során ϵ_{feas_k} sosem haladhatja meg a maximális tolerancia, ϵ_{feas} értékét. A kiindulási tolerancia, ϵ_0 a maximális tolerancia valahányszorosaként kerül inicializálásra $\epsilon_0 = \epsilon_{feas}\delta$, ahol δ előre megadott paraméter. Amennyiben az iterációk során $\epsilon_{feas} < \epsilon_{feas_k}$ teljesül, a korábban ismertetet iteratív szabály helyett az $\epsilon_{feas_{k+1}} = \epsilon_0$ lépést alkalmazzuk, ezzel megelőzhető, hogy túl nagymértékű nem-megengedettség alakuljon ki. Az EXPAND során tehát egyre nagyobb szabadságot biztosítunk az algoritmusnak a toleranciát tekintve, az eljárás elnevezése a kiterjesztett toleranciából adódik (EXPanding tolerance ANti Degeneracy).

A Harris hányadoseszthez hasonlóan ekkor is két körben definiáljuk a hányadosokat, ez a fenti leírással analóg módon történik, a második kör végén az $\mathcal{S}$ halmazból vett legnagyobb abszolút értékű generálóelemet vesszük, ebben sincs különbség. A kiválasztott változóhoz tartozó hányadost azonban nem nullával, hanem egy minimális előrehaladást biztosító értékkel hasonlítjuk össze. Ez nem más, mint az iterációnként konstans tolerancia-inkrementálásból és a meghatározott legjobb generálóelemből számolt hányados, tehát $\theta_{min} = \frac{\tau}{\alpha_q^i}$. Abban az esetben, ha a kiválasztott hányados értéke ennél kisebb, akkor átállítjuk a θ_p lépéshossz értékét θ_{min} -re. Ez a gyakorlatban azt jelenti, hogy a következő iterációban a mostanihoz képest τ mértékkel nőhet a nem-megengedettség egyes változókra, azonban az biztosan az inkrementált toleranciaértéken belül marad. Így minden iterációban legalább θ_{min} -nek megfelelő javulást biztosítunk a célfüggvényben. Így gyakorlatilag kizárható a ciklizálás jelensége, hiszen az algoritmusnak köszönhetően szinte sohasem kerülhetünk ugyanabba a bázisba, legfeljebb az újrainicializálás során lehet periodikus a viselkedés, azonban ennek valószínűsége nagyon kicsi.

Az eddig leírt technikák hatékonyan segítenek a primál második fázisú degeneráció kezelésében. Viszont, ahogy azt már korábban említettük szükség van az első fázisú degeneráció kezelésére is, ezért a továbbiakban egy olyan általános technikát szeretnénk bemutatni, amely

mindkét fázisra vonatkozóan általánosan jó tulajdonságokat a mutat a degenerációra vonatkozóan.

2.4.4. Célfüggvény együtthatók perturbálása

A korábban említett megengedettségi problémában a konstans nulla célfüggvény okozott degenerációt, általánosan, ha a c vektor elemei egymáshoz nagyon hasonlóak, akkor fellép ez a jelenség. A szakirodalomban széleskörűen alkalmazott megoldás a perturbálás, jelen esetben a célfüggvény együtthatóinak perturbálása, vagyis kismértékű eltolása. Legyen a módosított célfüggvényünk tehát $\bar{c} = c + \delta$. Ezen δ vektor elemeinek megválasztására eltérő stratégiákat találhatunk, nincs egyértelműen hatékony módszer, hiszen ezek heurisztikus eljárások. [3] Azon kérdésre, hogy csak a strukturális, vagy a logikai változók célfüggvény együtthatóját is perturbáljuk, eltérő elméleteket találunk.

A perturbálás során minden egyes c_j együtthatót különböző, kis nagyságrendű, véletlenszerűen generált δ_j értékekkel eltoljuk, ezáltal az egy pontban definiált redukált költségeket mesterségesen eltávolítjuk egymástól valamilyen irányba. Amennyiben az eltoló célfüggvényre optimális megoldást találunk, vissza kell állítanunk az eredeti célfüggvényt, majd erre vonatkozóan ellenőrizni kell az optimális megoldást. Ez a gyakorlatban legfeljebb néhány további iterációt jelent. Általában elmondhatjuk, hogy amennyiben valamilyen véletlenszerűen generált értékkel végezzük a perturbálást, az javít a feladat degeneráltságán, azonban a javulás mértéke különböző módszereknél igen eltérő lehet, akár monoton, de lassabb konvergenciát is eredményezhet.

Most pedig tekintsük át egy, a szakirodalomban fellelhető célfüggvény perturbálási eljárás lényegét. [3] Ezen eljárásban az egyes δ_j értékek meghatározása egy négylépéses szabály alapján történik:

1. Fix és relatív rész meghatározása: A perturbálási tényező ilyen mértékű felbontásának lényege, hogy a rögzített rész biztosít egy minimális eltolást, emellett egy a célfüggvény együttható nagyságrendjétől függő relatív részt is figyelembe veszünk a perturbáció során. Így az egyes elemeket nagyságrendjüktől függően, különböző értékekkel változtatjuk meg. A fix része legyen $100\epsilon_{opt}$, a relatív rész pedig Ψc_j , ezek összegét jelöljük δ_{fr_j} -vel, ahol ϵ_{opt} az optimalitási tolerancia, Ψ pedig egy paraméter, melynek alapértelmezett értéke legyen 10^{-5} .
2. A perturbálás irányának meghatározása: A különböző változótípusokra ismertettük az optimalitási feltételeket, ez a felső korláton lévő változókra $d_j \leq 0$, az alsó korláton lévőkre $d_j \geq 0$. Ezek alapján, hogy az eltolás megengedett irányba történjen, a véges felső korláttal rendelkező változókat negatív irányba, a többi változót pedig pozitív irányba toljuk el. Véges felső korlátú változóknál legyen $\delta_j = -0,5\delta_{fr}(1 +$

μ), a többi változóra ennek -1 szerese. Az összefüggésben szereplő μ egy $[0,1]$ intervallumból vett véletlen szám, így tehát minden j -re különböző értékeket kapunk.

3. Ritkásság kihasználása: Az egyes $\mathbf{a}_j$ oszlopvektorok nem-nulla elemeinek száma eltérő, ezért érdemes a ritkább vektorokat kisebb mértékben perturbálni, mint a telítettebbeket. Ez a pivot-választás során a ritkább oszlopokat fogja előnyben részesíteni, így a bázis ritkásságának fenntartását figyelembe vehetjük. Ennek érdekében készíthetünk egy $\mathbf{w}$ súlyvektort, melynek egyes elemével súlyozhatjuk a perturbáció mértékét az $\mathbf{a}_j$ oszlopvektorok nem-nulla számának alapján. Az így kapott vektorral súlyozzuk δ -t. Egy ilyen $\mathbf{w}$ vektor lehet például a $[10^{-2}, 10^{-1}, 1, 2, 5, 10, 20, 30, 40, 100]^T$ vektor. Amennyiben egy vektor nem-nulláinak száma nagyobb, mint $\mathbf{w}$ dimenziója, alkalmazzuk annak utolsó elemét.

4. A δ_j értékek nagyságrendjének vizsgálata: Érdemes az előző lépésekből kapott perturbálási tényező nagyságrendjét egy adott intervallumon belül tartani, hogy egy jól meghatározott nagyságrendből kerüljenek ki ezek a perturbáló értékek. Ezen intervallum meghatározásakor célszerű figyelembe venni az eltolás fix és relatív részét, valamint a célfüggvény együtthatók átlagos nagyságrendjét is. Ennek értelmében meghatározhatunk két korlátot, legyen $\delta_{min} = \min\{10^{-2}\epsilon_{opt}, \Psi\}$ az intervallum alsó határa, valamint $\delta_{max} = \max\{10^3\epsilon_{opt}, 10\Psi_{avg}|c|\}$ az intervallum felső határa, ahol $avg|c|$ a célfüggvény együtthatók abszolút értékének számtani közepe.

Az itt bemutatott degeneráció-ellenes stratégiákat megvalósítottuk és megvizsgáltuk. Az erre vonatkozó eredményeinket a következő fejezetben ismertetjük.

3. Hatékony megvalósítás

Ebben a fejezetben részletesen bemutatjuk az általunk megvalósított hányadoseszteket, illetve az azokhoz kapcsolódó legfontosabb gyakorlati kérdéseket. Az első alfejezetben ismertetjük a nagyméretű LP feladatokban rejlő kihívásokat és nehézségeket, valamint, hogy ezek elkerülésére milyen fejlett technikákat valósítottunk meg.

A második alfejezetben bemutatjuk az általunk implementált duál hányadosesztek felépítését, a program struktúráját, és a megvalósított osztályokat a kutatócsoport által fejlesztett PanOpt LP megoldó szoftverben. Tárgyaljuk továbbá a szoftver sajátosságait, és a benne rejlő lehetőségeket.

3.1. Egy nagyméretű LP megoldó implementációs kihívásai

A gyakorlati életből vett lineáris programozási feladatokra a nagy méret mellett jellemzők a különböző numerikus nehézségek. Éppen ezért előfordulhat, hogy adott esetben egy kisebb LP feladat megoldása sokkal magasabb szintű implementációt követel meg, mint egy nála nagyobb, de numerikusan jóindulatú feladat. [2] Tekintsük át, hogy milyen kihívásokkal kell szembenézni egy ilyen LP megoldó szoftver implementációja során.

A feladat mérete az $A \in \mathbb{R}^{m \times n}$ együtthatómátrix dimenziójával, valamint a benne szereplő nem-nullák számával számszerűsíthető. A mátrix mérete leírja, hogy az adott feladatban mennyi döntési változó (n), illetve mennyi feltétel (m) szerepel. Általában ezen együtthatók tárolására dupla pontosságú lebegőpontos (double) aritmetikát célszerű használni, egy ilyen szám mérete 8 bájt. Így, ha egy $100\,000 \times 100\,000$ -es feladatot tekintünk, akkor a teljes mátrix tárolása körülbelül 74,5 GB memóriát igényelne, ami még a mai fejlett technikákkal sem megvalósítható, nem beszélve arról, hogy LP megoldással már az első számítógépek megjelenése óta foglalkoznak.

Mi az, ami mégis lehetővé teszi az ilyen feladatok megoldását? Általában az ilyen méretű feladatok jellemezhetők egy úgynevezett ritkassági vagy telítettségi aránnyal, ami az adott feladatban szereplő nem-nulla és nulla elemek arányára ad információt. A ritkassági arány jellemzően 1% alatti, vagy akár annak töredéke. [2] Így, ha csak a nem-nulla elemeket, és azok pozícióit tároljuk, akkor kezelhetővé válnak az egészen nagyméretű mátrixok is.

Mint már említettük, a nagy méret mellett a valós életből vett LP feladatok másik nehézségét a rossz numerikus tulajdonságok adják. Ez jelentheti például azt, hogy az A mátrix elemeinek szórása nagy. Általános probléma, ha adott két szám, melyek nagyságrendje lényegesen eltér, akkor bár önmagukban még pontosak lehetnek az értékek, de a másik szám nagyságrendjéhez képest a lebegőpontos aritmetika sajátosságai miatt már nem. Példa lehet erre, ha egy 10^{10} -es nagyságrendű számot összeadunk egy pici 10^{-10} nagyságrendű számmal, akkor az eredmény pontosan a nagyobb számmal egyezik meg, ami nyilvánvalóan helytelen.

Egy másik probléma forrása lehet, ha adott egy hosszabb komplex művelet sor eredménye, és egy fix érték, ami a művelet pontos eredménye. Mivel nem pontosan ábrázoljuk a számokat, ezért mindig van numerikus hiba, ami a különböző műveletek során felhatványozódhat. Ez akár olyan méreteket is ölthet, hogy a műveleti eredmény és a várt eredmény közti eltérés összemérhető nagyságrendű. Az ebből származó hibák jelentőségét nem nehéz belátni.

Kiemelkedő jelentőségű hibaforrás lehet, ha a meghatározott pivotelem, amivel a báziscserével járó transzformációk során számolunk, nagyon kis nagyságrendű (pl. 10^{-10}), mert az így számított értékek közti nagyságrend jelentősen megnő, ami a már említett problémákat okozhatja. Az ilyen generálóelemek általában numerikus szemetek, vagy pontatlan számítások eredményei.

Ezek és ehhez hasonlóak azok az akadályok, amiket egy mai korszerű LP megoldónak különösebb nehézség nélkül kell megoldania, mindamellett, hogy a gyors futást garantálja, hiszen mint azt már többször említettük, egy nagyméretű egészértékű optimalizálási feladat megoldása akár több százezer relaxált LP feladat megoldását is jelentheti.

Az eddig tárgyalt numerikus hibák eredményeként előfordulhat, hogy az algoritmus rossz célfüggvény-értéket fogad el optimálisnak, esetleg egy feladatot helytelenül nem-korlátosnak vagy nem-megengedettnek nyilvánít. Előfordulhat továbbá, hogy egy adott javító változóhoz nem talál megfelelő hányadost a hányadosteszt, egy iterációról kiderül, hogy valójában ront a célfüggvény-értéken, az algoritmus visszaeshet első fázisba egy rossz második fázisú iteráció után, esetleg a bázis numerikusan szingulárisává válhat.

Láthatjuk tehát, hogy rendkívül sokrétűek az implementációs kihívások és nehézségek, a fent felsoroltak csupán néhány jelentősebb hibát képviselnek, de ettől akár egész eltérő jelenség is felléphet a szimplex algoritmus során.

3.2. Megvalósított technikák és algoritmusok

Ebben az alfejezetben szeretnénk bemutatni, hogy mik is azok az algoritmusok, amiket megvalósítottunk a PanOpt rendszerben, valamint részletesen bemutatjuk ezek működését. Tárgyaljuk továbbá, hogy az előző alfejezetben ismertetett numerikus hibák, és a degeneráció elkerülésére milyen saját technikákat dolgoztunk ki.

A kutatócsoport által fejlesztett PanOpt egy kutatási célra szánt nagyméretű feladatok megoldására alkalmas LP megoldó szoftver. Legfőbb célja, hogy a szimplex algoritmus implementációjához kapcsolódó fejlett technikák vizsgálatára, valamint újabb hatékony módszerek kutatására adjon lehetőséget. Mivel a szoftver valós, nagyméretű feladatok megoldására hivatott, ezért elvárható, hogy a feladat méretétől függetlenül hatékonyan és megbízhatóan jusson el az optimális megoldásig, amennyiben az létezik. Fontos megjegyezni továbbá, hogy a megoldási folyamat során a bázis egy paraméterben meghatározott gyakorisággal invertálásra (vagy faktorizációra) kerül, két invertálás (faktorizálás) között pedig csak frissí-

tésre (update) kerül. Ez a megvalósítás jellemző minden hatékony LP megoldóra, a bázis megfelelő alakjának újraszámolása kritikus a pontosság fenntartásában, azonban így kellően gyors a bázis frissítése is.

3.2.1. A PanOpt által használt matematikai modell

Mint azt már említettük, érdemes végiggondolni, hogy az LP feladatot milyen módon reprezentáló alakot használunk, hiszen ezek más és más előnyöket jelenthetnek a megoldás során. Az általunk használt alak a következő:

$$\begin{aligned} \min \mathbf{c}^T \mathbf{x} \\ \text{s.t. } \mathbf{Ax} = \mathbf{b} \\ \text{type}(x_j) \in \{0,1,2,3,4\}, j = 1,2, \dots, n \end{aligned} \quad (18)$$

Láthatjuk, hogy részben hasonlít a már ismertetett CF-1-es alakra, azonban a változótípusok lazább feltételekkel vannak definiálva. A célfüggvényt minden esetben minimalizáljuk, ez kedvező a megoldás szempontjából, hiszen a maximum feladatok könnyen átalakíthatók minimum feladatra, és így az optimalitási feltételeket elegendő minimalizálási esetben implementálni. Ennek megfelelően elkerülhetjük a sok különböző eset kezelését, az ebből következő hibalehetőségeket, valamint megspóroljuk a feltételvizsgálatokhoz szükséges időt. Egy lényeges eltérés az CF-1 számítási alakhoz képest a változótípusok definíciója, tekintsük át ezeket:

- type 0: fix típusú változók. $l_j = x_j = u_j$
Megegyezik az alsó és felső korlát értéke, az eddigihez képest annyi különbséggel, hogy ez az érték nem feltétlen 0, bármilyen előjelű lehet.
- type 1: korlátos változók. $-\infty < l_j \leq x_j \leq u_j < +\infty$,
Akármilyen valós korláttal rendelkezhetnek.
- type 2: alsó korlátos változók. $-\infty < l_j \leq x_j < +\infty$,
Rendelkeznek egy rögzített alsó korláttal, ez szintén bármilyen érték lehet.
- type 3: szabad változók. $-\infty < x_j < +\infty$,
Ezeket a CF-1-es alaknak megfelelően kezeljük.
- type 4: mínusz típusú, vagy felső korlátos változók. $-\infty < x_j \leq u_j < +\infty$,
Rendelkeznek egy felső korláttal, ami a többi változóhoz hasonlóan bármilyen konstans lehet.

Ez az alak azért előnyös, mert nem kell elő-feldolgozások során átkonvertálni a változótípusokat, eltolni a korlátokat, ezzel a numerikus hibák lehetőségét lecsökkentjük, illetve ha sok LP feladatot kell megoldani, akkor sem kell elő-feldolgozásra pazarolni az erőforrásokat.

A már korábban ismertetett optimalitási feltételek ebben az alakban az alábbira módosulnak (3. táblázat):

3. táblázat: Optimalitási feltételek a PanOpt matematikai modelljében

$type(x_j)$	a változó értéke	optimalitási feltétel
0	$x_j = l_j = u_j$	d_j tetszőleges
1	$x_j = l_j$	$d_j \geq 0$
1	$x_j = u_j$	$d_j \leq 0$
2	$x_j = l_j$	$d_j \geq 0$
3	$x_j = 0$	$d_j = 0$
4	$x_j = u_j$	$d_j \leq 0$

Lényeges változás a korábbihoz képest, hogy a változók alsó korlátja tetszőleges lehet, ezért az általunk használt modell egy általánosabb formalizmusnak tekinthető.

3.2.1. Toleranciaértékek használata

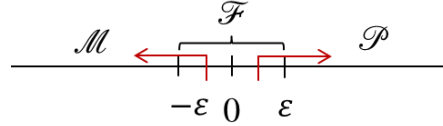
Az előbbieken már ismertettük, hogy a különböző komplex számításoknak köszönhetően felskálázódhatnak a lebegőpontos aritmetikából következő pontatlanságok, ezért a számított értékek vizsgálatára különböző toleranciaértékeket vezettünk be. Ennek megfelelően sok esetben nem azt vizsgáljuk, hogy két érték megegyezik-e, hanem egy $|a - b| \leq \epsilon$ alakú egyenlőtlenséget, ahol a megengedett maximális eltérés mértéke ϵ . Ennek értéke attól függően, hogy milyen vizsgálódásról van szó, eltérő lehet. [10] Például, a duál megengedettség vizsgálatakor sosem az egzakt korlátokat vizsgáljuk, hanem ezek egy kiterjesztett változatát.

Egy szabad változó redukált költségét duál megengedettnak tekintjük, ha teljesül rá az $-\epsilon_{opt} \leq d_j \leq \epsilon_{opt}$ egyenlőtlenség, ahol ϵ_{opt} az optimalitási tolerancia. A tolerancia analóg módon értelmezhető a többi változótípusra is. A PanOptban ezen tolerancia alapértelmezett értéke 10^{-8} . Így egy szabad változó valójában 5 különböző állapotban lehet, ezek nem megfelelő kezelése rontó iterációhoz vezethet. Ennek oka, hogy a hányadosok definiálásakor alapvetően meghatározó a redukált költségek előjele, hiszen más és más feltétel vonatkozik a különböző megengedettségi állapotokra. Tekintsünk át az alábbi példát a probléma megértése érdekében.

Legyen adott egy megengedett szabad típusú változó, tehát $d_j = 0$. Mint azt már ismertettük, egy ilyen változó az $\mathcal{M}$, $\mathcal{F}$ és $\mathcal{P}$ halmazok bármelyikében lehet, illetve a halmazok határán definiál hányadost. Az $\mathcal{M}$ felé mozgáshoz vagy $t \geq 0$ és $\alpha_j^p > 0$, vagy $t \leq 0$ és $\alpha_j^p < 0$ előjelek szükségesek, $\mathcal{P}$ halmazba pedig a $t \geq 0$ és $\alpha_j^p < 0$, vagy $t \leq 0$ és $\alpha_j^p > 0$ esetekben kerülhet. Minden esetben a definiált d_j/α_j^p hányados d_j miatt 0. Ha ugyanez a változó negatív irányba mozogva toleranciával tekintve megengedett, tehát ϵ_{opt} -nál kisebb mértékben negatív, vagy pozitív irányba haladva ϵ_{opt} -nál kisebb mértékben pozitív, akkor a definiált hányados $t \geq 0$ esetben negatív, $t \leq 0$ esetben pozitív, ami pont ellentétes a többi definiált hányadossal.

Ennek figyelmen kívül hagyása rontó iterációhoz vezet, hiszen $\Delta f = -tv_j$. Ezt a jelenséget, mivel a toleranciával tekintett megengedettségből következik, fake feasibilitynek (hamis megengedettség) neveztük el, erre vonatkozóan nem találtunk általános leírást a szakirodalomban, bár az egyes degeneráció-ellenes eljárások bizonyos mértékben implicit módon foglalkoznak ezzel a jelenséggel.

Számegyenesen ábrázolva (3. ábra) egy szabad változóra az alábbi irányok jelentenek problémát:



3. ábra: A fake feasibility jelensége

Ezen probléma elkerülésére a következő technikát alkalmaztuk. A hányadosesztben a hányadosok abszolút érték szerinti sorba rendezése elrejti ezt a hibát, pedig az előjel különbség jó lehetőséget nyújt a hibás hányadosok kiszűrésére. Megfigyeltük, hogy a negatív hányadosok -1 -szeresét véve, és ezeket sorba rendezve minden esetben a jó töréspontok pozitívak, a hamis megengedett töréspontok pedig ezektől eltérően negatívak lettek. Így, a sorba rendezés előtt, egy egyszerű előszűréssel könnyen megszabadulhatunk a hibás, negatív hányadosoktól. Ezzel a módszerrel a törtvonalfüggvénybe nem számoljuk bele a hamis megengedett töréspontokat, ezzel elkerülve az ebből származó pontatlan célfüggvényt. A szűrés során ennek érdekében korrigálnunk kell a törtvonalfüggvény meredekségét az $s_k^p = s_{k-1}^p - \left| \alpha_{j_k}^p \right|$ összefüggés alapján, így a hibásan belekerült töréspontokat kihagyjuk a meredekségből, hiszen azok nem javító változók. Ez a technika, mivel a rontó iterációk elkerülését segíti, egyfajta degeneráció-ellenes stratégiának tekinthető, a probléma ilyenfajta kezelése újszerűnek tekinthető.

Az optimalitási tolerancia mellett a PanOptban megengedettségi toleranciát (ϵ_{feas}) is használunk, aminek megfelelően egy változót megengedettnak tekintünk, ha az előzőhöz hasonlóan kielégíti kiterjesztett korlátait, vagyis $l_j - \epsilon_{feas} \leq x_j \leq u_j + \epsilon_{feas}$ teljesül. [3] Megjegyzésként hozzáfűznénk, hogy a toleranciák elnevezése a primál analógiát követi.

További két fontos tolerancia az abszolút (ϵ_{abs}) és a relatív (ϵ_{rel}) tolerancia. Abszolút toleranciát elsősorban arra használunk, hogy a nála kisebb értékeket nullának tekintjük. Másrészt a már ismertetett probléma miatt, miszerint egy számítás elvégzése után nem tudjuk a számítási eredmény pontos értékét, így amennyiben két érték eltérése abszolút értékben kisebb, mint az abszolút tolerancia, akkor ezeket egyenlőnek tekintjük. Emellett természetesen, ha az $|a + b| < \epsilon_{abs}$ egyenlőtlenség igaz, akkor az összeget 0-nak tekintjük. A relatív toleran-

cia értelmében két szám összegét akkor is 0-nak tekintjük, ha teljesül a következő egyenlőtlenség: $(|a| + |b|)\epsilon_{rel} \geq |a + b|$.

Ezen toleranciák használata egy kis plusz szabadságot biztosít a simplex algoritmusnak, ez által sok numerikus hibát elkerülhetünk, valamint a numerikus számítási szemetek eliminálásával a ritkásság is fenntartható, viszont annak kérdése, hogy a megoldó mely részeiben hogyan kell ezeket alkalmazni, mint azt a fenti tárgyalásban láthattuk nem triviális.

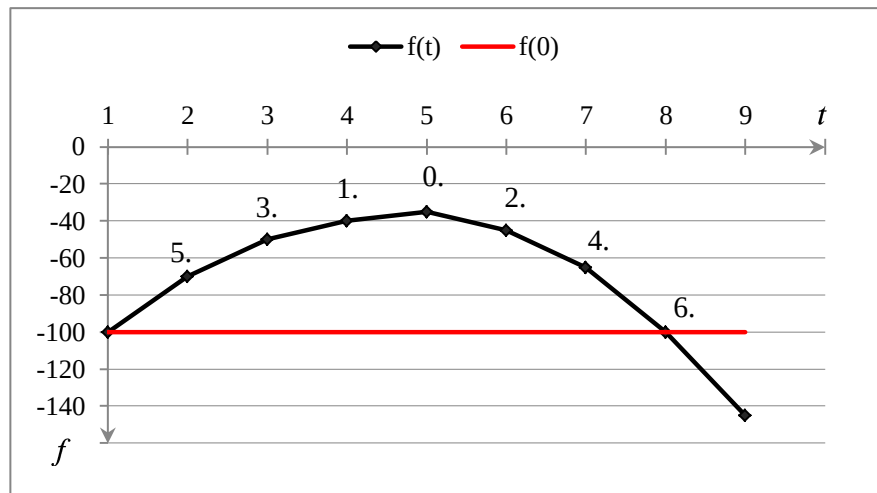
3.2.2. Stabil pivotálási eljárás

Mint már említettük, a törtvonalfüggvényes hányados-meghatározási módszer jóval több lehetőséget rejt magában, mint a hagyományos egy lépéses megvalósítás, most az első fázisú szakaszonként lineáris függvényes módszer nyújtotta lehetőségek egy hatékony kihasználására szeretnénk kitérni. A már ismertett kis nagyságrendű pivotelemből származó problémákat az alábbi algoritmus megvalósításával sikerült elkerülni. Tegyük fel, hogy a hányadosesztben meghatározott maximális lépéshez tartozó α_j^p generálóelem numerikusan kicsi. Ennek vizsgálatára a már korábban ismertett pivot toleranciát (ϵ_{pivot}) használjuk, melynek alapértelmezett értéke 10^{-6} . Mivel a törtvonalfüggvény sok esetben több különböző mértékű javító alternatívát kínál, ezért szélesebb választék áll rendelkezésünkre a lehetséges pivotelemeket tekintve.

A törtvonalfüggvényes hányados-meghatározás általános jellemzője nagyméretű feladatoknál, hogy egy iterációban rengeteg, akár több ezer töréspontot is definiálhatunk, és ebből sok esetben csupán néhányat használunk fel. Ezért egy olyan algoritmust célszerű választani a hányadosok rendezésére, amely egyesével rendezi az adatokat, hiszen az összes töréspont rendezése rengeteg felesleges munkát jelentene. Ilyen esetben a kupacrendezés (heapsort) algoritmus a leghatékonyabb, a PanOptban is ez a módszer került megvalósításra.

A törtvonalfüggvény maximumának helyétől elindulva, vizsgáljuk meg célfüggvény-érték szerint csökkenő sorrendben az adott töréspontokhoz tartozó lehetséges generálóelemeket, majd, ha a toleranciánál nagyobb értéket találunk, akkor a megfelelő változót jelöljük belépő változónak. Ha a célfüggvény meredeksége negatív, de a kezdeti célfüggvény-érték feletti tartományban van az adott töréspont, akkor az még egy potenciálisan jó generálóelem lehet. Ennek megfelelően mindkét irányban, a maximumhely előtt és az után is végezhetjük a keresést. Ezen keresési tér korlátja a kiindulási célfüggvény-érték, hiszen, ha ez alatti értékhez tartozó változót hoznánk be a bázisba, az rontó iterációhoz vezetne, ezáltal hátráltatná a megoldást.

A megértés elősegítése érdekében tekintsük az alábbi törtvonalfüggvényt (4. ábra), ahol láthatjuk, hogy milyen sorrendben vizsgáljuk az egyes töréspontokat:



4. ábra: Első fázisú stabil pivotálási eljárás keresési tere

Első lépésben az algoritmus megvizsgálja, hogy van-e további, illetve a kiválasztott maximumhely előtti töréspont, ezt követően ezeket összehasonlítja, és a nagyobb célfüggvényértékűhöz tartozó generáloelemnek elvégzi a numerikus vizsgálatát, majd iteratíván lefelé haladva folytatja a keresést a már ismertetett korlátig. Amennyiben az eljárás nem talál jó pivotelemet, akkor üzenetet küld erről a vezérlőnek, ami a pricing-ot új javító sor kijelölésére utasítja.

Ha a kis nagyságrendű generáloelemből következő hibát nem kezelnénk, akkor az a célfüggvény értékének romlásához, vagy egyéb numerikus problémához vezetne, ezért ez a megvalósítás egyfajta degeneráció ellenes stratégiának fogható fel. Újszerűsége abban rejlik, hogy a szakirodalom a maximumhelytől egyszerű visszalépést alkalmaztak, ezen probléma kezelésére.

Második fázisban az eljárás kicsit más formában valósítható meg, visszalépve már boundflip-re kijelölt változót találunk, a továbblépés pedig elméleti szinten hibás, hiszen sérülne a megengedettség, és az algoritmus visszaesne első fázisba. Ezért, a második fázisú hányadosesztben megvalósított algoritmus megvizsgálja, hogy van-e több töréspont a maximumhelyen, és ezek közül a numerikusan legjobbat adja vissza, amennyiben létezik a toleranciánál nagyobb érték. Ezt megtehetjük, hiszen előfordulhat, hogy egy adott pontban több töréspont is található. Az ezekhez tartozó célfüggvény-értékek megegyeznek, hiszen akármilyen meredekséget véve a nulla lépéshossz miatt nem változik a célfüggvény-érték, csupán a meredekség. Az eljárás nem módosítja az előzetesen korlátcsérés lépésre kijelölt váltókat, ezeket minden esetben elvégezzük.

3.2.3. Harris hányadoseszt, EXPAND

Tekintsük át a primál második fázisra ismertetett Harris hányadosesztet, és EXPAND eljárás duál második fázisú megfelelőjét. Amint azt korábban említettük ezen leírás saját kutatásunk eredménye, hasonló leírást nem találtunk a szakirodalomban, ezeket a 2.4.2-2.4.3 alfe-

jezetben leírtak alapján dolgoztuk ki. A korábbiakban ismertetett toleranciák közül a Harris hányadoseszt duál változatában az $\epsilon_{opt} = 10^{-8}$ optimalitási toleranciát használjuk. Ezen kívül szükségünk van továbbá az EXPAND eljárásban ismertetett δ tolerancia szorzóra, ennek alapértelmezett értéke megoldónkban 10^{-2} , a tolerancia újrainicializálása pedig 10000 iterációnként történik. A tolerancia inkrementálás mértéke az ismertetett számok alapján $\tau = \frac{(1-\delta)\epsilon_{opt}}{10000}$. Ezen értékek valamelyest eltérnek a szakirodalomban ajánlottaktól, ennek oka az, hogy a szakirodalmi leírással ellentétben mi a duál algoritmussal számolunk, ezért az EXPAND-hoz kapcsolódó paramétereket az általunk használt egyéb toleranciaértékekhez finomítottuk. Legyen ϵ_{work} az aktuális tolerancia, ez tehát Harris hányadosesztben 10^{-8} , EXPAND esetén pedig az inkrementált tolerancia aktuális értéke.

A kiterjesztett hányadosok definiálására az alábbi összefüggést alkalmazzuk $t \geq 0$ esetben:

$$t_j = \begin{cases} \frac{d_j - \epsilon_{work}}{\alpha_j^p}, & \text{ha } \alpha_j^p < -\epsilon_{pivot} \\ \frac{d_j + \epsilon_{work}}{\alpha_j^p}, & \text{ha } \alpha_j^p > \epsilon_{pivot} \end{cases} \quad (19)$$

Ennek értelmében a redukált költségek megengedettségi tartományuktól az adott iterációban legfeljebb ϵ_{work} mértékben térhetnek el. A felső korláton lévő nem-bázis változók, melyek redukált költsége felfelé változik, a felső összefüggésnek, az alsó korláton lévő nem-bázis változók, melyek redukált költsége lefelé változik, pedig az alsó összefüggésnek megfelelően definiálnak hányadosokat. Ezt követően meghatározhatjuk θ_1 -et az ismertetett hányados-meghatározási módszerek valamelyikével:

- hagyományos módszernél legyen $\theta_1 = \min\{t_j\}$,
- törtvonalfüggvényes algoritmusnál pedig a függvény maximumához tartozó hányados, vagyis $\theta_1 = \arg \max_t f(t)$.

Második körben, a meghatározott θ_1 értéknél kisebb, egzakt határon definiált hányadosokat kigyűjtjük az $\mathcal{S}$ halmazba, majd ezek közül a legnagyobb $|\alpha_j^p|$ -vel rendelkező hányadosot jelöljük θ_D -vel, ez lesz a javasolt duál lépéshossz. Az így kapott érték vizsgálatára vezessük be a következő változót:

$$\theta_{min} = \begin{cases} 0, & \text{Harris hányadosesztben} \\ \frac{\tau}{|\alpha_j^p|}, & \text{EXPAND eljárásban.} \end{cases} \quad (20)$$

Az így meghatározott érték lesz a garantálandó minimális előrehaladás az adott iterációban, ha a kiválasztott hányados ennél kisebb, akkor mesterségesen erre az értékre állítjuk θ_D -

t, és úgy végezzük el a báziscserét a maximális generálóelemhez tartozó változóval, tehát $\theta_D = \max\{\theta_D, \theta_{min}\}$.

Munkánk során vizsgáltuk a duál második fázisú EXPAND kiterjesztésének lehetőségét a törtvonalfüggvényes hányados-meghatározás módszerével. Ezen módszernél, mint azt ismertettük, a törtvonalfüggvény maximumhelye előtti töréspontok korlátcserés lépésre (boundflip) kijelölt változókhoz tartoznak. Amennyiben ezeket az eredeti eljárásban ismertetett módon vizsgáljuk, majd a kijelölt korlátcserés lépésekhez tartozó változókat kihagyjuk az EXPAND eljárásból, akkor a két módszer együttesen alkalmazható, így ötvöztethetjük a két eljárás előnyeit. Határozzuk meg θ_1 -et tehát úgy, hogy a legkisebb kiterjesztett töréspontoz tartozzon, amelyet nem jelöltünk korlátcserés lépésre. Az eljárás további része a korábbival analóg működésű.

Érdemes kiemelt figyelmet fordítani a pivot toleranciát (ϵ_{pivot}) kielégítő értékek által definiált hányadosokra. Mint már említettük, a kis nagyságrendű generálóelemeket nem választhatjuk ki a hányadosesztben rossz numerikus tulajdonságaik miatt. Amennyiben már a hányadosok definiálásánál kizárjuk az ilyen értékeket és a numerikusan rossz generálóelemhez tartozó hányadosokat nem generáljuk, egyszerűsíthetjük a hányados-választást, hiszen ilyen esetben nem kell utólag megvizsgálni a pivotelem nagyságrendjét. Ez természetesen okoz némi pontatlanságot, ennek értelmében a hányadosesztben meghatározott lépéshossz, és célfüggvény-érték kis mértékben megváltozhatnak, azonban ez nem jelentős. Amennyiben így definiáljuk a hányadosokat, akkor ez a megvalósítás egy alternatíva az ismertetett stabil pivotálás eljárásra, ellenkező esetben a 3.2.2 alfejezetben leírtak alkalmazására továbbra is szükség van. A Harris hányadoseszt ezen funkcióját megvalósítottuk az első fázisú duál hányadosesztben is, az erre vonatkozó mérési eredményeinket a következő fejezetben ismertetjük.

Az eljárás gyakorlati működéséhez érdemes a már korábban említett bázis újrainvertálás (újrafaktorizálás) kérdését megvizsgálni. Ennek értelmében invertáláskor (faktorizáláskor) újraszámolunk minden értéket, ezek között pedig frissítéssel (update) számolunk. Ez azt jelenti, hogy amennyiben nem történik θ_{min} -nél nagyobb lépéshosszú, tényleges javító iteráció két invertálás (faktorizálás) között, akkor az EXPAND eljárás által biztosított garantált előrehaladás a redukált költségek újraszámolása miatt elvész. Ennek elkerülésére adminisztráljuk a futás során az iterációk nagyságát, és ha két invertálás (faktorizálás) között csak az EXPAND által biztosított lépéshosszú előrehaladást észlelünk, a redukált költségeket nem számítjuk újra. Ennek alternatív megoldása a célfüggvény együtthatók eltolása egyelőre nem került megvalósításra, hiszen azt tapasztaltuk, hogy az eljárásunk így is megfelelő hatékonysággal működik.

Mivel az EXPAND eljárásban egyre nagyobb mértékű nem-megengedettséget fogadunk el, ezért gyakoriak a negatív hányadosok, de azok abszolút értékben nagyon kicsik, valamint

az algoritmus biztosítja a θ_{min} mértékű minimális duál előrehaladást. Ebben az esetben nincs tehát szükség a hamis megengedett változók szűrésére sem. Összességében ez az eljárás hatékonyan orvosolja a hamis megengedettséget, valamint biztosítja a megfelelő nagyságrendű pivotelemet mindamellett, hogy a második fázisú degenerációt is hatékonyan kezel. Ez a módszer tehát kiemelkedő jelentőségű a magas szintű megvalósítás szempontjából.

3.2.4. Célfüggvény együtthatók perturbálása

A célfüggvény együtthatók perturbálására megvalósítottuk a 2.4.4 alfejezetben ismertetett eljárást, munkánk során ennek vizsgálatára, továbbfejlesztési lehetőségeire fókuszálunk. Ez a gyakorlatban azt jelenti, hogy a feladat modelljében lévő c vektort a megoldási folyamat elején egyszer módosítjuk, majd a módosított feladat optimális megoldás megtalálása után visszaállítjuk az eredeti célfüggvényt, és a kapott bázisról megkeressük az eredeti feladat optimális megoldását. Egy másik módszer, hogy amennyiben degenerációt detektálunk, akkor lokálisan, csak a degenerációt okozó célfüggvény együtthatókat toljuk el, ezen eljárást a szakirodalom shifting-nek (eltolásnak) nevezi. Ennek megvalósítása, vizsgálata további terveink részét képezi.

Az ismertetett néglépéses perturbálás első két lépését összehasonlítottuk néhány saját, egyszerűbb perturbálási logikával, a következőkben tekintsük át ezeket.

- A legegyszerűbb megvalósításban véletlenszerűen generáltunk értékeket egy előre megadott konstans ($\epsilon_{turb} = 10^{-6}$) által meghatározott intervallumból, ennek értelmében $\delta_j \in [-\epsilon_{turb}, \epsilon_{turb}]$.
- A második módszerben a célfüggvény együtthatók előjelének megfelelően választottuk a perturbálás irányát (0 értéknél véletlenszerűen választott előjellel) ez alapján:

$$\delta_j \in \begin{cases} -\left[\frac{\epsilon_{turb}}{2}, \epsilon_{turb}\right], & \text{ha } c_j < 0 \\ \pm\left[\frac{\epsilon_{turb}}{2}, \epsilon_{turb}\right], & \text{ha } c_j = 0. \\ +\left[\frac{\epsilon_{turb}}{2}, \epsilon_{turb}\right], & \text{ha } c_j > 0 \end{cases} \quad (21)$$

- Az általunk megvalósított harmadik perturbálási eljárásban az elsődleges szempont az, hogy a redukált költségeket megengedett irányba toljuk el. Így tehát az egyes változótipusok szerint definiáltuk a perturbálás irányát (**4. táblázat**):

4. táblázat: Perturbálás megengedett irányba

$type(x_j)$	d_j megengedettsége	perturbálási tényező
0	d_j tetszőleges	–
1	$d_j \leq 0$ vagy $d_j \geq 0$	$\delta_j \in \pm \left[\frac{\epsilon_{turb}}{2}, \epsilon_{turb}\right]$
2	$d_j \geq 0$	$\delta_j \in + \left[\frac{\epsilon_{turb}}{2}, \epsilon_{turb}\right]$

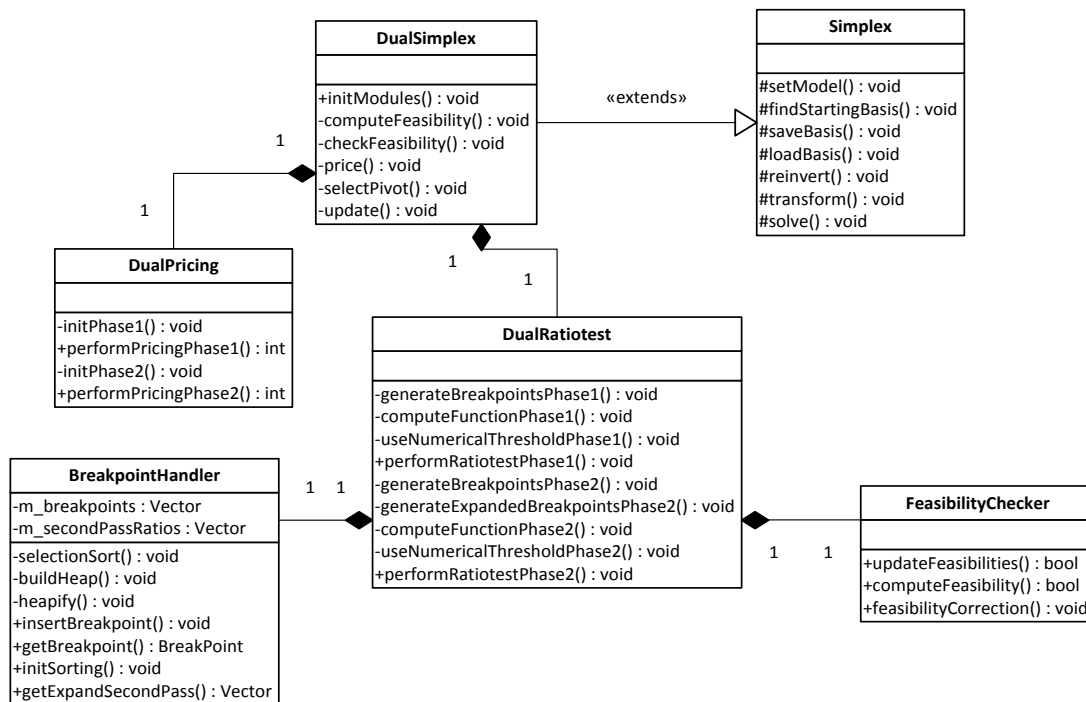
3	$d_j = 0$	–
4	$d_j \leq 0$	$\delta_j \in -\left[\frac{\epsilon_{turb}}{2}, \epsilon_{turb}\right]$

A különböző perturbálási módszerek összehasonlítását a következő fejezetben részletezzük, bemutatjuk mérési eredményeinket.

3.3. Az implementáció struktúrája

Ebben az alfejezetben ismertetjük az implementált osztályokat, ezek kapcsolatát, a hányadoseszték felépítését. Az implementáció C++ programozási nyelven, a C++11 szabvány szerint készült. Egy adott művelet elvégzéséhez a szükséges adatokat kétféleképpen biztosíthatjuk, vagy egy nyílt hozzáférésű adattaggal (osztálysintű adattag, friend osztály), vagy a függvénynek átadott paraméterrel. Ha egy adattag az osztályban nem módosul, akkor konstans referenciaként, ha módosulhat, akkor pedig pointeren keresztül kerül tárolásra. A függvények argumentumait azok nagy száma miatt a későbbiekben nem részletezzük.

A program struktúrájának részleteit az alábbi osztálydiagramon (5. ábra) láthatjuk. Szeretnénk hangsúlyozni, hogy az ábra elsődleges fókuszja a duál hányadoseszték, illetve az ahhoz szorosan kapcsolódó osztályok, ezért nem tér ki a PanOpt minden részletére, csak a számunkra fontos elemeket jeleníti meg. Láthatjuk a megoldás vezérlését végző *DualSimplex* osztályt, mely a *Simplex* absztrakt osztály leszármazottja, illetve a pivot választáshoz kapcsolódó két legfontosabb osztályt, a *DualPricing*ot, illetve a *DualRatiotest*et.



5. ábra: A hányadosesztékhez kapcsolódó osztályok

A következőkben tekintsük át az ábrán a hányadosesztékhez kapcsolódó *FeasibilityChecker*, *BreakpointHandler* osztályokat, majd magát a *DualRatiotest* osztályt.

3.3.1. A FeasibilityChecker osztály

Ez az osztály az egyes redukált költségek megengedettségét vizsgálja, és azok változását kezeli a már ismertett ϵ_{work} toleranciával. A megengedettségi halmazokat partíciókra osztott kétszeresen láncolt lista formájában valósítottuk meg, így az állapotok kezelése egyszerű pointerműveletekkel valósul meg. Ez rendkívül gyors, konstans bonyolultságú. Mindehhez az alábbi függvények nyújtanak segítséget:

- *computeFeasibility()*: A redukált költségek előjele alapján az aktuális toleranciát alkalmazva kiszámolja, hogy az egyes megengedettségi halmazokba mely változók kerülnek, ez tehát egy láncolt lista felépítését jelenti. A függvény minden invertáláskor meghívásra kerül a *DualSimplex* osztályban, a két invertálás közötti változásokat a következő függvény kezeli. A függvény visszatérési értéke igaz, ha megengedett megoldást kaptunk. Mindez egy egyszerű iterátor-összehasonlítás formájában valósul meg, vizsgáljuk, hogy $\mathcal{M} = \mathcal{P} = \emptyset$, hiszen ekkor minden d_j megengedett.
- *updateFeasibilities()*: Ennek a függvénynek a segítségével nem kell újraszámolni minden iterációban a megengedettségi halmazokat. Létrehoztunk egy segédvektort, amellyel a hányadosesztben rögzítjük, hogy a bejövő változó meghatározása után, mely redukált költség melyik halmazba mozgott át, ez a felhasznált töréspontokból egyértelműen adódik. Az egyes redukált költségek aktuális állapota és a segédvektorban tárolt értékek alapján a függvény elvégzi a megengedettségek megváltozásának adminisztrálását. Visszatérési értéke az előző függvénnyel megegyező.
- *feasibilityCorrection()*: Az első fázis végén elvégzi a megengedettségi korrekciókat az egyes típusú változókon, ezáltal a korlátos változók megengedettségét biztosítja. Emellett a függvény második fázisban minden invertálásnál ellenőrzi, hogy megmaradt-e a korlátos változók megengedettsége.

3.3.2. A BreakpointHandler osztály

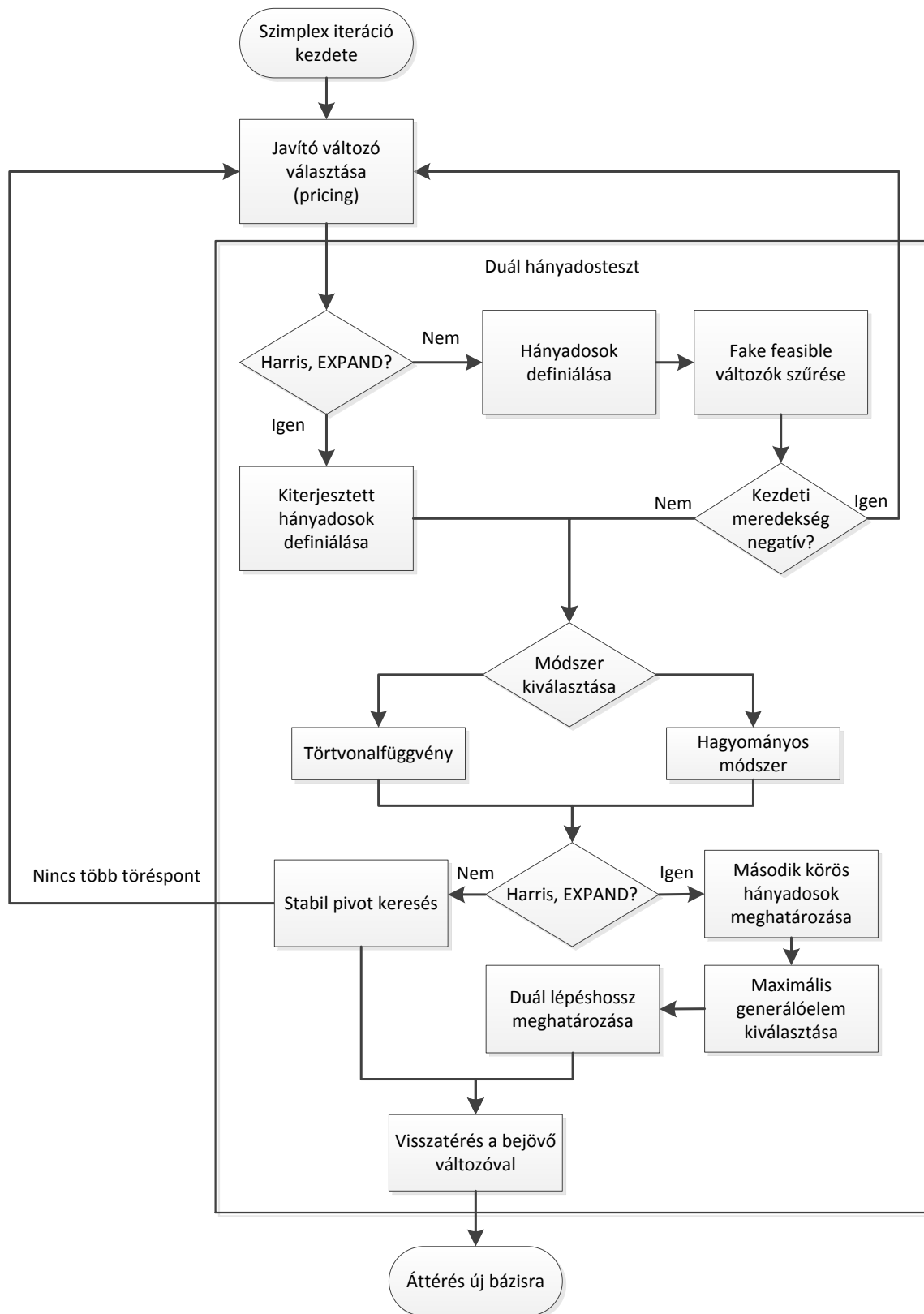
Ez a hányadoskezelő osztály végzi a definiált hányadosok tárolását és kezelését. Az *insertBreakpoint()* függvény segítségével adhatjuk hozzá az elemeket a tárolóhoz. A hányadosok definiálása után a *DualRatiotest* osztály meghívja a hányadoskezelő *initSorting()* függvényét, amely egy kupac adatszerkezetet épít fel a *buildHeap()* függvény segítségével, hiszen a hányadosokra rendezve van szükség. Az egyes elemek rendezésekor a *heapify()* függvény tartja fenn a kupac tulajdonságot. Ezen metódusok segítségével az osztály lehetőséget ad a hányadosok egységbe zárt kezelésére, amelynek köszönhetően a *DualRatiotest* osztály átlátható és egyszerűen értelmezhető, valamint a hányadoskezelő felhasználható a primál megvalósításban is. Mindemellett tervezzük, hogy több különböző rendezőalgoritmust is kipróbálunk az osztály segítségével, hogy ajánlást tudjunk megfogalmazni arra vonatkozóan, hogy mikor milyen rendezést célszerű használni.

3.3.3. A DualRatiotest osztály

A *DualRatiotest* osztály végzi a hányadosok definiálását, majd a paraméter fájlban megadott módszernek megfelelően a bejövő változó kiválasztását. Szintén ebben az osztályban került megvalósításra a hamis megengedett változók kezelése, stabil pivotálási eljárás, a Harris hányadoseszt, valamint az EXPAND eljárás. Ezen feladatok elvégzésére az alábbi függvényeket implementáltuk:

- *generateBreakpointsPhase1()*, *generateBreakpointsPhase2()*: Ezek a függvények végzik a megfelelő fázisban az egzakt korlátos hányadosok definiálását, a definiált megengedettségi halmazok alapján.
- *generateExpandedBreakpointsPhase2()*: Ez végzi a Harris hányadosesztben, illetve az EXPAND eljárásban alkalmazott kiterjesztett hányadosok definiálását.
- *computeFunctionPhase1()*, *computeFunctionPhase2()*: A törtvonalfüggvények kiszámolását végzik, meghatározzák azok maximumát, illetve a bejövő változót.
- *useNumericalThresholdPhase1()*, *useNumericalThresholdPhase2()*: Az ismertetet stabil pivotálási eljárásokat végzik a megfelelő fázisra.

A *DualRatiotest* osztály működését az alábbi ábrán (6. ábra) láthatjuk:



6. ábra: A kibővített duál hányadoseszt működése

4. Mérési eredmények

Ebben a fejezetben ismertetjük az általunk elvégzett mérések eredményeit, ez képezte munkák lényegi részét, hiszen ezen eredményekkel tudjuk alátámasztani a dolgozatban megfogalmazott állításokat, a megvalósított technikák helyességét. Az első alfejezetben bemutatjuk a tesztelésre választott feladatok jellemzőit, ezt követően pedig ismertetjük a mért eredményeket, és az azokhoz kapcsolódó következtetéseinket.

4.1. Tesztfeladatok

Az elvégzett méréseket a szakirodalomban közismert *LP netlib* feladatcsomagból származó, valós életből vett feladatokon végeztük. [13] A feladatok kiválasztásánál arra törekedtünk, hogy a *netlib* közel 100 példájából lehetőség szerint a nagyobb méretűeket, illetve a numerikusan nehezebb, degenerált feladatokat válasszuk ki. Így igazolni tudjuk az előző fejezetben ismertetett numerikus kihívások jelentőségét, illetve az általunk megvalósított technikák hatékony és helyes működését. Továbbá szeretnénk hangsúlyozni, hogy a jelenlegi duál implementációnk az összes *netlib* feladatot képes megoldani. Tekintsük át a kiválasztott *netlib* feladatokat (5. táblázat):

5. táblázat: Netlib tesztfeladatok

Feladat neve	Sorok száma	Oszlopok száma	Nem-nulla elemek száma	Ritkassági arány
25FV47	822	1571	11127	0,8616%
80BAU3B	2263	9799	29063	0,1311%
BNL1	644	1175	6129	0,8100%
BOEING1	351	384	3865	2,8676%
BOEING2	167	143	1339	5,6070%
D2Q06C	2172	5167	35674	0,3179%
DEGEN3	1504	1818	26230	0,9593%
FIT2D	26	10500	138018	50,5560%
FIT2P	3001	13525	60784	0,1498%
GREENBEA	2393	5405	31499	0,2435%
GREENBEB	2393	5405	31499	0,2435%
GROW15	301	645	5665	2,9179%
GROW22	441	946	8318	1,9938%
MAROS-R7	3137	9408	151120	0,5120%
MODSZK1	688	1620	4158	0,3731%
NESM	663	2923	13988	0,7218%
PEROLD	626	1376	6026	0,6997%
PILOT	1442	3652	43220	0,8207%
PILOT_JA	941	1988	14706	0,7861%
PILOT4	411	1000	5145	1,2518%
SCAGR25	130	140	553	3,0385%
STOCFOR2	2158	2031	9492	0,2166%
STOCFOR3	16676	15695	74004	0,0283%

Feladat neve	Sorok száma	Oszlopok száma	Nem-nulla elemek száma	Ritkassági arány
TRUSS	1001	8806	36642	0,4157%
VTP-BASE	199	203	552	1,3664%
WOOD1P	245	2594	70216	11,0484%
WOODW	1099	8405	37478	0,4057%

Mivel ezek a feladatok valós életből vett problémák, amelyek sokféle különböző kihívást rejtenek magukban, ezáltal egy LP megoldó szoftver általános tesztelésére kiválóan alkalmasak. Jellemző, hogy míg néhány példa megoldásához elegendő az egyszerű szakirodalmi leírások megvalósítása, addig más, kevésbé jóindulatú feladatoknál egészen fejlett technikák szükségesek a helyes megoldáshoz.

A különböző futtatások kezeléséhez hatékonyan tudtuk kihasználni a PanOpt paraméterfájlját, amelyben kényelmesen lehet változtatni a megoldó paramétereit, így a különböző módszerek és algoritmusok változtatása a mérések elvégzése során egyszerű volt. Az előző fejezetben ismertetett toleranciák alapértelmezett értékeit használtuk, ezek a következők (**6 táblázat**):

6. táblázat: Toleranciák alapértelmezett értékei

ϵ_{work}	ϵ_{turb}	ϵ_{pivot}	ϵ_{feas}	ϵ_{opt}	ϵ_{rel}	ϵ_{abs}
$\left[\frac{\epsilon_{opt}}{100}, \epsilon_{opt} \right]$	10^{-6}	10^{-6}	10^{-8}	10^{-8}	10^{-10}	10^{-14}

4.2. Elvégzett mérések

Ebben az alfejezetben ismertetjük a méréseink eredményeit, a következő elv alapján: egy adott problémát, jelenséget néhány szemléletes példa segítségével mutatunk be, táblázatos formában. Méréseinket három csoportba soroltuk a következő problémás szituációk alapján: hamis megengedett hányadosok kezelése, numerikusan jó generálóelem választása, és a degeneráció elhárítása. Az egyes funkciók tesztelésénél mindig csak az adott paramétert változtattuk, hogy a levont következtetések helyesek legyenek. Elöljáróban még elmondanánk, hogy a *netlib* feladatokra jellemző, hogy a duál megoldót alkalmazva nagyon gyorsan átkerülnek második fázisba, így az iterációk nagy része (~80%) itt zajlik, ennek megfelelően kevésbé szignifikánsak az első fázisú jelenségek. Méréseink során a megoldóban már rendelkezésre álló technikák közül nem alkalmaztunk sem skálázást, sem előfeldolgozást. A megoldás során a bázis szorzat alapú inverzét (PFI), Dantzig pricing eljárást és alsó logikai kiindulási bázist használtunk.

4.2.1. Hamis megengedett hányadosok kezelése

Tekintsük az alábbi néhány feladatot, melyekkel reprezentatívan bemutathatjuk a hamis megengedettségek kezelésének fontosságát. (**7. táblázat:** Példák a fake feasibility jelenségre)

25FV47: Ez a feladat rossz numerikus tulajdonságai miatt kiváló lehetőséget nyújt a jelenség megfigyelésére. Láthatjuk, hogy azon futtatás során, ahol nem volt bekapcsolva a fake feasible változók kezelése, ott az algoritmus rossz célfüggvényértéken állt meg, jelentős volt a degeneráció és rontó iterációk is történtek.

BNL1: Ezen feladat azt az esetet reprezentálja, amikor a fake feasible változók nem kezelése azt okozza, hogy az algoritmus nem tudja megoldani a feladatot. A degeneráció számának megnövekedése az iterációszám növekedésével magyarázható, hiszen az első esetben 2656 iteráció után leállt a megoldó.

D2Q06C: A feladatban jól megfigyelhetők a hamis megengedettség okozta károk. A célfüggvény értékének eltérése nem szignifikáns, hiszen az optimális érték 10^5 nagyságrendű.

7. táblázat: Példák a fake feasibility jelenségre

Feladat	Iterációszám	Futási idő (mp)	Célfüggvény eltérése ¹	Rontó iterációk száma	Degenerált iterációk száma	Fake feasible aktiválódás ²
BNL1	2656	0,92	∞	0	866	nincs
	3514	1,05	0	0	1264	29 (61)
25FV47	14577	12,01	2,4%	16	4635	nincs
	10814	9,03	0	2	2776	42 (103)
D2Q06C	47096	230,41	0	474	4344	nincs
	26457	103,62	< 0,01%	37	3026	6983 (88860)

4.2.2. Numerikusan stabil pivotelem biztosítása

Mint azt korábban ismertettük a numerikusan rossz tulajdonságú generálóelem kiválasztása a bázis numerikus szingularitásához, illetve nullával való osztáshoz vezethet, amely problémák az algoritmus szempontjából kritikusak, kezelésük nem megkerülhető. Tekintsünk a továbbiakban a bemutatott technikák eredményeit.

Első fázisú törtvonalfüggvény

Az alábbi 6 feladatban jól megfigyelhetjük az első fázisú törtvonalfüggvény hatékonyságát a hagyományos egylépéses módszerrel szemben (**8. táblázat:** Példák az első fázisú törtvonalfüggvény hatékonyságára). Ez mind az iterációszám, mind a megoldási idő csökkenésében megmutatkozik, valamint a *PILOT4* esetében a hagyományos módszer nem is képes megoldani a feladatot.

¹ Az algoritmus által optimálisnak elfogadott és a *netliben* ismertett optimális célfüggvény-érték eltérése.

² Ha aktív a fake feasible szűrés, akkor az első szám mutatja a megoldás során történt szűrések számát, a zárójelben lévő szám pedig az összes fake feasible változó számát.

8. táblázat: Példák az első fázisú törtvonalfüggvény hatékonyságára

Feladat	Iterációszám	Futási idő (mp)	Első fázisú módszer
BOEING1	935	0,18	egy lépéses
	848	0,13	törtvonalfüggvény
BOEING2	532	0,09	egy lépéses
	240	0,03	törtvonalfüggvény
DEGEN3	9323	15,7	egy lépéses
	8296	13,91	törtvonalfüggvény
NESM	3276	1,35	egy lépéses
	2953	1,07	törtvonalfüggvény
PILOT4	-	-	egy lépéses
	4301	3,82	törtvonalfüggvény
SCAGR25	685	0,09	egy lépéses
	607	0,09	törtvonalfüggvény

Második fázisú törtvonalfüggvény

A második fázisú törtvonalfüggvényes hányados-meghatározás, mint már említettük, csak abban az esetben előnyös, ha a feladatban vannak egyes típusú, azaz korlátos változók, viszont ilyenkor nagyon hatékony, mint azt a következő táblázatban (9. táblázat: Példák a második fázisú törtvonalfüggvény hatékonyságára) láthatjuk.

9. táblázat: Példák a második fázisú törtvonalfüggvény hatékonyságára

Feladat	Iterációszám	Futási idő (mp)	Második fázisú módszer
80BAU3B	6574	7,32	egylépéses
	4235	3,53	törtvonalfüggvény
BOEING1	935	0,21	egylépéses
	650	0,11	törtvonalfüggvény
BOEING2	260	0,04	egylépéses
	201	0,02	törtvonalfüggvény
FIT2D	7674	15,11	egylépéses
	187	0,74	törtvonalfüggvény
FIT2P	14713	26,13	egylépéses
	6655	17,01	törtvonalfüggvény
GREENBEB	22474	25,29	egylépéses
	17640	18,51	törtvonalfüggvény
NESM	5171	2,31	egylépéses
	3783	1,98	törtvonalfüggvény
PILOT4	4301	3,82	egylépéses
	2012	3,21	törtvonalfüggvény
VTP-BASE	323	0,03	egylépéses
	257	0,03	törtvonalfüggvény

A két törtvonalfüggvény alkalmazása azonban rosszindulatú feladatok esetén nem elegendő, külön figyelmet kell fordítani a generálóelem kiválasztására, annak numerikus tulajdonságai alapján.

Numerikusan rossz pivotelem választásának hatása

Vizsgáljuk meg mi történik, ha rossz pivotelemet választunk a hányados-kiválasztás során, erre tekintsük az alábbi példákat: (10. táblázat:)

10. táblázat: Pivotálás numerikusan rossz generálóelemen

Feladat	Iterációszám	Futási idő (mp)	Első fázisba visszaesés	Numerikusan szinguláris bázis	Kritikus leállás	Pivotelem vizsgálata
25FV47	23883	21,38	6	3	Nem	Nem
	9935	5,72	0	0	Nem	Igen
BNL1	3685	0,79	3	3	Igen	Nem
	3265	0,65	0	0	Nem	Igen
PILOT_JA	5972	5,36	0	12	Igen	Nem
	25000	-	0	0	Nem	Igen

Láthatjuk, hogy ezen probléma a megoldáshoz szükséges iterációszám lényeges megnövekedéséhez (25FV47), numerikusan nehezebb feladatoknál akár az algoritmus kritikus leállításához (BNL1) vezethet. Ennek kezelését végezhetjük a hányadosteszthez képes magasabb szinten, ezért célszerű a kiválasztott generálóelem vizsgálatát a hányadosteszt végén elvégezni, és amennyiben $|\alpha_j^p| < \epsilon_{pivot}$ teljesül, a vezérlőnek jelzést küldeni. Ennek egy lehetséges kezelése a pricing által kijelölt javító sor tiltása, és egy másik javító sor kijelölése. Ezen heurisztika alkalmazása bizonyos esetekben hatékonynak tűnhet, azonban nem veszélytelen. Előfordulhat, hogy nagyszámú sor kerül letiltásra, ami szélsőséges esetben az optimum megtalálásának kizárását jelentheti, ezért érdemes alternatív megoldásokat keresni erre a problémára. Az eredményekből nyilvánvalóan látszik, hogy a numerikusan rossz tulajdonságú, rosszul kondicionált feladatoknál bár a kritikus leállást megakadályozta, az optimális megoldás megtalálásához nem volt elegendő, ciklizálást eredményezett (PILOT_JA).

Ahhoz, hogy a lehetséges generálóelemek tekintetében nagyobb szabadságunk legyen mindenképpen érdemes a törtvonalfüggvényes módszert alkalmazni a hagyományossal szemben, azonban azt ki kell egészíteni az ismertetett stabil pivotálási eljárással.

Stabil pivotálási eljárás

A stabil hányados-meghatározás a duál algoritmus mindkét fázisában megvalósítható, célszerű tehát erre alapozni a numerikusan jó generálóelem biztosításának kérdését is. Az alábbi feladatokban (11. táblázat) megfigyelhetjük, hogy hogyan járult hozzá az iterációszám, a megoldási idő, valamint néhány esetben a degenerált iterációk számának jelentős csökkenéséhez a megvalósított stabil pivotálási eljárás. Néhány esetben (D2Q06C, GREENBEA, MA-ROS-R7) az algoritmus nélkül nem sikerült megoldani a feladatot.

11. táblázat: Példák a stabil pivotálási eljárás hatékonyságára

Feladat	Iterációszám	Futási idő (mp)	Degenerált iterációk száma	Stabil pivotálás
BOEING2	189	0,019	90	Nincs
	174	0,017	78	Van
D2Q06C	-	-	-	Nincs
	26457	103,62	37	Van
DEGEN3	8296	17,51	623	Nincs
	8007	16,12	683	Van
MODSZK1	1117	0,17	1048	Nincs
	698	0,13	639	Van
GREENBEA	-	-	-	Nincs
	9150	9,01	2969	Van
GREENBEB	20589	22,81	13062	Nincs
	13330	12,99	5496	Van
MAROS-R7	-	-	-	Nincs
	4608	93,19	746	van
VTP-BASE	257	0,02	47	nincs
	231	0,02	56	van
WOOD1P	2231	10,37	2209	nincs
	328	1,02	318	van
WOODW	48862	52,72	48114	nincs
	2167	3,44	1858	van

A futási eredmények alapján láthatjuk, hogy a törtvonalfüggvényes módszer valóban minden esetben hatékonyabb a hagyományos eljárásnál, legrosszabb esetben a működésük hasonló eredményt szolgáltat. Láthatjuk továbbá azt is, hogy a numerikusan rossz pivotelemből származó problémákat az általunk megvalósított stabil pivotálási eljárás hatékonyan kezeli, így a törtvonalfüggvényes módszert célszerű ezzel kiegészítve alkalmazni. Bizonyos feladatoknál láthatjuk, hogy a stabil pivotálási eljárás ellenére sem sikerült numerikusan jó generálemet kiválasztani, ilyenkor továbbra is szükség van a vezérlés bevonására a pivotválasztásba.

4.2.3. Degenerált feladatok kezelése

Az alábbi táblázatban (12. táblázat) azt láthatjuk, hogy az eddig bemutatott eljárások szinte mindegyike képes valamilyen mértékben csökkenteni a degenerált iterációk számát, ennek köszönhetően általában hatékonyabb megoldást biztosítanak.

12. táblázat: Példák a degeneráció jelenségére

Feladat	Iterációs szám	Futási idő (mp)	Degenerált iterációk száma	Módszer
STOCFOR2	2183	0,91	883	egy lépéses
	2383	1,09	481	törtvonalfüggvény (első fázis)
STOCFOR3	17373	75,9	6387	egy lépéses
	19150	91,1	3361	törtvonalfüggvény (első fázis)
80BAU3B	6574	7,3	1421	egy lépéses
	6663	7,32	1382	törtvonalfüggvény (első fázis)
	4235	3,53	1189	törtvonalfüggvény (második fázis)
FIT2D	7674	15,11	553	egy lépéses
	187	0,74	4	törtvonalfüggvény (második fázis)
FIT2P	14713	26,13	1535	egy lépéses
	6655	17,01	89	törtvonalfüggvény (második fázis)
VTP-BASE	323	0,02	98	egy lépéses
	257	0,02	47	törtvonalfüggvény (második fázis)
NESM	2951	1,07	2091	fake feasible inaktív
	2769	1,01	1828	fake feasible aktív
MAROS-R7	4776	97,73	822	fake feasible inaktív
	4608	93,19	746	fake feasible aktív
D2Q06C	47096	230,41	4344	fake feasible inaktív
	26457	103,62	3026	fake feasible aktív
GREENBEB	18656	19,66	11521	törtvonalfüggvény (mindkét fázis)
	13330	12,99	5496	stabil pivotálás
MODSZK1	1117	0,17	1048	törtvonalfüggvény (mindkét fázis)
	698	0,13	639	stabil pivotálás
TRUSS	7714	13,19	1463	törtvonalfüggvény (mindkét fázis)
	7896	15,33	1333	stabil pivotálás
WOOD1P	2231	10,37	2209	törtvonalfüggvény (mindkét fázis)
	328	1,02	318	stabil pivotálás
WOODW	48862	52,72	48114	törtvonalfüggvény (mindkét fázis)
	2167	3,44	1858	stabil pivotálás

Összefoglalva láthatjuk tehát, hogy bár a degenerált iterációk száma az egyszerű megvalósításhoz képest jelentősen csökkent, azok száma még mindig igen jelentős. A továbbiakban tehát a bemutatott Harris hányadosesztet, az EXPAND eljárást és a perturbációs módszerek hatását vizsgáljuk. A továbbiakban az eddigi legjobb mért eredményeket használjuk referenciaként a megvalósítás hatékonyságának vizsgálatára.

A Harris hányadoseszt és az EXPAND eljárás

Az általunk megvalósított duál második fázisú Harris hányadoseszt és EXPAND eljárások működésére tekintünk az alábbi táblázatot (**13. táblázat:** Hatékony degeneráció-ellenes stratégiák).

13. táblázat: Hatékony degeneráció-ellenes stratégiák

Feladat	Iterációs szám	Futási idő (mp)	Degenerált iterációk száma	Módszer
STOCFOR2	2383	1,09	481	az eddigi legjobb megvalósítás
	2118	0,73	273	Harris hányadoseszt
	2118	0,58	0	EXPAND eljárás
STOCFOR3	19150	91,1	3361	az eddigi legjobb megvalósítás
	16145	56,05	1974	Harris hányadoseszt
	16145	47,50	1948	EXPAND eljárás
80BAU3B	4235	3,53	1189	az eddigi legjobb megvalósítás
	4205	1,92	1121	Harris hányadoseszt
	4191	1,55	69	EXPAND eljárás
FIT2D	187	0,74	4	az eddigi legjobb megvalósítás
	215	0,37	7	Harris hányadoseszt
	191	0,27	0	EXPAND eljárás
FIT2P	6655	17,01	89	az eddigi legjobb megvalósítás
	6769	18,72	87	Harris hányadoseszt
	6818	15,35	0	EXPAND eljárás
VTP-BASE	257	0,02	47	az eddigi legjobb megvalósítás
	236	0,01	50	Harris hányadoseszt
	222	0,01	0	EXPAND eljárás
NESM	2769	1,01	1828	az eddigi legjobb megvalósítás
	2559	0,59	1601	Harris hányadoseszt
	2903	0,62	0	EXPAND eljárás
MAROS-R7	4608	93,19	746	az eddigi legjobb megvalósítás
	4630	123,46	891	Harris hányadoseszt
	4630	100,41	0	EXPAND eljárás
D2Q06C	26457	103,62	3026	az eddigi legjobb megvalósítás
	29875	75,69	3150	Harris hányadoseszt
	29536	61,74	167	EXPAND eljárás
GREENBEB	13330	12,99	5496	az eddigi legjobb megvalósítás
	11665	12,00	5211	Harris hányadoseszt
	12933	11,66	265	EXPAND eljárás
MODSZK1	698	0,13	639	az eddigi legjobb megvalósítás
	723	0,09	655	Harris hányadoseszt
	721	0,07	0	EXPAND eljárás
TRUSS	7896	15,33	1333	az eddigi legjobb megvalósítás
	7509	10,79	1643	Harris hányadoseszt
	7412	7,9	0	EXPAND eljárás
WOOD1P	328	1,02	318	az eddigi legjobb megvalósítás
	487	0,46	484	Harris hányadoseszt
	560	0,54	0	EXPAND eljárás
WOODW	2167	3,44	1858	az eddigi legjobb megvalósítás
	2358	1,29	1997	Harris hányadoseszt
	2513	1,41	0	EXPAND eljárás

Láthatjuk, hogy a legtöbb esetben az eddigi leghatékonyabb megvalósítással szemben a Harris hányadoseszt, illetve az EXPAND eljárás bár hatékonyabbnak bizonyultak, de a két módszer ötvözése a jövőben egy új, hatékony eljáráshoz vezethet. Az EXPAND futtatása során azon iterációkat, amelyben a θ_{min} érték került kiválasztásra, degenerált iterációként mérjük, hiszen a gyakorlatban ezek így értelmezhetők, bár elméleti szinten a 0 hányadosok kizárása a degeneráció megszűnését jelenti. Néhány feladat esetében elmondhatjuk, hogy az EXPAND megvalósítása az, ami az adott feladat megoldhatósága szempontjából kritikus volt, ezek megoldásához a korábbi módszerek önmagukban kevésnek bizonyultak. (14. táblázat: EXPAND nélkül nem megoldható feladatok)

14. táblázat: EXPAND nélkül nem megoldható feladatok

Feladat	Iterációszám	Futási idő (mp)	Degenerált iterációk száma
GROW15	3210	1,17	1879
GROW22	6022	3,97	4000
PEROLD	3854	3,27	1242

Célfüggvény együtthatók perturbálása

A megvalósított perturbálási eljárásokat az alábbi táblázatban láthatjuk (15. táblázat: Perturbálási eljárások).

15. táblázat: Perturbálási eljárások

Feladat	Iterációszám	Futási idő (mp)	Degenerált iterációk száma	Perturbálási módszer
25FV47	9497	5,27	1744	Nincs
	8957	5,42	1131	véletlenszerű
	10475	6,47	1828	c_j előjele alapján
	10467	6,13	234	megengedett irányba
	9147	5,32	158	4 lépéses perturbálás
80BAU3B	4191	1,52	1110	Nincs
	4427	1,79	786	véletlenszerű
	4377	1,74	766	c_j előjele alapján
	4195	1,55	11	megengedett irányba
	4092	1,52	8	4 lépéses perturbálás
GREEBEA	10048	7,77	2889	Nincs
	13415	11,65	1871	véletlenszerű
	13051	11,12	1594	c_j előjele alapján
	11493	9,18	145	megengedett irányba
	10541	8,461	175	4 lépéses perturbálás
GROW15	3210	1,17	1879	Nincs
	3194	1,26	4	véletlenszerű
	3052	1,29	3	c_j előjele alapján
	2497	0,92	4	megengedett irányba

Feladat	Iterációs szám	Futási idő (mp)	Degenerált iterációk száma	Perturbálási módszer
	5053	1,74	10	4 lépéses perturbálás
PEROLD	3854	3,18	1242	Nincs
	5211	4,91	1	véletlenszerű
	4643	4,04	7	c_j előjele alapján
	3440	2,72	109	megengedett irányba
	3655	2,9	16	4 lépéses perturbálás
PILOT	8794	93,18	1608	nincs
	10023	99,64	66	véletlenszerű
	11101	93,73	58	c_j előjele alapján
	7655	87,15	36	megengedett irányba
	9142	98,74	69	4 lépéses perturbálás
PILOT4	2292	1,41	653	nincs
	1598	0,9	1	véletlenszerű
	2105	1,23	0	c_j előjele alapján
	1249	0,54	83	megengedett irányba
	1350	0,64	1	4 lépéses perturbálás
STOCFOR2	2118	0,58	273	nincs
	2219	0,65	0	véletlenszerű
	2240	0,65	0	c_j előjele alapján
	2287	0,66	0	megengedett irányba
	2287	0,66	0	4 lépéses perturbálás

A táblázat alapján a szakirodalomban ismertetett 4 lépéses perturbálási eljárás jónak bizonyult, viszont az egyértelműen látszik, hogy több feladatnál a saját perturbálási logikánk hatékonyabban szerepelt. Ennek értelmében a továbbiakban szeretnénk egy hatékony perturbálási stratégia-kombinációt meghatározni. A táblázatból szintén láthatjuk, hogy nem minden esetben éri meg perturbálást végezni, ez az eljárás heurisztikus mivoltából adódik. A degenerált iterációk számával kapcsolatban szeretnénk hozzáfűzni, hogy az EXPAND eljárásban ismertett gyakorlati degenerációk detektálására a perturbáció keretében nem volt lehetőségünk, hiszen a megváltozás értéke véletlenszerű, így a degenerált iterációkat nagyrészt apró lépések váltották fel. Összefoglalva a célfüggvény együtthatók perturbálása hatékony degeneráció-ellenes stratégiának bizonyul, ez legfrissebb kutatásaink eredménye.

Összegzőképpen elmondhatjuk, hogy a megvalósított technikák helyes és hatékony működését, az általunk megfigyelt hamis megengedettség jelenlétét, az erre kidolgozott algoritmust, a törtvonalfüggvényes hányados-meghatározást, a stabil pivotálási eljárásokat, a degeneráció ellen kidolgozott hányados-választási módszereket, valamint a perturbálást mérési eredményekkel validáltuk. További terveink közt szerepel a stabil pivotálás és az EXPAND eljárás kombinációjának elméleti és gyakorlati kidolgozása, valamint a perturbációs módszerek egyesítése annak érdekében, hogy még jobb eredményeket érjünk el.

5. Összefoglalás

Kutatómunkám során részt vettem egy fejlett lineáris programozási feladatmegoldó fejlesztésében az Operációkutatási Kutatólaboratórium munkájában. Ezen kutatás célja egy olyan stabil és gyors szimplex alapú megoldó megvalósítása és fejlesztése, amely képes a valós életből vett nagyméretű LP feladatokat hatékonyan megoldani.

Dolgozatunkban áttekintettük a lineáris programozás alapjait, a különböző LP modellek jellemzőit, valamint bemutattuk, hogy mik a terület legfontosabb alapfogalmai és összefüggései. Ezt követően bevezettük a primál-duál feladatpárt, és a duál szimplex algoritmus főbb jellemzőit. Ismertettük és értékeltük a legfontosabb hányados-meghatározási módszereket, valamint tárgyaltuk a degeneráció jelenségét, feltártuk annak okait, és bemutattuk a szakirodalomban fellelhető fontosabb degeneráció-ellenes stratégiákat.

Részletesen bemutattuk a duál hányadoseszték megvalósítását, a degeneráció-ellenes algoritmusokat, valamint ezek jellemzőit. Kitértünk a nagyméretű LP feladatok megoldásához kapcsolódó kihívásokra, illetve arra, hogy ezen problémákat milyen technikák és algoritmusok megvalósításával sikerült hatékonyan kezelni. Tárgyaltuk a lebegőpontos aritmetikából származó numerikus hibák jelentőségét, sort kerítettünk az általunk használt toleranciák ismertetésére. Bemutattuk az általunk megvalósított stabil pivotálási eljárást, valamint a hamis megengedettség jelenségét, melynek kezelésére új módszert dolgoztunk ki. Részletesen ismertettük az általunk kidolgozott duál második fázisú Harris hányadosesztét, az EXPAND eljárás duál adaptációját, valamint a megvalósított perturbálási módszereket. Bemutattuk továbbá a hányadoseszték megvalósításának struktúráját és az ehhez kapcsolódó legfontosabb implementációs jellemzőket.

Végül ismertettük az általunk elvégzett mérési eredményeket, állításainkat több széles körben ismert LP feladat segítségével támasztottuk alá, melyek kiválasztásakor a nagy méret mellett a rossz numerikus tulajdonságú, degenerált feladatokat részesítettük előnyben. Dolgozatunk mellett, hogy a duál szimplex algoritmus hányadosesztetjeinek elméletét részletesen bemutatja, a gyakorlati jelentőségű implementációs kihívásokat, és fejlett degeneráció-ellenes technikákat részletesen tárgyalja.

Munkám további kutatási irányaként a degeneráció elméleti előrejelzésének lehetőségét, illetve a degeneráció-ellenes stratégiák ideális kombinációjának megvalósítását tűztük ki célul. Mindemellett, jövőbeni terveink között szerepel a megvalósított algoritmusok párhuzamosítási lehetőségeinek feltárása. Hosszú távon munkámmal szeretnék hozzájárulni egy hatékony, a mai piaci megoldókkal versenyképes, kutatási célú szoftver fejlesztéséhez. Emellett szeretném tovább mélyíteni ismereteimet a lineáris programozás tudományában, az elméleti alapok elsajátítása mellett a fejlett technikák hatékony megvalósításának vizsgálatával. Végezetül kiemelnénk, hogy a teljes *netlib* feladatcsomag megoldásának potenciálját a PanOpt szempontjából mérőföldkő jelentőségű eredménynek tekintjük.

Irodalomjegyzék

- [1] **W. Orchard-Hays**, Advanced Linear-Programming Computing Techniques, New York: Mc.Graw-Hill, 1968.
- [2] **I. Maros**, Computational Techniques of the Simplex Method, Springer, 2003.
- [3] **A. Koberstein**, „The Dual Simplex Method, Techniques for a fast and stable implementation,” Paderborn, 2005.
- [4] **G. Dantzig, P. Wolfe és A. Orden**, „A generalized simplex method for minimizing a linear form under linear inequality constraints,” *Pacific Journal of Mathematics*, pp. 5(2):183-195, 1955.
- [5] **C. E. Lemke**, „The dual method of solving the linear programming problem,” *Naval Research Logistics Quarterly*, %1. kötet1., pp. 36-47, 1954.
- [6] **G. Dantzig és W. Orchard-Hays**, „The product form of the inverse in the simplex method,” *Mathematical Tables and Other Aids to Computation*, pp. 64-67, 1954.
- [7] **I. Maros**, „A Piecewise Linear Dual Phase-1 Algorithm for the Simplex Method,” *Computational Optimization and Applications*, %1. kötet26, pp. 63-81, 2003.
- [8] **I. Maros**, „A Generalized Dual Phase-2 Simplex Algorithm,” Imperial College, London, 2001.
- [9] **H. Wagner**, „The dual simplex algorithm for bounded variables,” *Naval Research Logistics Quarterly*, %1. kötet5., pp. 257-261, 1958.
- [10] **P. Harris**, „Pivot Selection Method of the Devex LP Code,” *Mathematical Programming*, 1973.
- [11] **M. Gill, P. Murray, W. Saunders és M. Wright**, „A Practical Anti-Cycling Procedure for Linearly Constrained Optimization,” *Mathematical Programming*, pp. 434-474, 1989.
- [12] **T. Gal**, „Degeneracy in mathematical programming and degeneracy graphs,” *ORiON*, pp. 3-36, 1990.
- [13] **D. Gay**, „Electronic mail distribution of linear programming test problems,” *COAL Newsletter*, pp. 10-12, 1985.
- [14] **G. Dantzig**, Linear Programming and Extensions, Princeton: Princeton University Press, 1963.

Köszönetnyilvánítás

Szeretném megragadni az alkalmat, hogy köszönetet mondjak az Operációkutatási Kutatólaboratórium vezetőjének Dr. Maros István professzor úrnak, amiért lehetőséget biztosított számomra, hogy csatlakozzam a kutatócsoporthoz. A témához kapcsolódó kiemelkedő munkásságával és széleskörű ismereteivel nagymértékben segítette a kutatómunkámat, és így a dolgozat létrejöttét. Ezen kívül köszönetet mondanék témavezetőmnek, Tar Péter PhD hallgatónak, aki elméleti téren szintúgy, mint az implementációra vonatkozó kérdésekben folyamatosan támogatta munkámat. Szeretném kiemelni, hogy nagy megtiszteltetésnek érzem, hogy ilyen emberekkel van lehetőségem dolgozni, akiket mind szakmai, mind emberi téren követendő példának tekinthetek.

„A publikáció az Európai Unió, Magyarország és az Európai Szociális Alap társfinanszírozása által biztosított forrásból a TÁMOP-4.2.2.C-11/1/KONV-2012-0004 azonosítójú "Nemzeti kutatóközpont fejlett infokommunikációs technológiák kidolgozására és piaci bevezetésére" című projekt támogatásával jött létre.”