

TDK dolgozat

Stágel Bálint Marcell

A duál szimplex algoritmus hányadostesztjeinek vizsgálata

Szerző: Stágel Bálint Marcell

Szak/évfolyam: Mérnökinformatikus BSc. II. évfolyam

Témavezető: Tar Péter PhD hallgató

Pannon Egyetem, Műszaki Informatikai Kar

Rendszer- és Számítástudományi Tanszék

Operációkutatási Kutatólaboratórium

Absztrakt

Az optimalizálás napjaink egyik gyorsan fejlődő tudományága, munkánk során ezen belül a fejlett lineáris programozási (LP) technikákkal foglalkoztunk. Ezen feladatok megoldására az elmúlt 50 évben kidolgozott két megoldó típus közül mi a báziscserén alapuló szimplex algoritmussal foglalkoztunk.

A módszer két változata a primál, illetve duál szimplex, napjaink korszerű megoldóiban mindkét implementáció magas színvonalon megtalálható. Munkánk során a duál szimplex algoritmus kétfázisú változatának kiemelkedő elemét, a hányadoseszteket vizsgáltuk, amely duál esetben a bázisba belépő változót határozza meg. Ezen hányadosesztekek érdekessége abban rejlik, hogy az egyszerűnek mondható szakirodalmi leírásokat a hatékonyság és a stabilitás érdekében újabb és újabb fejlett technikákkal kiegészítve, egy magas szintű implementációval párosíthatjuk.

Munkánk célja a duál hányadosesztekek megvalósítása és részletes vizsgálata a kutatócsoport által fejlesztett Pannon Optimizer (PanOpt) LP megoldó szoftver segítségével. A duál hányadoseszteket mindkét fázisra implementáltuk, vizsgálódásunk ezek tulajdonságainak feltárására, illetve hatékony továbbfejlesztési lehetőségek kutatására irányult. Az implementált algoritmus numerikus tulajdonságainak vizsgálatát különböző tolerancia értékek segítségével végeztük, amelyek hatásait közismert tesztfeladatokon vizsgáltuk meg. Kitértünk továbbá a példafeladatok degeneráltságának vizsgálatára, bemutatjuk az általunk implementált degeneráció-ellenes stabil pivotálási stratégiákat, majd ezeket értékeljük a tesztfeladatok segítségével. További terveink egyrészt újabb hatékony technikák implementálása és vizsgálata, másrészt a primál hányadosesztekek magas szintű megvalósítása.

Kulcsszavak: Optimalizálás, lineáris programozás, duál szimplex algoritmus

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	5
2.2.2. Második fázis.....	6
2.3. A hányados tesztek szerepe	8
2.3.1. Első fázis	8
2.3.2. Második fázis.....	11
3. A hányados tesztek hatékony megvalósítása	14
3.1. Egy nagyméretű LP megoldó implementációs kihívásai	14
3.2. Megvalósított technikák és algoritmusok	15
3.2.1. A PanOpt által használt matematikai modell	16
3.2.2. Toleranciaértékek használata.....	17
3.2.3. Stabil pivotálási eljárás	18
3.3. Az implementáció struktúrája	20
3.3.1. A FeasibilityChecker osztály.....	21
3.3.2. A DualRatioTestUpdater osztály	22
3.3.3. A DualRatioTest osztály.....	22
3.3.4. A BreakpointHandler osztály	22
4. Mérési eredmények	24
4.1. Tesztfeladatok	24
4.2. Elvégzett mérések	25
5. Összefoglalás.....	30
6. Irodalomjegyzék.....	31
Köszönetnyilvánítás.....	32

1. Bevezetés

Manapság számos eszköz létezik a különböző gazdasági, politikai vagy egyéb döntések támogatására, a XX. században egész tudomány fejlődött ki ezen kérdéskör mélyebb vizsgálataira: 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 ezen 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 munkák során a lineáris programozással (Linear Programming, LP) foglalkoztunk.

Lineáris programozási problémáról akkor beszélünk, ha a matematikai modellben szereplő összefüggések mind lineárisak, de ezen felül még számos egyéb nem lineáris eset megoldásának alapját képezi ez a terület, 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.

Munkák során a duál szimplex algoritmus hányadosesztjeivel foglalkoztunk, ami a báziscsere során a bázisba belépő változót határozza meg. Ezen vizsgálódásainkat az Operációkutatási Kutatólaboratórium által fejlesztett Pannon Optimizer (PanOpt) LP megoldó keretében végeztük. Bátran állíthatjuk, hogy ezen hányadosesztjek implementációja nagy kihívás, amennyiben a célunk egy hatékony, gyors és megbízható LP megoldó kidolgozása. Az egyszerű szakirodalmi leírásokat ennek érdekében különböző fejlett technikákkal kiegészítjük, hogy stabil, megbízható és pontos megoldást tudjunk biztosítani. A legjellemzőbb nehézségek, amikkel szembe kell nézni, a számítógépek által használt lebegőpontos aritmetikából következő pontatlanságok és numerikus instabilitás, valamint a degeneráció, mint a szimplex algoritmus hatékonyságát nagyban megnehezítő jelenség. Dolgozatunkban ezt a kérdéskört vizsgáljuk részletesen. Ismertetjük a duál hányadosesztjek elméleti hátterét, majd kitérünk ezen hányadosesztjek 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 nagyvonalakban 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 a dolgozatban alkalmazott jelöléskonvenciót:

v konstans, vagy érték,

$\mathbf{v}$ vektor,

$\mathbf{M}$ mátrix,

$\mathcal{M}$ halmaz.

2.1. A lineáris programozási feladat

A lineáris programozás egy széles körben alkalmazható technika, amely 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:

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

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

$$l_j \leq x_j \leq u_j, j = 1 \dots n,$$

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]:

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

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 $R \in \mathbb{R}^{m \times (n-m)}$. Az x vektort ennek megfelelően két részre bonthatjuk, így a lineárisan független oszlopok által meghatározott változókat bázisváltozóknak, míg a többit nem-bázis változóknak nevezzük. A változók indexeinek tárolására vezessünk be két halmazt, legyenek a bázisváltozók indexei a $\mathcal{B}$, a nembázis-változóké pedig az $\mathcal{R}$ halmazban. Ezek alapján vezessük be a következő jelöléseket a könnyebb áttekinthetőség érdekében: $A = [B \mid R]$, $x = \begin{bmatrix} x_{\mathcal{B}} \\ x_{\mathcal{R}} \end{bmatrix}$, valamint $c = \begin{bmatrix} c_{\mathcal{B}} \\ c_{\mathcal{R}} \end{bmatrix}$. Egy bázisról azt mondjuk, hogy degenerált, ha szerepel benne olyan bázisváltozó, amely a korlátján van. A későbbiekben részletesen tárgyaljuk a degeneráció kérdéseit.

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 célfüggvényérték a lehető legjobb. A célfüggvényérték megváltozásából levezethetők a redukált költségek: $d_j = c_j - c_{\mathcal{B}}^T B^{-1} a_j$, amelyek a változó egységnyi megváltozásának a célfüggvényre gyakorolt hatását írja le. [2] Ha a változó értéke elmozdul, akkor az elmozdulás a célfüggvényt d_j -nek megfelelően befolyásolja. Ha nincs olyan változó, amely elmozdulása javítana a célfügg-

vényen, akkor optimális megoldást találtunk, az erre vonatkozó feltételek CF-1-es alakot feltételezve:

- alsó korlátos nem-bázis változók esetében $d_j \geq 0$,
- felső korláton lévő nem-bázis változókra $d_j \leq 0$,
- egy szabad változóé $d_j = 0$,
- fix változók redukált költsége bármilyen előjelű lehet, hiszen ha egyszer elhagyta a bázist, utána többször nem kerülhet bele.

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

$type(x_j)$	a változó értéke	optimalitási feltétel
0	$l_j = x_j = u_j = 0$	-
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	$x_j = 0$	$d_j = 0$

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

$$\begin{aligned}
 & \max \mathbf{b}^T \mathbf{y} \\
 & s. t. \mathbf{A}^T \mathbf{y} + \mathbf{w} = \mathbf{c} \\
 & \mathbf{w} \geq \mathbf{0}
 \end{aligned}$$

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. Érdekes, hogy ha a primál-duál feladatpár bármelyikének van egy megengedett megoldása véges optimummal, akkor ugyanez igaz a másik feladatra, és az optimális célfüggvényértékek 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. [3] A továbbiakban a duál változók értékének az így kiszámítható értékeket tekintjük.

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. [3] Ennek megfelelően a primál szimp-

lex 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 primál és a duál 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 megoldó sem dominál a másik felett, mindkettőnek megvannak a maga előnyei.

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 megoldó a redukált költségek értékével és korlátjaival számol. Egy adott d_j -re vonatkozó optimalitási feltétel által meghatározott intervallumot tekintjük a redukált költség megengedettségi tartományának, 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 az első fázisú célfüggvényt [2]:

$$f = \sum_{j \in \mathcal{M}} d_j - \sum_{j \in \mathcal{P}} d_j$$

Mivel az $\mathcal{M}$ halmazban lévő változók redukált költsége negatív, a $\mathcal{P}$ halmazhoz tartozóké pozitív, ezért ez a célfüggvény minden esetben nem-pozitív, értelemszerűen tehát maximalizáljuk.

A szomszédos bázisra való áttérés során meg kell határoznunk a redukált költségek vektorának az új bázisra vonatkozó képét. Ha a generálóelemet a p sorból választottuk, és áttérünk a szomszédos bázisra, akkor a redukált költségek $d_j(t) = d_j - t\alpha_j^p$ szerint változnak, tehát a p sor régi bázisra vonatkozó képének t szeresével változik meg. [2] Vizsgáljuk meg a célfüggvény megváltozását a t elmozdulás függvényében:

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

A zárójelben szereplő kifejezést nevezzük a változó duál első fázisú redukált költségének, és így jelöljük:

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

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 $Z = \mathbf{b}^T \mathbf{y}$, illetve adott egy duál megengedett $\mathbf{B}$ bázis.

Válasszuk ki a k bázisváltozóhoz tartozó egyenlőségként teljesülő feltételt, majd relaxáljuk azt úgy, hogy a célfüggvényérték javuljon, ennek értelmében az $\mathbf{a}_k^T \mathbf{y} = \mathbf{c}_k$ feltétel alakja $\mathbf{a}_k^T \mathbf{y} \geq \mathbf{c}_k$ vagy $\mathbf{a}_k^T \mathbf{y} \leq \mathbf{c}_k$ 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 eltolá-

sának mértéke. Mindezt a duál változók értékeinek az alábbi módon történő megváltozása kompenzálja [2]:

$$B^T y = B^T y(t) - t e_p$$

amit átrendezve megkapjuk a duál változók megváltozását:

$$y(t) = y + t \rho_p,$$

ahol ρ_p az inverz bázis transzponáltjának p sora.

Ez alapján megvizsgálhatjuk a célfüggvény értékének megváltozását t függvényében.

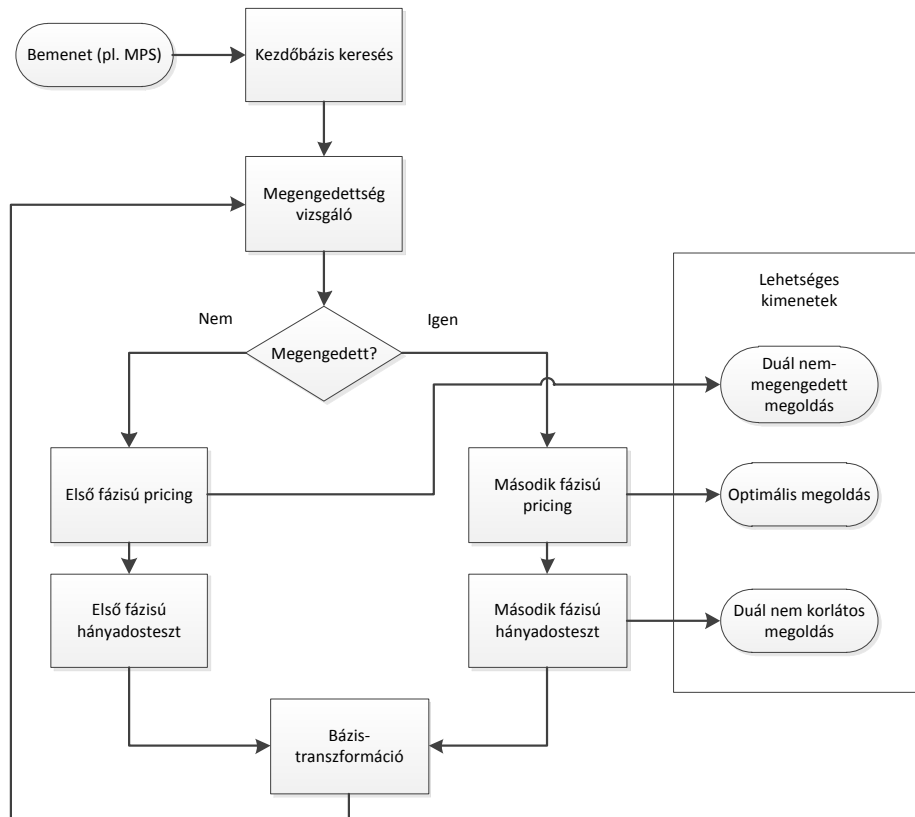
$$Z(t) = b^T y(t) = b^T y + t b^T \rho_p = b^T y + t \beta_p = Z(0) + t \beta_p$$

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

- ha $t < 0$, akkor $\beta_p < 0$,
- ha $t > 0$, akkor pedig $(\beta_p - u_p) > 0$, ahol u_p a bázisváltozó felső korlátját jelenti.

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.

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



1. ábra: A duál szimplex algoritmus működése

A következő alfejezetben bemutatjuk a duál hányadosesztet mindkét fázisra, részletesen ismertetjük azok jellemzőit.

2.3. A hányadosesztek szerepe

Az eddigiekben áttekintettük, hogy a duál szimplex algoritmus során hogyan kapunk javító változót, ehhez milyen feltételeknek kell teljesülnie. A hányadosesztnek az elsődleges be- menete tehát nem más, mint a bázis-transzformáció során a bázist elhagyó változó. Ehhez a hányadoseszt során meghatározunk egy bejövő változót. Ezt követően elvégezhetők a szükséges transzformációk, a báziscsere, és így haladhatunk iteratíván az optimum felé. [6] A báziscseréhez szükség van egy α_j^p generálóelemre (pivotelem), amit a pricing és a hányadoseszt 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. A redukált költségek az alábbiképpen változnak a t elmozdulás függvényében [2]:

$$d_j(t) = d_j - t\alpha_j^p,$$

ahol α_j^p a mátrix p sorának j elemének az adott bázisra vonatkozó képe.

Ennek alapján, ha egy változó duál megengedettségi állapotot váltana, definiál egy hányadost, mert ekkor kerül megengedettségi tartományának szélére. Az előbbi kifejezésből könnyen látható, hogy az a lépés, ahol a j változó hányadost definiál:

$$t_j = \frac{d_j}{\alpha_j^p}, j \in R$$

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

2. táblázat: Hányadost definiáló irányok

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

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 hányadosot.

- 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 hányadosesztben.
- 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 így már megengedett lesz. Ezt az eljárást nevezzük megengedettségi korrekciónak (feasibility correction), amelyet a hányadoseszt 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 valamelyikébe 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 hányadosot 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 hányadosot 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

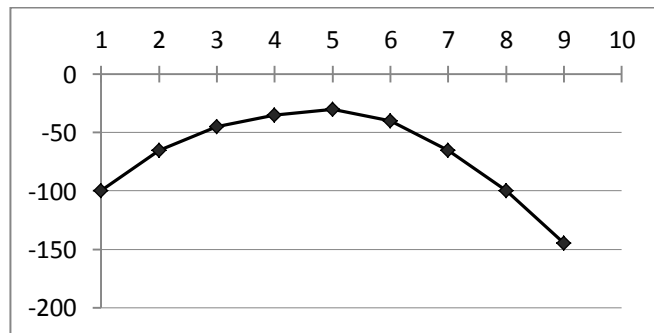
Ez a módszer jóval összetettebb, az adott iterációban elérhető legnagyobb előrehaladást garantálja. Vizsgáljuk meg az első fázisú célfüggvényt az elmozdulás függvényében.

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 célfüggvényérték $f_k = f_{k-1} + s_p^{k-1}(t_k - t_{k-1})$, 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 lévónél, 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.

Tehát 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 célfüggvényértékkel, 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 korlátban 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 korlátban lévő változókét pedig $d_j(t) = d_j - t\alpha_j^p \leq 0$. Ezek alapján 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 korlátban lévő nem-bázis változóra, felső korlátban lévőkre pedig $\alpha_j^p > 0$ mellett szintén $t_j \geq d_j/\alpha_j^p$ adható. Ellenkező alfa 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 korlátban lévőkre pedig $\alpha_j^p < 0$ teljesülése esetén $t_j \leq d_j/\alpha_j^p$. Az alfa 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 változókra vonatkozókat 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 korlátozott változókat. Ha egy ilyen változót választunk ki a hányadosesztben, 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 célfüggvényérték 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 (boundflipet) 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 tehát korlátozott 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 ismételtethetjük, 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 célfüggvényérték 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 boundflipet, valamint egy bejövő változót. Könnyen látható, hogy vagy egy olyan korlátozott változót fogunk behozni a bázisba, aminek megengedettségi tartománya nagyobb, mint a primál lépéshossz, vagy egy plusz típusú változót, hiszen azon túllépve a célfüggvényérték 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 a működése a hagyományossal megegyezik. [9]

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

3. A hányadostesztek hatékony megvalósítása

Ebben a fejezetben részletesen bemutatjuk az általunk megvalósított hányadosteszteket 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 hányadostesztek felépítését, a program struktúráját, a megvalósított osztályokat a kutatócsoport által fejlesztett PanOpt LP megoldó szoftver keretében. Tárgyaljuk a szoftver sajátosságait, 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 számszerűsíthető, ennek megfelelően jól látszik, hogy az adott feladatban mennyire sok a döntési változó (n), illetve a változókra vonatkozó feltétel (m). Általában ezen együtthatók értékeinek tárolására dupla pontosságú lebegőpontos (double) aritmetikát célszerű használni, egy ilyen szám mérete 8 bájt. Így ha egy 100 000 x 100 000-es feladatot tekintünk, akkor a teljes mátrix tárolására körülbelül 74,5 GB memóriát igényelne, ez 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 elemeinek szórása nagy. Általános probléma, ha adott két szám, melyek nagyságrendje lényegesen eltér, akkor önmagukban még pontosak ezek az értékek, de a másik szám nagyságrendjéhez képest már nem. Példa lehet erre, ha egy 10^{10} -es nagyságrendű számot összeadunk egy kicsi 10^{-10} nagyságrendű számmal, akkor az eredmény a nagyobb számmal egyezik meg teljes egészében, ami nyilvánvalóan nem pontos. Egy másik probléma forrása lehet, ha adott egy hosszabb

komplex műveletsor eredménye, és egy fix érték, amit mi a művelet eredményének gondolunk. Mivel nem pontosan ábrázoljuk a számokat, ezért mindig van hiba, amely a különböző műveletek során felhatványozódik. 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.

Emellett hiba forrása 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, vagyis pontatlan számítások eredménye, a helyes értékük 0 lenne, ami helyesen reprezentálva nem kerülne kiválasztásra.

Ezek és ehhez hasonlóak azok az akadályok, amiket egy mai korszerű LP megoldónak különösebb nehézség nélkül kell vennie, 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 eredménye lehet, hogy az algoritmus rossz célfüggvényértéket fogad el optimálisnak, egy feladatot helytelenül nem-korlátosnak vagy nem-megengedettnek nyilvánít, esetleg a bázis numerikusan szingulárisává válik. Előfordulhat, hogy egy adott javító változóhoz nem talál megfelelő hányadost a hányadosteszt, egy iterációról kiderül, hogy valójában ront a célfüggvényértéken, így sérül a megoldási folyamat konvergenciája, az algoritmus visszaeshet első fázisba egy rossz második fázisú iterációnak köszönhetően.

Egy másik érdekes probléma a degeneráció és a ciklizálás kérdésköre. Degenerációnál több iteráción keresztül nem változik a célfüggvényérték, miközben az algoritmus más és más bázisokon halad át. Ciklizálásról akkor beszélünk, ha adott egy k darab bázisból álló sorozat, ami a megoldás során ismétlődik. Ezen jelenségekről biztosat állítani szinte lehetetlen, mert előfordulhat, hogy csak néhány iterációig lesznek jelen, de olyan eset is előállhat, hogy az algoritmus végtelen ciklusba kerül.

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 működésüket. Tárjaljuk továbbá, hogy az előző alfejezetben ismertetett numerikus hibák elkerülésére milyen technikákat dolgoztunk ki.

A kutatócsoport által fejlesztett PanOpt lineáris optimalizáló rendszer egy kutatási célra szánt nagyméretű LP megoldó szoftver. Legfőbb célja, hogy a szimplex algoritmus implementációjához kapcsolódó fejlett technikákat megvalósítsa, ezek vizsgálatára adjon lehetőséget. Mivel valós, nagyméretű feladatok megoldására készül, ezért elvárható, hogy a feladat méretétől függetlenül hatékonyan és megbízhatóan jusson el az optimális megoldásig, amennyiben létezik ilyen.

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:

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

Láthatjuk, hogy részben hasonlít a már ismertetett CF-1-es alakra, a célfüggvényt minden esetben minimalizáljuk, ez kedvező a megoldás szempontjából, hiszen a maximum feladatokra eltérnek az optimalitási feltételek, ennek megfelelően elkerülhetjük a sok eset külön kezelését, az ebből következő hibalehetőségeket és megspóroljuk a feltételvizsgálathoz szükséges időt. Ezt megtehetjük, hiszen minden maximalizálandó célfüggvényt -1 -gyel megszorozva, és az így kapott ellentettet minimalizálva ugyanazt az eredményt kapjuk. Ami lényeges eltérés az egyes számítási alakhoz képest, az a változótipusok, és ezek korlátjai, 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 rendelkezhet.
- 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. $-\infty < x_j \leq u_j < +\infty$,
Ezek 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.

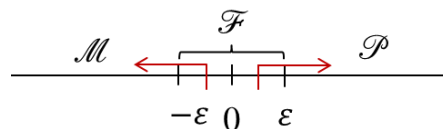
3.2.2. Toleranciaértékek használata

Az előbbiekben 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 tetszőleges változó redukált költségét megengedettnak tekintünk, ha teljesül rá az $l_j - \epsilon_{opt} \leq d_j \leq u_j + \epsilon_{opt}$ egyenlőtlenség, ahol ϵ_{opt} az optimalitási tolerancia, l_j, u_j a redukált költség megengedettségi korlátai. 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 bármelyikébe 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 a $t \geq 0$ és $\alpha_j^p < 0$, vagy $t \leq 0$ és $\alpha_j^p > 0$ esetek. Minden esetben a definiált d_j/α_j^p hányados 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. Ezt a jelenséget, mivel a toleranciával tekintett megengedettségéből következik, fake feasibilitynek (hamis megengedettség) neveztük el, erre vonatkozóan nem találtunk általános leírást a szakirodalomban.

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



3. ábra: Fake feasibility

Ezen probléma elkerülésére a következő technikát alkalmaztuk. A hányadosesztben a hányadosok abszolút érték szerinti sorba rendezése elrejtí 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 ezekről eltérően negatívak lettek. Így a sorba rendezés előtt egy egyszerű feltételes ciklussal könnyen megszabadulhatunk a hibás, negatív hányadosoktól, és így módon a törtvonalfüggvénybe nem számoljuk bele a hamis megengedett töréspontokat, ezzel elkerülve az ebből származó pontatlan célfüggvényt. A szűrés során ennek érdekében korrigáljuk 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 degenerációellenes stratégiának tekinthető, a probléma ilyenfajta kezelése újszerűnek tekinthető.

Az optimalitási tolerancia mellett használunk a PanOptban megengedettségi toleranciát (ϵ_{feas}), 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 első sorban arra használunk, hogy a nála kisebb értékeket nullának tekintjük, másrészt a már ismertett probléma miatt, miszerint egy komplex számítási sor elvégzése után nem tudjuk a számítási eredmény pontos értékét. Így ha a két érték eltérése abszolút értékben kisebb, mint az abszolút tolerancia, akkor ezeket egyenlőnek tekintjük. Emellett, ha az $|a + b| < \epsilon_{abs}$ egyenlőtlenség igaz, akkor az összeget 0-nak tekintjük. A relatív tolerancia 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 szimplex algoritmusnak, ez által sok numerikus hibát elkerülhetünk, viszont annak kérdése, hogy a megoldó mely részeiben hogyan kell ezeket alkalmazni, mint azt a fenti tárgyalásban láthattuk nem triviális.

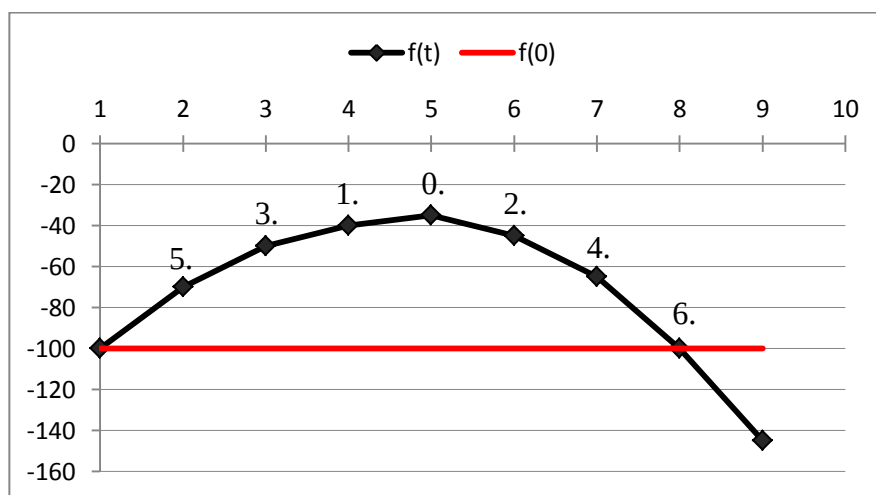
3.2.3. Stabil pivotálási eljárás

Mint már említettük, a törtvonalfüggvényes hányados-meghatározási módszer jóval több lehetőséget rejt magában, mint a hagyományos egylépéses 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 generáloelem numerikusan kicsi. Ennek vizsgálatára az úgynevezett pivot toleranciát (ϵ_{pivot}) használjuk, melynek alapértelmezett értéke 10^{-6} . [10] 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 definiálunk, 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. A szakirodalomban a heapsort algoritmust találták a legalkalmasabbnak, a PanOptban is ez a módszer került megvalósításra. [2]

A törtvonalfüggvény maximumának helyétől elindulva, célfüggvényérték szerint csökkenő sorrendben vizsgáljuk meg az adott töréspontokhoz tartozó lehetséges generálóelemet, majd, ha a toleranciánál nagyobb értéket találunk, akkor a megfelelő változót jelöljük belépő változónak. Ha a célfüggvény meredeksége negatív, de a kezdeti célfüggvényérték 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 célfüggvényérték, hiszen, ha ez alatti értékhez tartozó változót hoznánk be a bázisba, az rontana a célfüggvény értékén, ezáltal hátráltatná a megoldást. A megértés elősegítése érdekében tekintsük az alábbi célfüggvényt (**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 generálóelemnek elvégzi a numerikus vizsgálatát, majd ezt iteratíván lefelé haladva folytatja a már ismertetett korlátig. Amennyiben az eljárás nem talál jó pivotelemet, akkor üzenetet küld erről a vezérlőnek, majd az a pricingot új javító sor kijelölésére utasítja.

Ha a kis nagyságrendű generálóelemből következő hibát nem kezelnénk, akkor az a célfüggvény értékének romlásához, vagy a feladat numerikus megnehezítéséhez vezetne, amit

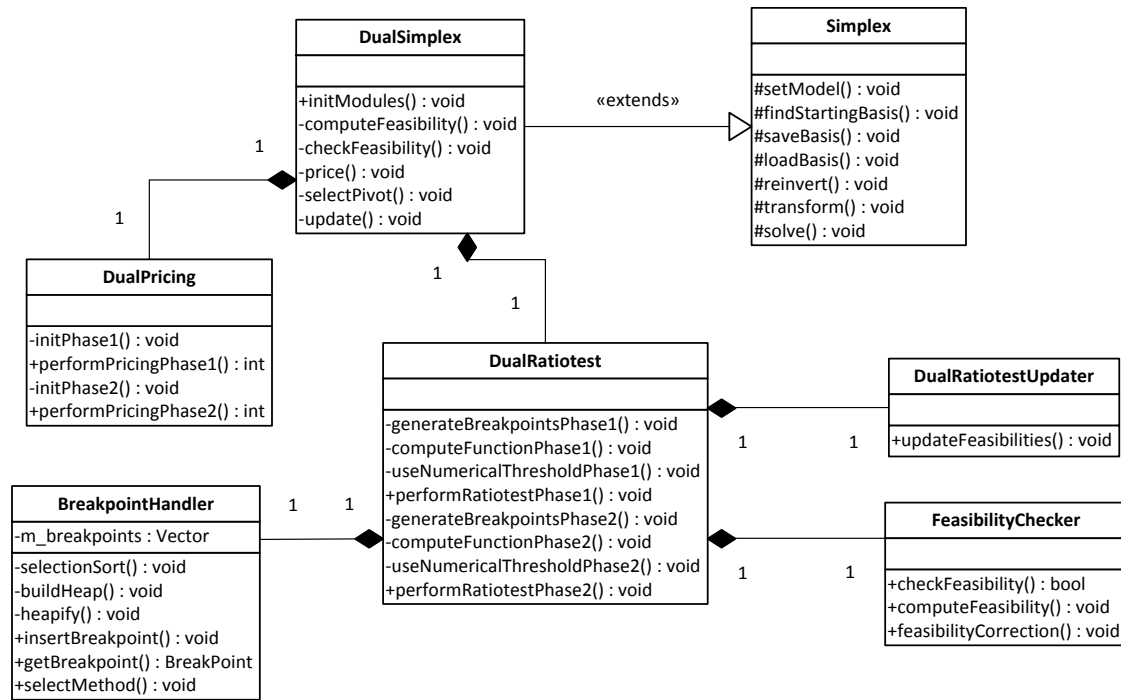
utána kezelni kell, ezért ez a megvalósítás egyfajta degeneráció ellenes stratégiának fogható fel. Újszerűsége abban rejlik, hogy a szakirodalom csak a maximumhelytől balra történő keresést alkalmazta.

Második fázisban az eljárás kicsit más formában valósítható meg, visszalépve már boundflipre kijelölt változót találunk, 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 (pl. szabad változó két töréspontot is definiálhat). Az ezekhez tartozó célfüggvényértékek megegyeznek, hiszen akármilyen meredekséget véve a nulla lépéshossz miatt nem változik a célfüggvényérték, csupán a meredekség. Az eljárás nem módosítja az előzetesen korlátcserés lépésre kijelölt váltókat, ezeket minden esetben elvégezzük.

3.3. Az implementáció struktúrája

Ebben az alfejezetben ismertetjük az implementált osztályokat, ezek kapcsolatát, a hányadosesztek felépítését. Az implementáció c++ programozási nyelven valósult meg. A megvalósítással kapcsolatban tekintsünk át néhány általános jellemzőt. Egy adott művelet elvégzéséhez a szükséges adatokat kétféleképpen biztosíthatjuk, vagy egy nyílt hozzáférésű adattaggal (osztályszintű adattag, friend osztály), vagy a függvénynek átadott paraméterrel. Az osztályszintű adattagok inicializálása minden esetben a konstruktorban történik. A dinamikus szerkezetek lefoglalását és felszabadítását általában az általunk implementált *init()* és *release()* függvények végzik. 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 fókusza a duál hányadoseszt, 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 az ehhez kapcsolódó két legfontosabb osztályt, a *DualPricingot*, illetve a *DualRatiotestet*. A következőkben tekintsük át az ábrán a hányadoseszthez kapcsolódó *FeasibilityChecker*, *DualRatiotestUpdater*, *BreakpointHandler* osztályokat, majd magát a *DualRatiotest* osztályt.



5. ábra: A hányadoseszthez kapcsolódó osztályok

3.3.1. A FeasibilityChecker osztály

Létrehoztuk a *FeasibilityChecker* osztályt, amely az egyes redukált költségek megengedettségét vizsgálja, és azok változását kezeli. A megengedettségi halmazokat partíciókra osztott kétszeresen láncolt lista formájában valósítottuk meg, így az állapotok kezelése egyszerű pointerműveletekkel valósul meg, ami rendkívül hatékony és gyors, konstans bonyolultságú. Mindehhez az alábbi függvények nyújtanak segítséget:

- *computeFeasibility()*: A redukált költségek előjele alapján kiszámolja, hogy az egyes megengedettségi halmazokba mely változóindexek kerülnek, ez tehát egy láncolt lista felépítését jelenti. A függvény mindaddig meghívásra kerül a *DualSimplex* osztályban, ameddig duál megengedett megoldást nem találunk.
- *checkFeasibility()*: A függvény megvizsgálja a redukált költségek megengedettségét, ez alapján kerül eldöntésre a vezérlőben, hogy első vagy második fázisban van az algoritmus. Mindez egy egyszerű iterátor-összehasonlítás formájában valósul meg, vizsgáljuk, hogy $\mathcal{M} = \mathcal{P} = \emptyset$, hiszen ekkor minden d_j megengedett.
- *feasibilityCorrection()*: Az első fázis végén elvégzi a megengedettségi korrekciókat az egyes típusú változókon, ezáltal a korlátos változók megengedettségét biztosítja. Emellett a függvény az algoritmus esetleges első fázisba történő visszaesése után kerül meghívásra.

3.3.2. A DualRatiotestUpdater osztály

A megengedettségi állapotok változásának hatékony kezeléséhez implementáltuk ezt az osztályt. Ennek segítségével, nem kell újraszámolni minden iterációban a megengedettségi halmazokat. Létrehoztunk egy segédvektort, amely a hányadosesztben kerül beállításra, hiszen a bejövő változó meghatározása után, tudjuk, hogy mely redukált költség melyik halmazba mozgott át, ez a felhasznált töréspontokból egyértelműen adódik. A vektor egész számokat tárol, ami a mozgás nagyságát és irányát reprezentálja, ezek a következők: $\{-2, -1, 0, 1, 2\}$. Például, ha egy változó az $\mathcal{F}$ halmazból a $\mathcal{P}$ halmazba mozog át, akkor ez az érték 1. Az egyes redukált költségek aktuális állapota és ezen értékek alapján az osztály *updateFeasibilities()* függvénye elvégzi a megengedettségek megváltozásának adminisztrálását.

3.3.3. A DualRatiotest osztály

A *DualRatiotest* osztály végzi a hányadosok definiálását, majd a paraméterben megadott módszernek megfelelően a bejövő változó kiválasztását. Szintén ebben az osztályban került megvalósításra a stabil pivotálási eljárás. Ezen feladatok elvégzésére az alábbi függvényeket implementáltuk:

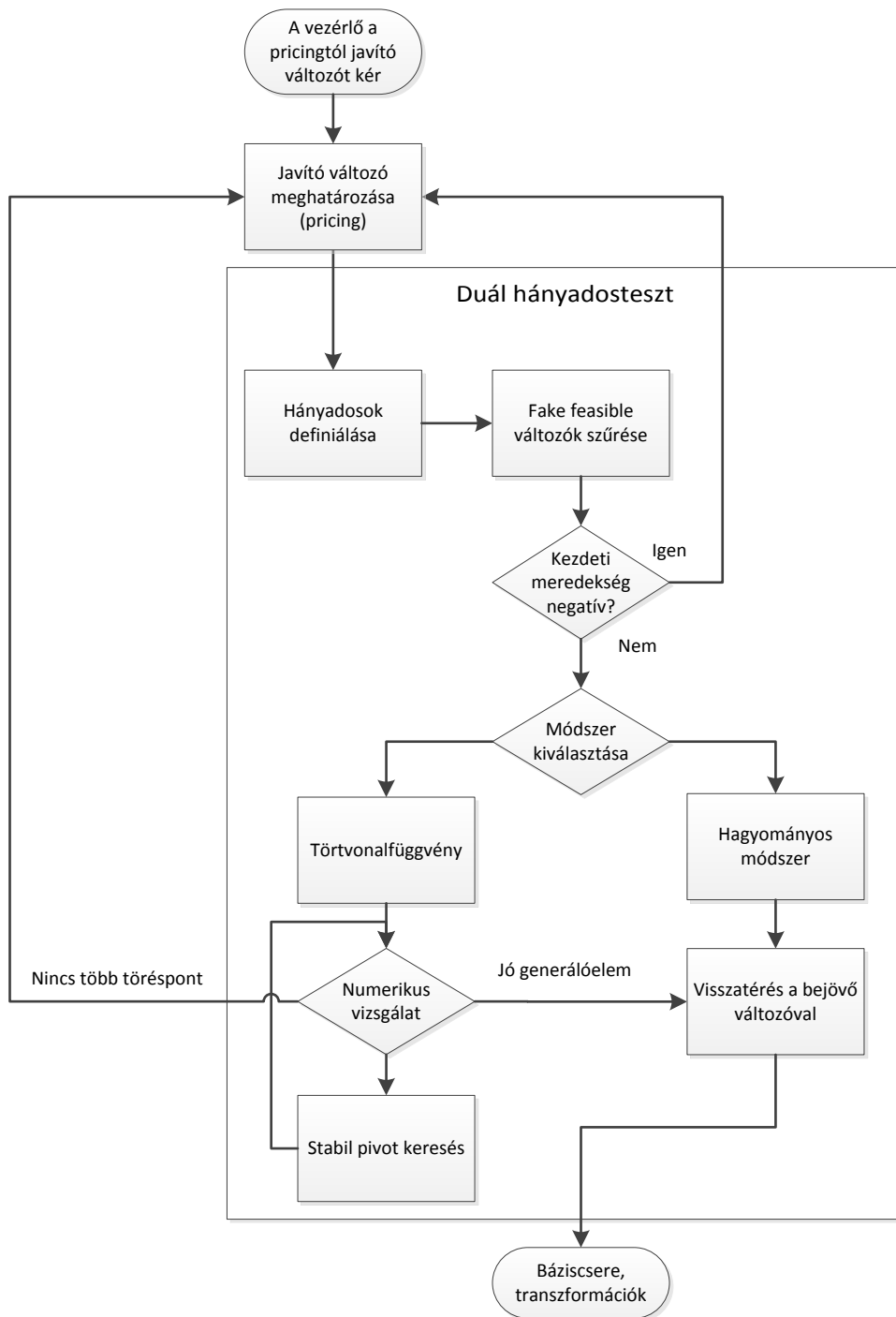
- *generateBreakpointsPhase1()*, *generateBreakpointsPhase2()*: A függvények végzik a megfelelő fázisban a hányadosok definiálását. Ehhez végigmennek a definiált megengedettségi halmazokon, és vizsgálják a redukált költségek megváltozását.
- *computeFunctionPhase1()*, *computeFunctionPhase2()*: A törtvonalfüggvények kiszámolását végzik, meghatározzák azok maximumát, illetve a maximumhelyhez tartozó bejövő változót.
- *useNumericalThresholdPhase1()*, *useNumericalThresholdPhase2()*: Az ismertett stabil pivotálási eljárást végzik a megfelelő fázisra.

3.3.4. A BreakpointHandler osztály

Ez az osztály végzi a definiált hányadosok tárolását és kezelését. Az *insertBreakpoint()* függvény segítségével adhatjuk hozzá az elemeket a tárolóhoz, amely egy vektor formájában került megvalósításra. A hányadosok definiálása után a *DualRatiotest* osztály meghívja az osztály *selectMethod()* függvényét, amely a definiált hányadosok számát figyelembe véve meghatározza a rendező algoritmust. Amennyiben kevés elem van a vektorban, akkor a *selectioSort()* metódus rendezi az összeset, ellenkező esetben pedig egy kupac adatszerkezet kerül felépítésre a *buildHeap()* függvény segítségével, valamint az egyes elemek rendezésekor a *heapify()* függvény segít a kupac tulajdonság fenntartásában. Ezen metódusok segítségével az osztály lehetőséget ad a hányadosok külön kezelésére, amelynek köszönhetően a *DualRatiotest* osztály átlátható és egyszerűen értelmezhető lett. Mindemellett tervezzük, hogy

több különböző rendezőalgoritmust is kipróbálunk az osztály segítségével, hogy ajánlást tudjunk megfogalmazni arra vonatkozóan, hogy mikor milyen rendezést célszerű használni. Továbbá a primál hányadostesztek is hatékonyan megvalósíthatók ezen osztály segítségével.

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



6. ábra: A duál hányadosteszt 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. Az első alfejezetben bemutatjuk a tesztelésre választott feladatok jellemzőit, ezt követően pedig ismertetjük a mért eredményeket, és az azokhoz kapcsolódó következtetéseket.

4.1. Tesztfeladatok

Az elvégzett méréseket a nyílt hozzáférésű *LP netlib* feladatcsomagból származó valós életről vett feladatokon végeztük. [11] A feladatok kiválasztásánál arra törekedtünk, hogy a *netlib* közel 100 példájából lehetőség szerint a nagyobb méretűeket, illetve a numerikusan nehezebb feladatokat válasszuk ki, így igazolni tudjuk az előző fejezetben ismertetett numerikus kihívások jelentőségét, illetve az általunk megvalósított technikák hatékony és helyes működését. Tekintsük át a kiválasztott *netlib* feladatokat (3. táblázat):

3. 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%
DEGEN2	445	534	4449	1,8722%
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%
MAROS-R7	3137	9408	151120	0,5120%
MODSZK1	688	1620	4158	0,3731%
NESM	663	2923	13988	0,7218%
PILOT4	411	1000	5145	1,2518%
SCAGR25	130	140	553	3,0385%
SHIP12L	1152	5427	21597	0,3454%
SHIP12S	1152	2763	21597	0,3454%
STOCFOR2	2158	2031	9492	0,2166%
STOCFOR3	16676	15695	74004	0,0283%
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 ezen 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 megoldáshoz.

A különböző futtatások kezeléséhez hatékonyan tudtuk kihasználni a PanOphoz megvalósított paraméterfájlt, amelyben kényelmesen lehet változtatni a megoldó paramétereit, így a különböző módszerek és algoritmusok változtatása a mérések elvégzése során rendkívül egyszerűvé vált. Az előző fejezetben ismertetett toleranciák alapértelmezett értékeit használtuk, ezek a következők (**4 táblázat**):

4. táblázat: Toleranciák alapértelmezett értékei

ϵ_{pivot}	ϵ_{feas}	ϵ_{opt}	ϵ_{rel}	ϵ_{abs}
10^{-6}	10^{-8}	10^{-8}	10^{-10}	10^{-12}

4.2. Elvégzett mérések

Ebben az alfejezetben ismertetjük a méréseink eredményeit, a következő elv alapján: egy adott tulajdonságot, jelenséget néhány szemléletes példa segítségével mutatunk be, táblázatos formában. A paraméterfájl kialakításának köszönhetően külön tudtuk mérni az egyes funkciókat első és második fázisra, most a következő jelenségeket, módszereket tekintjük át: hamis megengedettség, első és második fázisú törtvonalfüggvény, stabil pivotálási eljárás, közös feladatok és degeneráció. Az egyes funkciók tesztelésénél mindig csak az adott paramétert változtattuk, hogy a levont következtetések helyesek legyenek. Előljáróban még elmondanánk, hogy a *netlib* feladatokra jellemző, hogy nagyon gyorsan átkerülnek második fázisba, az iterációk nagy része (~80%) második fázisban történik, ennek megfelelően kevésbé tudtuk vizsgálni az első fázisú jelenségeket.

Fake feasibility (5. táblázat)

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.

5. 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	1,22	∞	0	866	nincs
	3514	1,65	0	0	1264	29 (61)
25FV47	14577	21,11	-1,33	16	4635	nincs
	10814	15,03	0	2	2776	42 (103)
D2Q06C	47096	329,24	0	474	4344	nincs
	26457	163,46	0,5	37	3026	6983 (88860)

Első fázisú törtvonalfüggvény

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

6. táblázat: Példák az első fázisú törtvonalfüggvény hatékonyságára

Feladat	Iterációszám	Futási idő (mp)	Első fázisú módszer
BOEING1	935	0,28	egy lépéses
	848	0,25	törtvonalfüggvény
BOEING2	532	0,1	egy lépéses
	240	0,04	törtvonalfüggvény
DEGEN3	9323	35,7	egy lépéses
	8296	31,09	törtvonalfüggvény
NESM	3276	2,05	egy lépéses
	2953	1,88	törtvonalfüggvény
PILOT4	-	-	egy lépéses
	4301	6,92	törtvonalfüggvény
SCAGR25	685	0,12	egy lépéses
	607	0,12	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 (**7. táblázat**) láthatjuk.

¹ Az algoritmus által optimálisnak elfogadott és a *netliben* ismertett optimális célfüggvényérték 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.

7. táblázat: Példák a második fázisú törtvonalfüggvény hatékonyságára

Feladat	Iterációszám	Futási idő (mp)	Második fázisú módszer
80BAU3B	6574	10,3	egylépéses
	4235	6,45	törtvonalfüggvény
BOEING1	935	0,27	egylépéses
	650	0,16	törtvonalfüggvény
BOEING2	260	0,04	egylépéses
	201	0,028	törtvonalfüggvény
FIT2D	7674	21,81	egylépéses
	187	1	törtvonalfüggvény
FIT2P	14713	63,53	egylépéses
	6655	36,36	törtvonalfüggvény
GREENBEB	22474	51,27	egylépéses
	17640	39,65	törtvonalfüggvény
NESM	5171	3,15	egylépéses
	3783	2,41	törtvonalfüggvény
PILOT4	4301	6,92	egylépéses
	2012	3,21	törtvonalfüggvény
VTP-BASE	323	0,04	egylépéses
	257	0,04	törtvonalfüggvény

Stabil pivotálási eljárás

Az alábbi feladatokban (**8. 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*, *MAROS-R7*) az algoritmus nélkül nem sikerült megoldani a feladatot.

8. táblázat: Példák a stabil pivotálási eljárás hatékonyságára

Feladat	Iterációszám	Futási idő (mp)	Degenerált iterációk száma	Stabil pivotálás
BOEING2	189	0,027	90	Nincs
	174	0,025	78	Van
D2Q06C	-	-	-	Nincs
	26457	163,46	37	Van
DEGEN3	8296	31,15	623	Nincs
	8007	28,12	683	Van
MODSZK1	1117	0,37	1048	Nincs
	698	0,22	639	Van
GREENBEA	-	-	-	Nincs
	9150	17,02	2969	Van
GREENBEB	20589	47,86	13062	Nincs
	13330	30,54	5496	Van

Feladat	Iterációszám	Futási idő (mp)	Degenerált iterációk száma	Stabil pivotálás
MAROS R-7	-	-	-	Nincs
	4608	145,11	746	van
VTP-BASE	257	0,036	47	nincs
	231	0,032	56	van
WOOD1P	2231	13,73	2209	nincs
	328	3,02	318	van
WOODW	48862	112,91	48114	nincs
	2167	8,71	1858	van

Közömbös feladatok

Néhány feladat esetében megfigyelhetjük (**9. táblázat**), hogy nagyon kis mértékben változik a megoldás menete a különböző módszerek hatására, ezek numerikusan stabilabb feladatok. Bizonyos esetben a fejlettebb módszerek hatására más irányba indul el az algoritmus, ami hosszabb megoldást eredményezhet. Ennek oka részben a PanOptban használt Dantzig pricing, ami egy meglehetősen egyszerű eljárás, egy fejlettebb technikánál ennek a valószínűsége feltehetőleg kisebb, de akkor is előfordulhat, mivel ezek heurisztikus eljárások. [12]

9. táblázat: Példák közömbös feladatokra

Feladat	Iterációszám	Futási idő (mp)	Módszer mindkét fázisban
BNL2	6475	7,44	egy lépéses
	6475	7,41	törtvonalfüggvény
	7114	7,98	stabil pivotálás
DEGEN2	1575	1	egy lépéses
	1138	0,66	törtvonalfüggvény
	1138	0,65	stabil pivotálás
SHIP12L	1182	0,81	egy lépéses
	1182	0,84	törtvonalfüggvény
	1182	0,77	stabil pivotálás
SHIP12S	1124	0,49	egy lépéses
	1124	0,49	törtvonalfüggvény
	1124	0,47	stabil pivotálás
TRUSS	7714	24,29	egy lépéses
	7714	24,98	törtvonalfüggvény
	7896	26,39	stabil pivotálás
STOCFOR2	2183	1,29	egy lépéses
	2383	1,68	törtvonalfüggvény
	2377	1,65	stabil pivotálás
STOCFOR3	17373	103,18	egy lépéses
	19150	130,68	törtvonalfüggvény
	19150	129,79	stabil pivotálás

Degeneráció

Az alábbi táblázatban (10. táblázat) azt láthatjuk, hogy a megvalósított technikák szinte mindegyike képes valamilyen mértékben csökkenteni a degenerált iterációk számát, ennek köszönhetően általában hatékonyabb megoldást biztosítanak.

10. táblázat: Példák a degeneráció jelenségre

Feladat	Iterációs szám	Futási idő (mp)	Degenerált iterációk száma	Módszer
STOCFOR2	2183	1,31	883	egy lépéses
	2383	1,69	481	törtvonalfüggvény (első fázis)
STOCFOR3	17373	103,6	6387	egy lépéses
	19150	128,3	3361	törtvonalfüggvény (első fázis)
80BAU3	6574	10,3	1421	egy lépéses
	6663	10,34	1382	törtvonalfüggvény (első fázis)
	4235	6,45	1189	törtvonalfüggvény (második fázis)
FIT2D	7674	21,5	553	egy lépéses
	187	1	4	törtvonalfüggvény (második fázis)
FIT2P	14713	63,53	1535	egy lépéses
	6655	36,36	89	törtvonalfüggvény (második fázis)
VTP-BASE	323	0,04	98	egy lépéses
	257	0,04	47	törtvonalfüggvény (második fázis)
NESM	2951	2	2091	fake feasible inaktív
	2769	1,88	1828	fake feasible aktív
MAROS-R7	4776	385,29	822	fake feasible inaktív
	4608	145,11	746	fake feasible aktív
D2Q06C	47096	329,24	4344	fake feasible inaktív
	26457	163,46	3026	fake feasible aktív
GREENBEB	18656	40,7	11521	törtvonalfüggvény (mindkét fázis)
	13330	30,46	5496	stabil pivotálás
MODSZK1	1117	0,36	1048	törtvonalfüggvény (mindkét fázis)
	698	0,22	639	stabil pivotálás
TRUSS	7714	24,98	1463	törtvonalfüggvény (mindkét fázis)
	7896	26,39	1333	stabil pivotálás
WOOD1P	2231	13,96	2209	törtvonalfüggvény (mindkét fázis)
	328	3,12	318	stabil pivotálás
WOODW	48862	110,34	48114	törtvonalfüggvény (mindkét fázis)
	2167	8,55	1858	stabil pivotálás

Összefoglalva láthatjuk tehát a megvalósított technikák hatékony működését, az általunk megfigyelt hamis megengedettség jelenlétét, az erre kidolgozott algoritmus helyességét, a törtvonalfüggvényes hányados-meghatározás, valamint a stabil pivotálási eljárások hatékonyságát. Elmondhatjuk, hogy az általunk megvalósított technikák részben újszerűek, valamint olyan implementációval párosulnak, amelyek nagymértékben elősegítik a megoldást.

5. Összefoglalás

A dolgozat készítése során részt vettem egy fejlett lineáris programozási feladatmegoldó megvalósításában 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, az ehhez kapcsolódó különböző matematikai modellek jellemzőit, valamint bemutattuk, hogy mik a terület legfontosabb alapfogalmai és összefüggései. Ezt követően ismertettük a primál-duál feladatpár, és a duál szimplex algoritmus főbb jellemzőit, hogy a dolgozat érdemi részét képező duál hányadoseszteket érdemben tudjuk tárgyalni. Ez után részletesen bemutattuk a duál hányadoseszteket, ismertettük és értékeltük a legfontosabb hányados-meghatározási módszereket.

Részletesen bemutattuk az általunk végzett munkát, a duál hányadosesztek megvalósítását, valamint annak jellemzőit. Kitértünk a nagyméretű LP feladatok megoldásához kapcsolódó kihívásokra, valamint, hogy ezeknek milyen technikák és algoritmusok megvalósításával sikerült megfelelnünk. 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. Ismertettük továbbá a hányadosesztek megvalósításának struktúráját, a megvalósított osztályokat, az ehhez kapcsolódó legfontosabb jellemzőket.

Végül ismertettük az általunk elvégzett mérési eredményeket, állításainkat 25 közismert LP feladat segítségével támasztottuk alá, melyek kiválasztásakor a nagy méret mellett a rossz numerikus tulajdonságú feladatokat részesítettük előnyben. Dolgozatunk mellett, hogy a duál szimplex algoritmus hányadoseszteinek elméletét részletesen bemutatja, gyakorlati jelentőségű implementációs kihívásokra is kitér.

Jövőbeni terveink között szerepel a primál-duál kapcsolat további vizsgálata, a megvalósított algoritmusok gyorsítása, hatékonyabbá tétele. Tervezzük, hogy a primál szimplex algoritmust a szakirodalmi leírások implementálása mellett, hatékony technikákkal egészítsük ki, hogy mindkét megoldó megvalósítása magas színvonalú, megbízható és gyors legyen. Emellett a degeneráció mélyebb vizsgálatát, illetve további hatékony degeneráció-ellenes stratégiák megismerését és megvalósítását tűztük ki célul. Hosszú távon munkámmal szeretnék hozzájárulni egy hatékony, és a mai piaci megoldókkal versenyképes szoftver kidolgozásában és fejlesztésében. Emellett szeretném elmélyíteni ismereteimet a lineáris programozás tudományában, az elméleti alapok elsajátítása mellett a fejlett technikák hatékony megvalósításának vizsgálatát egyaránt fontosnak tartom.

6. 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., 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*, 26., 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*, 5., pp. 257-261, 1958.
- [10] **P. Harris**, „Pivot Selection Method of the Devex LP Code,” *Mathematical Programming*, 1973.
- [11] **D. Gay**, „Electronic mail distribution of linear programming test problems,” *COAL Newsletter*, pp. 10-12, 1985.
- [12] **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.”