Alkalmazott Matematikai Lapok **23** (2006), 139–161.

A DUÁL SZIMPLEX ALGORITMUS ELSŐ FÁZISÁNAK VIZSGÁLATA

MAROS ISTVÁN

London

A cikk egy új, a szimplex módszer számára készített duál első fázis eljárás elméleti és számítástechnikai vizsgálatával foglalkozik. Megmutatja, hogy a GDPO algoritmus lényegesen jobb teljesítményt képes nyújtani egy duál megengedett bázis megtalálásában, mint a hagyományos módszer. A 48 feladaton végrehajtott kísérletek meggyőzően igazolják, hogy az algoritmus előnyös elméleti tulajdonságai a gyakorlatban ténylegesen és nagy mértékben érvényesülni tudnak.

1. Bevezetés

A lineáris programozási (LP) feladat megoldására Dantzig [1] által kidolgozott szimplex algoritmus különféle variánsai rendkívül fontos szerepet játszanak az optimalizálás elméletében és gyakorlatában. A Karmarkar [3] által bevezetett és sok szerző (cf. Terlaky et al. [7, 8]) által továbbfejlesztett belsőpontos algoritmusok (BPA) nem homályosították el a szimplex alapú algoritmusok jelentőségét. Bebizonyosodott, hogy a két algoritmus család jól kiegészíti egymást. Bizonyos feladattípusokra a BPA-k a jobbak, másokra pedig a szimplex eljárások hatékonyabbak.

A szimplex módszernek két alapvető változata van: a primál és a duál algoritmus. Kezdetben a primál módszer kapta a több figyelmet. Ennek megfelelően a (primál) szimplex alapú programrendszerek folyamatos algoritmikus és implementációs fejlődésen mentek át és egyre nagyobb feladatokat tudtak megoldani egyre megbízhatóbban és hatékonyabban. A duál szimplex ettől jelentősen elmaradt, és használata elsősorban arra az esetre korlátozódott, amikor egy megengedett (fizibilis) bázis a rendelkezésre állt. Ez volt a helyzet az egészértékű LP feladatok egyszerű branch-and-bound (B&B) módszerrel történő megoldása esetén. Ekkor ugyanis egy származtatott relaxált LP feladatra nézve a megelőző LP relaxáció optimális bázi-

sa egy duál megengedett bázis. Erről indulva a duál módszernek általában sokkal kevesebb iterációra van szüksége, mint a primál módszernek (lásd pl. [4]).

A korszerű B&B algoritmusok lokális technikákat alkalmaznak a kereső fa csúcsainál, mint például *logical testing, implied bounds, added cuts.* Ilyen esetekben már nem igaz, hogy az előd LP feladat optimális bázisa duál megengedett marad a származtatott feladat számára. Így a duál algoritmus duál első fázissal tud csak indulni, de még mindig ez a jobb választás a primállal szemben.

A kedvező tapasztalatok alapján felmerült a kérdés, vajon a duál algoritmus egyenértékű, esetleg jobb alternatívája tud-e lenni a primálnak nagyméretű LP feladatok megoldására. A duál második fázis sikeres fejlesztése [4, 5, 2] után a duál első fázis algoritmikus maradt a hiányzó láncszem. Ezen dolgozva jutott el a szerző a GDPO algoritmushoz [6, 5], amely a fenti kérdésre pozitív választ adott. Elméletileg kimutatható, hogy az új algoritmusnak számos kedvező tulajdonsága van. Az, hogy ezek a valóságban milyen mértékben realizálódnak, még nem volt ismeretes. Jelen cikk ezt a hiányt pótolja és pozitív következtetésekre jut.

A továbbiakban a 2. szakasz a vizsgált általános alakú LP feladatot fogalmazza meg, amit a 3. szakaszban a GDPO algoritmus leírása követ. A 4. szakasz GDPO elméleti vizsgálatát tárgyalja, míg a 5. szakasz a számítástechnikai vizsgálatot mutatja be. A 6. szakasz a következtetéseket összegzi.

2. A feladat megfogalmazása

A gyakorlat szempontjából fontos és az elmélet számára sem közömbös, hogy az LP feladatot a természetes felmerülésnek megfelelő formában lehessen kezelni. Ez szükségessé teszi az előforduló összes típusú változó és feltétel algoritmikus kezelését. Ez nem pusztán esztétikai kérdés, hanem a hatékonyság növelésének a forrása is.

2.1. A primál feladat

Egy LP feladat legáltalánosabb alakja a következő:

$$\min z = c_0 + c_1 x_1 + \cdots + c_n x_{\bar{n}},$$

feltéve, hogy a közös korlátok

$$L_i \leq \sum_{j=1}^{\bar{n}} a_{ij} x_j \leq U_i, \quad i = 1, \ldots, m,$$

és a változók egyedi korlátai

$$\ell_j \leq x_j \leq u_j \quad j = 1, \ldots, \bar{n}$$

is teljesülnek. Az alsó $(L_i$ vagy $\ell_j)$ korlátok közül bármelyik lehet $-\infty$. Hasonlóan, a felső $(U_i$ or $u_j)$ korlátok között is lehet $+\infty$.

Egyszerű átalakítások és a közös feltételekhez egy-egy z_i logikai változó hozzáadása után a feltételek az alábbi alakra hozhatók:

$$z_i + \sum_{j=1}^{\bar{n}} a_{ij} x_j = b_i, \quad i = 1, \ldots, m, \tag{1}$$

ahol a változók egyedi korlátai:

(2)

Megengedett tartomány					Típus	Hivatkozás
		z_i, x_j	$=$	0	0	Fix
0	$\leq$	z_i, x_j	$\leq$	u_j	1	Korlátos
0	$\leq$	z_i, x_j	$\leq$	$+\infty$	2	Nem-negatív
$-\infty$	$\leq$	z_i, x_j	$\leq$	$+\infty$	3	Szabad

A változók típusára $\text{type}(x_j)$ vagy $\text{type}(z_i)$ módon fogunk hivatkozni.

Az LP feladat az (1) feltételrendszerrel mátrix alakban is felírható:

$$\min \ \mathbf{c}^T\mathbf{x} \tag{3}$$

$$\text{s.t.} \ \ \mathbf{A}\mathbf{x} + \mathbf{I}\mathbf{z} = \mathbf{b} \tag{4}$$

és a változók eleget tesznek (2)-nek.

Az x változókat *strukturális*, a z változókat pedig *logikai* változóknak nevezzük. Miután c_0 nem játszik szerepet az optimalizálásban, (3)-ban már nem tüntettük fel.

Technikai és algoritmikus szempontból a logikai és strukturális változók egyenrangú szerepet játszanak és általában nem szükséges megkülönböztetni őket. Ha a (4) bal oldalán álló mátrixot és változókat újra definiáljuk:

$$\mathbf{x} := \begin{bmatrix} \mathbf{x} \\ \mathbf{z} \end{bmatrix}, \quad \mathbf{c} := \begin{bmatrix} \mathbf{c} \\ \mathbf{0} \end{bmatrix}, \quad \mathbf{A} := [\mathbf{A} \mid \mathbf{I}],$$

ahol $\mathbf{0}$ az m dimenziós null vektor, akkor a feladat a következő alakra hozható:

(5)

$$\boxed{\begin{aligned} &\min \mathbf{c}^T\mathbf{x} \\ &\text{s.t.} \ \ \mathbf{A}\mathbf{x} = \mathbf{b} \\ &\qquad \boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u}, \end{aligned}}$$

az $\boldsymbol{\ell}$ és $\mathbf{u}$ vektorok értelemszerű kiterjesztése után. $\mathbf{x}$ és $\mathbf{c}$ új dimenziója $n = \bar{n} + m$. (5)-öt az LP feladat primál alakjának, vagy primál LP feladatnak szokás nevezni.

(5) egy bázisát $\mathbf{B}$-vel jelöljük, míg a bázisváltozók indexhalmazát $\mathcal{B}$-vel, a többi változó indexhalmazát pedig $\mathcal{R}$-rel. Az összes változó indexhalmaza: $\mathcal{N} = \mathcal{B} \cup \mathcal{R}$ és

$|\mathcal{N}| = n$. Az általánosság megszorítása nélkül feltesszük, hogy $\mathbf{B}$ az $\mathbf{A}$ mátrix első m oszlopa. Ez $\mathbf{A}$ particionálását eredményezi $\mathbf{A} = [\mathbf{B} \mid \mathbf{R}]$ alakban, ahol $\mathbf{R}$ jelöli $\mathbf{A}$ nem-bázis részét. Ennek megfelelően particionáljuk $\mathbf{x}$-et és $\mathbf{c}$-t is. $\mathbf{A}$ mátrix j-edik oszlopát $\mathbf{a}_j$-vel, míg i-edik sorát $\mathbf{a}^i$-vel jelöljük. (5)-nek a $\mathbf{B}$-hez tartozó bázis megoldása

$$\mathbf{x}_B = \boldsymbol{\beta} = \mathbf{B}^{-1}\Big(\mathbf{b} - \sum_{j \in \mathcal{U}} u_j \mathbf{a}_j\Big),$$

ahol $\mathcal{U}$ jelöli azon bázison kívüli változók indexhalmazát, amelyek felső korláton vannak. Az i-edik bázisváltozót β_i-vel jelöljük. A d_j redukált költség definíciója $d_j = c_j - \boldsymbol{\pi}^T \mathbf{a}_j = c_j - \mathbf{c}_B^T \mathbf{B}^{-1} \mathbf{a}_j$, ami tovább egyenlő $c_j - \mathbf{c}_B^T \boldsymbol{\alpha}_j$-vel, ha az $\boldsymbol{\alpha}_j = \mathbf{B}^{-1}\mathbf{a}_j$ jelölést használjuk.

2.2. A duál feladat

A duál feladathoz a primál dualizálásán keresztül jutunk el. Először a primál feladat azon változatát tekintjük, ahol minden változó 2-es típusú (nem-negatív):

Ha a primál

$$\begin{aligned} \text{(P1)} \quad & \min \ \mathbf{c}^T\mathbf{x} \\ & \text{s.t.} \ \mathbf{A}\mathbf{x} = \mathbf{b}, \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

alakban van megadva, akkor ennek a duálja

$$\begin{aligned} \text{(D1)} \quad & \max \ \mathbf{b}^T\mathbf{y} \\ & \text{s.t.} \ \mathbf{A}^T\mathbf{y} \leq \mathbf{c}. \end{aligned}$$

Megjegyzendő, hogy itt az $\mathbf{y}$ duál változók 3-as típusú (szabad) változók.

A $\mathbf{w} = [w_1, \ldots, w_n]^T$ duál logikai változók bevezetésével (D1) felírható egyenlőség formában:

$$\max \ \mathbf{b}^T\mathbf{y}$$

$$\text{s.t.} \ \mathbf{A}^T\mathbf{y} + \mathbf{w} = \mathbf{c}, \tag{6}$$

$$\mathbf{w} \geq \mathbf{0}. \tag{7}$$

Legyen $\mathbf{B}$ az $\mathbf{A}$ mátrix egy bázisa, aminek nem kell primál megengedettnek lenni. (6) átrendezésével a $\mathbf{w}^T = \mathbf{c}^T - \mathbf{y}^T\mathbf{A}$ alakot kapjuk, ami particionált formában:

$$\mathbf{w}_B^T = \mathbf{c}_B^T - \mathbf{y}^T\mathbf{B},$$

$$\mathbf{w}_R^T = \mathbf{c}_R^T - \mathbf{y}^T\mathbf{R}.$$

A (7) alatti, $\mathbf{w}$-re vonatkozó nem-negativitási feltétel particionálva: $[\mathbf{w}_{\mathcal{B}}^T, \mathbf{w}_{\mathcal{R}}^T]^T \geq \mathbf{0}$. Az $\mathbf{y}^T = \mathbf{c}_{\mathcal{B}}^T \mathbf{B}^{-1}$ választással azt kapjuk, hogy

$$\mathbf{w}_{\mathcal{B}}^T = \mathbf{c}_{\mathcal{B}}^T - \mathbf{c}_{\mathcal{B}}^T \mathbf{B}^{-1}\mathbf{B} = \mathbf{0}, \tag{8}$$

$$\mathbf{w}_{\mathcal{R}}^T = \mathbf{c}_{\mathcal{R}}^T - \mathbf{c}_{\mathcal{B}}^T \mathbf{B}^{-1}\mathbf{R} = \mathbf{d}_{\mathcal{R}}^T \geq \mathbf{0}, \tag{9}$$

ahol $\mathbf{d}_{\mathcal{R}}$ a primál redukált költségekből alkotott vektor bázison kívüli részét jelenti. Mivel (8) teljesül tetszőleges bázis esetén és $\mathbf{y}$ szabad változó, a $\mathbf{B}$ bázis duál megengedett, ha kielégíti (9)-et. Ez azonban nem más, mint a primál optimalitási feltétel. Vagyis a duál megengedettségi feltétel azonos a primál optimalitási feltétellel és a primál redukált költségek egyben a duál logikai változók.

A gyakorlatban a duál szimplex algoritmusok a primál feladaton dolgoznak, használva annak számítástechnikai eszköztárát, de a báziscserét a duál algoritmus szabályai szerint hajtják végre.

Nagyméretű LP feladatokat a gyakorlatban csak akkor lehet megoldani, ha kihasználjuk a feltételi mátrixok ritkásságát (kis kitöltöttségét). Ez a módosított szimplex módszer keretein belül valósítható meg a legjobban. Ilyen esetben csak az iterációhoz szükséges transzformált elemeket kell meghatározni. A duál szimplex algoritmusoknál mind első, mind második fázisban számítástechnikailag a „legdrágább" művelet a transzformált pivot sor előállítása. Ennek a sornak az esetleges többszörös felhasználása jelentősen motiválta a szerzőnek az általánosított duál első fázis eljárásra irányuló vizsgálatait.

A 3. szakaszban bemutatandó új algoritmus, amelyre GDPO (Generalized Dual Phase One) néven fogunk hivatkozni, egy iterációval képes annyi haladást elérni, mint a hagyományos módszer sok iterációval. Mindezt a drágán előállított transzformált pivot sor többszöri felhasználásával éri el. Ezen túlmenően, a GDPO-nak olyan további tulajdonságai is vannak, amelyek algoritmikusan és számítástechnikailag is egyaránt igen előnyösek. Ezeket a 4. és 5. szakaszok tárgyalják.

3. A GDPO algoritmus leírása

3.1. Elméleti alapok

A GDPO algoritmus részletesebb tárgyalása Maros eredeti cikkében (l. [6]) található.

Az LP (5) alatti általános esetére is igaz (lásd pl. [5]), hogy a primál redukált költségek azonosak a duál logikai változókkal. Ezért az (5) minimalizálási feladat duáljának a megengedett megoldásai kielégítik a következő feltételeket. Megjegyzendő, hogy a d_j duál logikai változók a primál strukturális változókkal állnak összefüggésben.

(10)

Type(x_j)	Érték	d_j	Megjegyzés
0	$x_j = 0$	Mindegy	
1	$x_j = 0$	≥ 0	
1	$x_j = u_j$	≤ 0	$j \in \mathcal{U}$
2	$x_j = 0$	≥ 0	
3	$x_j = 0$	$= 0$	

Ebből adódóan egy $(\mathcal{B},\mathcal{U})$ halmazpárhoz tartozó duál megoldás megengedett, ha teljesíti (10)-et.

Miután a fix (type-0) primál változók redukált költsége mindig egy megengedett d_j érték, ezeket a változókat kihagyhatjuk a duál megengedettség vizsgálatából. Hasonló a helyzet a korlátos (type-1) változókkal is. Ha ugyanis egy type-1 változóhoz tartozó d_j előjele nem teljesíti (10)-et, akkor a primál változó értékét az ellenkező korlátra állítva d_j duál megengedett lesz. Ez a lépés nem igényel bázischerét, mindössze a primál bázismegoldást kell transzformálni. Ugyanis a korlátos primál változókhoz tartozó d_j kétféleképpen lehet nem-megengedett (infízibilis). Formálisan: legyenek

$$\mathcal{T}^+ = \{j : \text{ type}(x_j) = 1,\, x_j = u_j,\, d_j > 0\}$$

$$\mathcal{T}^- = \{j : \text{ type}(x_j) = 1,\, x_j = 0,\, d_j < 0\}$$

a type-1 duál infízibilis indexek halmazai. Ekkor a primál változók korlátcseréje után a megoldás transzformációja:

$$\beta := \beta - \sum_{j\in\mathcal{T}^+} u_j\alpha_j + \sum_{j\in\mathcal{T}^-} u_j\alpha_j, \tag{11}$$

ahol $\alpha_j = \mathbf{B}^{-1}\mathbf{a}_j$, ':=' értékadást jelent, és a szumma értéke nulla, ha a vonatkozó indexhalmaz üres. α_j-k kiszámítása a (11)-ben szereplő összes j-re nagyon számításigényes lenne. Azonban az egész műveletet egyetlen lépésben el lehet végezni a következő módon:

$$\begin{aligned}\beta &:= \beta - \sum_{j\in\mathcal{T}^+} u_j\alpha_j + \sum_{j\in\mathcal{T}^-} u_j\alpha_j \\ &= \beta - \mathbf{B}^{-1}\Big(\sum_{j\in\mathcal{T}^+} u_j\mathbf{a}_j - \sum_{j\in\mathcal{T}^-} u_j\mathbf{a}_j\Big) \\ &= \beta - \mathbf{B}^{-1}\tilde{\mathbf{a}}\end{aligned} \tag{12}$$

$\tilde{\mathbf{a}}$ nyilvánvaló értelmezésével. $\tilde{\mathbf{a}}$ (egyszerű) meghatározása után csak egyetlen FTRAN műveletet (lásd pl. [5]) kell végrehajtani a bázis inverzével. (12)-t *duál megengedettségi korrekció*nak nevezzük. Ezt a műveletet elég akkor végrehajtani, amikor már sikerült (ha lehet) az összes type-2 és type-3 pozíció duál logikai változóját megengedett értékre hozni.

Fentiek alapján elegendő a type-2 és type-3 változókkal foglalkozni. Ezekre két infízibilitási indexhalmaz határozható meg:

$$\mathcal{M} = \{j : d_j < 0 \text{ és } \mathrm{type}(x_j) \geq 2\}, \tag{13}$$

$$\mathcal{P} = \{j : d_j > 0 \text{ és } \mathrm{type}(x_j) = 3\}. \tag{14}$$

Ezek segítségével a duál infízibilitások összegét a következőképpen definiáljuk:

$$f = \sum_{j \in \mathcal{M}} d_j - \sum_{j \in \mathcal{P}} d_j, \tag{15}$$

ahol bármely szumma egyenlő nullával, ha a vonatkozó indexhalmaz üres. Nyilvánvalóan mindig igaz, hogy $f \leq 0$.

Duál első fázisban a cél f maximalizálása. Ha $f = 0$-t el tudjuk érni, akkor a megoldás duál megengedett lesz egy esetleges megengedettségi korrekció után. Ha ez nem érhető el, akkor a feladatnak nincs duál megengedett megoldása.

A duál algoritmusok előbb a bázisból kilépő változót határozzák meg, ami egyben a pivot sort is definiálja a soronlévő iterációra. Tegyük fel, hogy a p-edik sort választottuk ki, mert egy később tárgyalandó kritérium alapján ez bizonyult a legjobbnak. A szimplex iteráció ennek a transzformált sornak bizonyos t többszörösét vonja le a duál logikai változók sorából: $\mathbf{d}_{\mathcal{R}}(t) = \mathbf{d}_{\mathcal{R}} - t\boldsymbol{\alpha}^p$. Így a duál logikaiak, mint a t függvényei:

$$d_j^{(p)}(t) = d_j - t\alpha_{pj}. \tag{16}$$

Ezzel a jelöléssel $d_j^{(p)}(0) = d_j$. Ha t kellően kicsi úgy, hogy az $\mathcal{M}$ és $\mathcal{P}$ halmazok változatlanok, az infízibilitások összege a t függvényében a következőképpen fejezhető ki:

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

Látható, hogy a (15)-beli f-et $t = 0$-ra kapjuk meg: $f = f^{(p)}(0)$. Könnyen kimutatható, hogy a soron következő tárgyalás akkor is igaz, ha a halmazok csak a $t = 0$ esetben változatlanok (degeneráció).

A duál infízibilitások megváltozása, ha t elmozdul a nulla értékről:

$$\Delta f = f(t) - f(0) = -t\left(\sum_{j \in \mathcal{M}} \alpha_{pj} - \sum_{j \in \mathcal{P}} \alpha_{pj} \right). \tag{17}$$

Ha bevezetjük a

$$v_p = \sum_{j \in \mathcal{M}} \alpha_{pj} - \sum_{j \in \mathcal{P}} \alpha_{pj} \tag{18}$$

jelölést, (17) a $\Delta f = -tv_p$ alakban írható. Ennek következtében a duál infízibilitások javulásának a követelménye, $\Delta f > 0$, ekvivalens a

$$-tv_p > 0$$

követelménnyel. Ezt pedig kétféleképpen lehet elérni:

(19) Ha $v_p > 0$, akkor $t < 0$-nak kell teljesülni,

ha $v_p < 0$, akkor $t > 0$-nak kell teljesülni.

Mindaddig, amig van olyan i sor hogy type$(\beta_i) \neq 3$ és a hozzá tartozó (18) által definiált v értéke nem nulla, lehetőség van a duál első fázis célfüggvény javítására. A feltételek pontos kimunkálása a későbbiekben történik meg. Amennyiben több sor is potenciális javító, akkor ezek közül valamilyen egyszerű vagy összetett szabály alapján választunk („dual phase-1 pricing").

Jelöljük a p-edik bázisváltozó β_p eredeti indexét $\mathbf{A}$-ban μ-vel. Így $x_\mu = \beta_p$ a kiválasztott kilépő változó. Ezen a ponton kikötjük, hogy a kilépő változó redukált költsége, $\bar{d}_\mu$, duál megengedett legyen a bázisból való kilépés után. Ez nem feltétlenül szükséges, de így a duál megengedettség jobban kézben tartható.

Ha t elmozdul nulláról, egyes d_j-k elmozdulnak a nulla irányába, ami a megengedettségi tartományuk határa, és bizonyos t esetén el is érik azt. Ezek az értékek (16) alapján $d_j - t\alpha_{pj}$-ból határozhatók meg:

$$t_j = \frac{d_j}{\alpha_{pj}} \quad \text{bizonyos } j \in \mathcal{R} \text{ indexekre,}$$

és ezek lehetővé tesznek egy báziscserét, mivel itt $d_j(t)$ nullává válik. Ez azt is jelenti, hogy a j-edik duál feltétel ekkor egyenlőség formában teljesül. Ha a több lehetséges j közül kiválasztott indexet q-val jelöljük és ezt a változót vonjuk be a bázisba x_μ helyett, akkor (a szimplex transzformációs képlete alapján) azt kapjuk, hogy

$$\bar{d}_\mu = -\frac{d_q}{\alpha_{pq}} = -t_q,$$

aminek duál megengedettnek kell lenni a fentiek szerint. $\bar{d}_\mu$ előjelét az határozza meg, hogy x_μ hogyan (alsó vagy felső korláton) hagyja el a bázist. Ez rögtön szabályt is ad arra, hogy a belépő változó hogyan határozható meg, ha a kilépőt már megválasztottuk. Alább a szabály szóbeli formáját adjuk, a részletes bizonyítás [6]-ban található.

1. Ha $v_p > 0$, akkor $t_q < 0$ kell, hogy (19) teljesüljön. Ez maga után vonja, hogy a p-edik bázisváltozónak alsó korláton kell kilépni (ugyanis $\bar{d}_\mu \geq 0$ szükséges a duál megengedettséghez). Duál degeneráció esetétől eltekintve ez azt jelenti, hogy d_q-nak és α_{pq}-nak ellenkező előjelűnek kell lenni, vagyis a potenciális pivot pozícióknak teljesíteni kell ezt a követelményt.

2. Ha $v_p < 0$, akkor $t_q > 0$ szükséges, ami csak úgy lehetséges, ha a kilépő változó β_p ($\equiv x_\mu$) korlátos típusú és a felső korláton lép ki a bázisból. Degenerciótól eltekintve ez azt jelenti, hogy d_q-nak és α_{pq}-nak azonos előjelűnek kell lenni.

3. Ha $v_p \neq 0$ és a kilépő változó 0 típusú (fix), akkor $\bar{d}_\mu$ előjele érdektelen. Ezért ha $v_p > 0$, akkor $t_q < 0$ hányadosokat keresünk, és ha $v_p < 0$, akkor pozitív t-ket vizsgálunk.

Hátra van még annak a vizsgálata, hogy a $\mathbf{v} = [v_1, \ldots, v_m]^T$ vektort hogyan lehet meghatározni. $\mathbf{v}$-t duál *első fázis redukált költség* vektornak nevezzük.

Vektor alakban (18) a következőképpen írható:

$$\mathbf{v} = \sum_{j \in \mathcal{M}} \boldsymbol{\alpha}_j - \sum_{j \in \mathcal{P}} \boldsymbol{\alpha}_j = \mathbf{B}^{-1}\Big(\sum_{j \in \mathcal{M}} \mathbf{a}_j - \sum_{j \in \mathcal{P}} \mathbf{a}_j \Big) = \mathbf{B}^{-1}\tilde{\mathbf{a}} \tag{20}$$

$\tilde{\mathbf{a}}$ nyilvánvaló értelmezése mellett. Ez a számítás egy FTRAN művelet segítségével végezhető el, ami egy standard szimplex technika.

3.2. Az $f(t)$ függvény vizsgálata

Miután a kilépő változót meghatároztuk (20) segítségével, a duál infízibilitások összege t függvényében:

$$f(t) = f(0) - t\Big(\sum_{j \in \mathcal{M}} \alpha_{pj} - \sum_{j \in \mathcal{P}} \alpha_{pj} \Big) = f(0) - tv_p. \tag{21}$$

Látható, hogy $f(t)$ előállításához az $\boldsymbol{\alpha}^p$ transzformált pivot sorra van szükség, amiből kiolvashatók az α_{pj} együtthatók. Ez számítástechnikailag egy költséges művelet.

Az előzményekből következően t elmozdulása nulláról lehet pozitív, illetve negatív irányban is, a helyzettől függően. (21) nyilvánvalóan lineáris függvény t-ben, amíg a $-v_p$ meredekséget definiáló halmazok ($\mathcal{M}$ és $\mathcal{P}$) változatlanok. Bebizonyítható (lásd Maros [6, 5]), hogy

> *$f(t)$ egy szakaszonként lineáris konkáv törtvonal függvény, melynek töréspontjai a belépő változó különböző megválasztásainak felelnek meg és amely pontokban az infízibilitási halmazok ($\mathcal{M}$ és/vagy $\mathcal{P}$) megváltoznak. Ennek megfelelően $f(t)$ a globális maximumát akkor éri el, amikor a meredeksége előjelet vált. Az így definiált báziscsere eredményezi a duál infízibilitások maximális javulását, amit a kiválasztott kilépő változóval el lehet érni.*

Ugyancsak bebizonyítható (lásd Maros [6, 5]), hogy a töréspontokat a következő $j \in \mathcal{R}$ pozíciók definiálják:

1. *A* $t \geq 0$ *esetben*
   $d_j < 0$ *és* $\alpha_{pj} < 0$ *vagy*
   $d_j \geq 0$ *és* $\alpha_{pj} > 0$,
2. *a* $t \leq 0$ *esetben pedig*
   $d_j < 0$ *és* $\alpha_{pj} > 0$ *vagy*
   $d_j \geq 0$ *és* $\alpha_{pj} < 0$.

A második eset közvetlenül származtatható az elsőből úgy, hogy $-\alpha_{pj}$*-t használunk* α_{pj} *helyett. Mindkét esetben egy további lehetőség az, hogy ha* $\text{type}(x_j) = 3$ *(szabad változó) és* $d_j \neq 0$*, akkor a* d_j/α_{pj}, $(\alpha_{pj} \neq 0)$ *töréspont multiplicitása* 2.

Ha a definiált töréspontokat nagyság szerint sorba rendezzük, akkor könnyen követhető, hogy $f(t)$ meredeksége akkor változik, amikor t eléri a legkisebb definiált hányadost (töréspontot). A rendezett értékek a $t \geq 0$ esetben $0 \leq t_1 \leq \cdots \leq t_Q$, ha Q-val jelöljük az összes definiált töréspontok számát, míg a $t \leq 0$ esetben fordított a sorrend $t_Q \leq \cdots \leq t_1 \leq 0$, vagy ezzel egyenértékűen, $0 \leq -t_1 \leq \cdots \leq -t_Q$, illetve a közös alak $0 \leq |t_1| \leq \cdots \leq |t_Q|$.

Ha sikeres volt p megválasztása (van kilépő) változó, akkor $Q > 0$. Bebizonyítható (lásd Maros [6, 5]), hogy

Akár a $v_p < 0$ $(t > 0)$*, akár a* $v_p > 0$ $(t < 0)$ *esetről van szó,* $f(t)$ *kezdeti meredeksége* $s_p^0 = |v_p|$*, és a lineáris szakasz meredeksége a* k*-adik töréspont után*

$$s_p^k = s_p^{k-1} - |\alpha_{pj_k}|\,, \text{ for } k = 1, \ldots, Q.$$

Ezen elméleti alapokon nyugszik a GDPO algoritmus, amelyet a következő szakaszban mutatunk be részletesen.

$f(t)$ jellegzetes alakja az 1. ábrán látható.

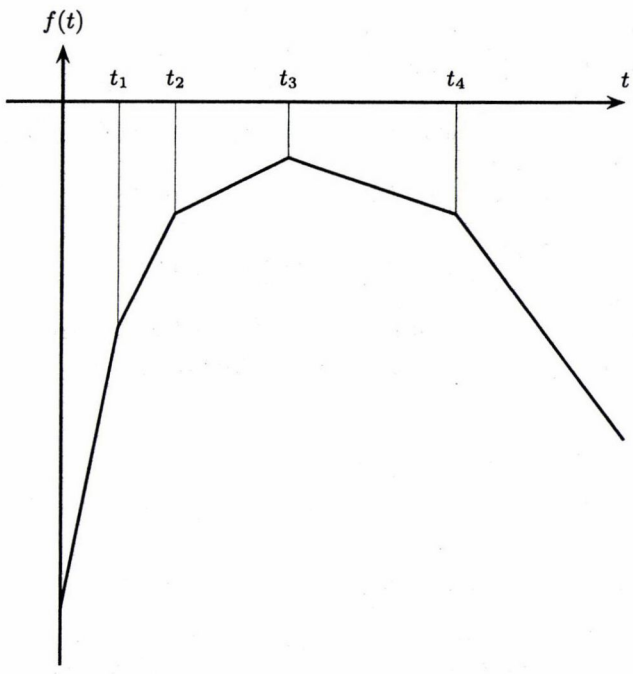
3.3. A GDPO algoritmus

Az alábbiakban a GDPO algoritmus egy iterációját definiáljuk a 3.1. és 3.2. szakaszokban tárgyaltakra alapozva.

Legyen $t_0 = 0$ és $f_i = f(t_i)$. Belátható, hogy a duál infízibiliások összege a töréspontokban a következő rekurzióval számítható ki: $f_k = f_{k-1} + s_p^{k-1}(t_k - t_{k-1})$, $k = 1, \ldots, Q$.

A duál első fázis egy iterációjának a lépései:

1. Lépés. Határozzuk meg az $\mathcal{M}$ és $\mathcal{P}$ halmazokat (13) és (14) alapján. Ha mindkettő üres, hajtsunk végre megengedettségi korrekciót. Ezután a bázis duál megengedett, az algoritmus befejeződik.



1. ábra. Duál infízibilitások mértéke (összege) t függvényében

2. Lépés. Állítsuk fel az $\tilde{\mathbf{a}} = \sum_{j \in \mathcal{M}} \mathbf{a}_j - \sum_{j \in \mathcal{P}} \mathbf{a}_j$ segédvektort (20) szerint.

3. Lépés. Határozzuk meg a duál redukált költségek vektorát: $\mathbf{v} = \mathbf{B}^{-1}\tilde{\mathbf{a}}$ (20) alapján.

4. Lépés. Kilépő vektor meghatározása: Válasszunk ki egy jelöltet a $\mathbf{v}$-ből valamilyen (egyszerű [pl. Dantzig], vagy összetett [pl. Devex]) szabály alapján. Jelöljük ezt a pozíciót p-vel. Ez lesz egyben a pivot sor indexe is.

Ha nem találunk jelöltet, akkor a duál feladatnak nincs megengedett megoldása, az algoritmus befejeződik.

5. Lépés. Határozzuk meg $\mathbf{B}^{-1}$ p-edik sorát: $\boldsymbol{\rho}^T = \mathbf{e}_p^T\mathbf{B}^{-1}$ és ennek segítségével $\mathbf{A}$ p-edik sorának transzformált alakját: $\alpha_{pj} = \boldsymbol{\rho}^T\mathbf{a}_j$ a $j \in \mathcal{R}$ pozíciókban.

6. Lépés. Duál hányadosteszt. Határozzuk meg és tároljuk a duál hányadosokat a 3.2. szakaszban leírtak alapján a $v_p > 0$, illetve $v_p < 0$ esetnek megfelelően. Rendezzük a hányadosokat (töréspontokat): $0 \leq |t_1| \leq \cdots \leq |t_Q|$.

7. Lépés. $f(t)$ maximalizálása.

Legyen $k = 0$, $t_0 = 0$, $f_0 = f(0)$, $s_p^0 = |v_p|$.
While $k < Q \wedge s_p^k > 0$ **do**
$k := k + 1$
Jelölje j_k azt az oszlopindexet, amelyik $|t_k|$-t, a k-adik legkisebb hányadost (töréspontot) határozta meg.
Noha algoritmikusan nem szükséges, a teljesség kedvéért számítsuk ki $f(t)$-t ebben a pontban: $f_k = f_{k-1} + s_p^{k-1}(t_k - t_{k-1})$,
Határozzuk meg $f(t)$ meredekségét ezután a pont után: $s_p^k = s_p^{k-1} - |\alpha_{pj_k}|$.
end while
Jelölje q az utolsó töréspontnak az indexét, amelyre a s_p^k meredekség még pozitív, $q = j_k$. $f(t)$ a maximumát ennél a töréspontnál éri el. A belépő változó x_q.

8. Lépés. Új megoldás meghatározása:

Ha x_μ felső korláton lép ki a bázisból ($v_p < 0$ eset), $\bar{x}_\mu = u_\mu$, különben ($v_p > 0$ eset) $\bar{x}_\mu = 0$.

Határozzuk meg a belépő oszlopvektor transzformált alakját: $\boldsymbol{\alpha}_q = \mathbf{B}^{-1}\mathbf{a}_q$.

Legyen $\theta_P = \beta_p/\alpha_{pq}$, ha $v_p > 0$, vagy $\theta_P = (\beta_p - \omega_p)/\alpha_{pq}$ (ω_p a p-edik bázisváltozó egyedi felső korlátja), ha $v_p < 0$. Transzformáljuk a megoldást:

$\bar{\beta}_p = x_q + \theta_P$ és $\bar{\beta}_i = \beta_i - \theta_P \alpha_{iq}$ minden $i \neq p$.

Transzformáljuk a bázison kívüli változókhoz tartozó duál logikai változókat:

$\bar{d}_\mu = -\dfrac{d_q}{\alpha_{pq}}$ és $\bar{d}_j = d_j + \bar{d}_\mu \alpha_{pj}$ minden $j \neq \mu$.

Módosítsuk a $\mathcal{B}$ és ha szükséges, az $\mathcal{U}$ halmazt, hogy tükrözzék a báziscserét.

Térjünk vissza az 1. Lépésre.

4. A GDPO algoritmus elméleti vizsgálata

Ebben a szakaszban először a GDPO algoritmus helyességét mutatjuk meg, majd néhány fontos tulajdonságát elemezzük.

4.1. Az algoritmus helyessége

Először is, a 3.2. szakaszban kiderült, hogy $f(t) = f(0) - t\Big(\sum_{j\in\mathcal{M}} \alpha_{pj} - \sum_{j\in\mathcal{P}} \alpha_{pj}\Big)$ egy szakaszonként lineáris konkáv törtvonal függvény, amit minden iteráció során újra definiálunk és maximalizálunk.

Másodszor, miután az $f(t)$ függvénynek 0 a felső korlátja, a GDPO során megoldott LP feladat *megoldása mindig korlátos.*

Harmadszor, $f(t)$ maximalizálása a duál megengedettségi feltételek mellett egy konvex probléma. Ezért ha nincs több javító sor és a megoldás még mindig nem duál megengedett, akkor feladatnak *nincs duál megengedett megoldása.*

Negyedszer, ha az algoritmus minden iterációban pozitív lépést tesz a duál megengedettség felé ($f(t)$ szigorúan monoton növekszik), akkor egyetlen bázis sem ismétlődhet, tehát ciklizálás nem fordulhat elő, így az *algoritmus véges számú lépés után befejeződik* vagy az 1., vagy a 4. Lépésben. Ha az iterációk során degeneráció lép fel, és a GDPO is csak degenerált lépést tudna végrehajtani (lásd még 4.2.4. szakaszt), akkor például az elméletileg garantált (és számítástechnikailag hatékony) Wolfe féle „ad hoc" módszert ([9]) lehet használni, amíg a degenerációból kikerül az algoritmus.

4.2. A GDPO néhány fontos tulajdonsága

4.2.1. A hagyományos eljárás általánosítása. A GDPO az $f(t)$ függvény maximalizálásával a lehető legnagyobb lépést teszi meg a duál megengedettség felé, amit a kiválasztott kilépő változó esetén el lehet érni. Könnyen látható, hogy a *hagyományos duál (HD) első fázis eljárás ennek speciális esete*, amikoris $f(t)$-nek csak az első töréspontjáig megyünk el. A másik oldalról nézve *GDPO a HD eljárás általánosításának* tekinthető.

4.2.2. A pivot sor többszörös hasznosítása. HD a „drágán" előállított transzformált pivot soron egyetlen iterációt végez el és utána eldobja azt. GDPO ezt a sort többszörösen hasznosítja, ami azt eredményezi, hogy egyetlen GDPO iterációval akár nagyon sok HD iterációnyi előrehaladást képes elérni. Mindez attól függ, hogy $f(t)$ maximumát hányadik töréspontnál éri el.

4.2.3. Nagyobb numerikus stabilitás. A szimplex implementációk numerikus problémáinak nagy része abból adódik, hogy abszolút értékben túl kicsinek adódik az α_{pq} pivot elem. GDPO esetén ezen könnyű segíteni, ha $f(t)$ maximuma nem az első töréspontón éretik el. Ekkor ugyanis lehetőség van az eggyel korábbi töréspontot venni. Ha az ehhez tartozó pivot elem még mindig nem megfelelő nagyságrendű, akkor fokozatosan lehet visszalépni a töréspontokon egy jó pivot elem megtalálásáig (ha egyáltalán van ilyen). Nagy valószínűséggel GDPO még ilyen visszalépéses esetben is nagyobb előrehaladást tud biztosítani, mint HD. Ezen túlmenően, ha az első töréspont pivot eleme „rossz", akkor HD nem is tud mit csinálni ezzel a pivot sorral, míg GDPO-nak van esélye egy jó iterációra, ha $f(t)$ a maximumát nem az első töréspontón éri el.

4.2.4. Jobb hatásfok degeneráció esetén. Duál degeneráció esetén egy vagy több bázison kívüli változóra $d_j = 0$. Ha ezek a pozíciók részt vesznek a duál hányados tesztben, akkor ezek 0 értékű hányadosokat definiálnak. Ez a (esetleg többszörös multiplicitással rendelkező) $t = 0$ töréspontot eredményezi. HD ezek közül választ egyet, és egy 0 lépéshosszú, nem-javító iterációt hajt végre.

Ezzel szemben GDPO esetén a helyzet a következő. Tegyük fel, hogy $t = 0$ multiplicitása ℓ, vagyis

$$0 = |t_1| = \cdots = |t_\ell| < |t_{\ell+1}| \leq \cdots \leq |t_Q|. \tag{22}$$

Jelöljék $\alpha_{j_1}, \ldots, \alpha_{j_\ell}, \ldots, \alpha_{j_Q}$ a megfelelő pozíciókban a transzformált pivot sor elemeit, az egyszerűség kedvéért elhagyva p-t a felső indexből. $f(t)$ maximumát definiáló töréspont k indexét az $s_k > 0$ és $s_{k+1} \leq 0$ relációk határozzák meg, ami részletesen:

$$s_k = s_0 - \sum_{i=1}^{k} |\alpha_{j_i}| > 0 \quad \text{és} \quad s_{k+1} = s_0 - \sum_{i=1}^{k+1} |\alpha_{j_i}| \leq 0.$$

Ha erre a k-ra $k > \ell$ teljesül, akkor (22) által azt kapjuk, hogy $|t_k| > 0$. Ez pedig azt jelenti, hogy a degeneráció ellenére *GDPO pozitív lépést tud tenni* a duál megengedettség felé. Ha $k \leq \ell$, akkor a lépés degenerált (0 lépéshosszú) lesz.

4.2.5. Egy iterációra eső számítási munka. GDPO egy iterációja valamivel, de általában nem sokkal több számítási munkát igényel, mint HD.

1. Hányados teszt: ugyanaz, mint a HD-nál. A különbség annyi, hogy GDPO esetén a hányadosokat ($f(t)$ töréspontjait) tárolni kell. Ez többlet memóriaigényt jelent, általában nem több, mint $n - m$ dupla pontosságú szám tárolását. Noha a type-3 pozíciók két töréspontot is meghatározhatnak, az ilyen változók száma kevés szokott lenni az LP modellekben.

2. Míg HD a legkisebb hányadost (első töréspontot) választja, addig GDPO számára a t_k töréspontokat nagyság szerint sorba kell rendezni. Ha sok töréspont definiálódik, akkor ez jelentős többletmunkát igényel, hiszen a rendezést minden iteráció során újból el kell végezni. Általában azonban nincs szükség az összes t_k-ra $f(t)$ maximumának a meghatározásánál. Így célszerű olyan rendezési módszert használni, amelyiknél az i-edik lépésben az első i elem helyes sorrendben van. A legmegfelelőbb rendezési módszer megválasztását nehezíti az, hogy a definiált és felhasznált töréspontok száma iterációról iterációra drasztikusan különböző lehet. Vizsgálataink azt mutatták, hogy a heap-sort módszer adaptációja megbízhatóan, jó hatékonysággal képes ezt a feladatot ellátni.

3. Tapasztalataink szerint a fenti többletmunka GDPO iterációs sebességét alig érzékelhetően befolyásolja HD-hez képest.

5. A GDPO algoritmus számítástechnikai vizsgálata

A GDPO tulajdonságainak elméleti vizsgálata során többször szerepelt olyan kitétel, hogy az igazán jó tulajdonságok akkor jönnek elő, ha valóban több töréspont definiálódik iterációnként és nem az elsőn éretik el $f(t)$ maximuma, mert így az algoritmusban rejlő nagyfokú rugalmasság jobban kihasználható. Az, hogy ilyen helyzet mennyire fordul elő, és ezáltal mekkora az algoritmus jelentősége, elsősorban gyakorlati tapasztalatok alapján derül ki. Ebben a szakaszban a szakmában elfogadott tesztfeladatokkal végzett számítások során nyert tapasztalatokról számolunk be. Miután GDPO egy lokális stratégiát valósít meg, elméleti úton nem lehet globális teljesítményt garantálni. Ezért is fontos a számítástechnikai vizsgálat elvégzése.

Az irodalomban nem ismeretes különféle duál szimplex algoritmusokkal kapcsolatos számítástechnikai tapasztalatok publikálása. Kommerciális LP rendszerekkel való összehasonlításnak nem lett volna értelme, mert azok algoritmikus háttere ismeretlen, publikálatlan üzleti titok, ezért algoritmikus összehasonlításra alkalmatlanok. Így GDPO-nak az első töréspontón alapuló hagyományos duál első fázissal való összehasonlításához folyamodtunk.

5.1. A tesztfeladatok jellemzői

A tesztelés célja GDPO algoritmikus *hatásosság*ának a vizsgálata. A tesztelésre a korábban a szerző által kifejlesztett, szimplex alapú MILP nevű kísérleti kódot használtuk. Noha MILP primál orientáltságú, tartalmazza azon eszközök nagy részét, amik a duál implementációjához szükségesek.

MILP a nagyméretű LP feladatok hatékony megoldására szolgáló algoritmikus és számítástechnikai elemek kipróbálására készült. Így ez egy kísérleti kód, amely tele van tűzdelve olyan utasításokkal, amelyek a működést figyelik és információt gyűjtenek. Ezáltal nagyon alkalmas új eljárások tulajdonságainak vizsgálatára, amikor is a hatásosság a kérdés. Mindezek mellett MILP működése igen hatékony, noha a „numerikus kernel"-je kb. 10–12 évvel ezelőtt készült (ami az akkori igényeket biztonságosan kielégítette). Mindezt azért szükséges megjegyezni, mert bár ez az esetek nagy részében jól ki tudta szolgálni a duál ismert nagyobb igényét a számítási pontosságra, de egy-két feladatnál ez nem bizonyult elegendőnek. Hangsúlyozni kell, hogy ez nem a GDPO algoritmikus hiányosságát jelenti, hiszen nem annak hibás működéséről volt szó, hanem pl. bizonyos mennyiségek kétféle kiszámítása közt adódó numerikus különbségről és az ebből esetleg bekövetkező kisebb visszaesésről. Noha MILP/Dual ilyenkor kijavítja a numerikus hibát néhány többlet iteráció árán, ez azonban eltorzítja az iterációs statisztikát. Az alább ismertetett tesztelésben olyan feladatok vettek részt, amelyek esetén MILP/Dual numerikus kernelje hibátlanul működött, mert így a különbségek feketén-fehéren a megoldó algoritmusok közti különbségből adódnak és ezáltal alkalmasak GDPO hatásosságának kiértékelésére. *A hatásosságot a duál első fázisban megtett iterációk számával*

mérjük. Ennek azért is van értelme, mert GDPO és HD iterációs sebessége alig mérhetően tér el egymástól.

Miután a type-0 és type-1 változók kezelése a duál első fázisban triviális (lásd duál megengedettségi korrekció), olyan feladatokat választottunk, amelyeknél a type-2 (nem-negatív) és type-3 (szabad) változók vannak túlsúlyban. Ezen túlmenően az is szempont, hogy a futási eredmények reprodukálhatók, illetve ellenőrizhetők legyenek. Ennek érdekében a használt feladatokat a szakmában elfogadott tesztfeladattárakból vettük. MILP két változatát használtuk a futások során. Az egyik (`m503d`) a GDPO, míg a másik (`m503d1`) a hagyományos duál (HD) implementálását tartalmazza. Az utóbbi valójában az előzőnek egy olyan változata, ahol az első töréspont alapján történik a báziscsere meghatározása (legkisebb duál hányados, $k = 1$). Összesen 48 feladat szerepel a vizsgálatokban. Köztük van kisebb, közepes és nagyméretű is. Ezek valós életből vett problémák, nem pedig generált feladatok.

Az 1. táblázat a tesztelésben résztvett feladatok főbb jellemzőit tartalmazza, amelyek: feltételek száma (m), strukturális változók száma ($\bar{n}$), nem-nullák száma **A**-ban, valamint a strukturális változók számának megoszlása típus szerint.

5.2. A tesztfutások eredményei és kiértékelése

Az eredményeket először összevont táblázatos formában mutatjuk be, majd a levont következtetéseknek megfelelő további táblázatok következnek.

A 2. táblázatban a duál első fázisban végzett iterációk számát mutatjuk be GDPO és HD esetén, továbbá a megoldás során használt stratégiát: volt-e normálás (általában igen), mi volt az induló bázis (logikai vagy crash), illetve a pivot sor meghatározása Devex technikával történt-e vagy sem. Nagy érdeklődésre tarthat számot az iteráció arányokat feltüntető két oszlop. A H/G jelentése az, hogy a HD algoritmus hányszor többet iterált, mint GDPO, a G/H oszlop ennek a reciproka. Látható, hogy nem minden feladat esetén voltak a futási paraméterek azonosak. Ennek több oka van. Először is, olyan induló bázist kellett választani, ami duál nem-megengedett. Bizonyos esetekben az egységvektorokból álló logikai bázisra ez teljesült, és amikor nem, akkor egy „crash" eljárással előállított bázis (lásd [5]) felelt meg a célnak. Másodszor, a nagyobb feladatok sorkiválasztására a duál Devex technikát használtuk a racionálisabb megoldási idő érdekében. Harmadszor, a feladatok legtöbbjét normálással oldottuk meg, hogy numerikus problémák ne lépjenek fel és az eredmények „tisztán" legyenek értelmezhetők az algoritmusok hatásossága szempontjából. Nagyon fontos megjegyezni, hogy minden konkrét feladat esetén ugyanazzal a stratégiával futott mindkét (`m503d` és `m503d1`) program. Ezért a megoldási stratégia gyanánt ezek a közös beállítások vannak feltüntetve.

A GDPO elvárt hatásos működésének az alapja a pivot sor többszörös felhasználása, ami abban nyilvánul meg, hogy $f(t)$ maximalizálására hány töréspontot használ fel. Ezt egy bizonyos értelemben *algoritmikus lépéshossz*nak lehet nevezni. A 3. táblázat erről ad képet. Miután nagyon nagy a variáció, az adatokat kissé tömöríteni kellett, hogy be lehessen mutatni. A táblázat egy sora azt mondja meg, hogy az összes duál első fázis iteráció során (ez a szám az utolsó oszlopban van)

Feladat	Sorok	Oszlopok	Nem-nullák	Változók száma típus szerint			
				Type-0	Type-1	Type-2	Type-3
25fv47	822	1571	11127	0	0	1571	0
80bau3b	2262	9799	29063	498	2986	6315	0
agg	488	163	2541	0	0	163	0
agg2	516	302	4515	0	0	302	0
agg3	516	302	4531	0	0	302	0
baxter	27441	15128	109823	0	1122	14006	0
bnl1	643	1175	6129	0	0	1175	0
bnl2	2325	3489	16124	0	0	3489	0
boeing1	351	384	3865	0	228	156	0
cre_a	3516	4067	19054	0	0	4067	0
cre_b	9648	72447	328542	0	0	72447	0
cre_c	3068	3678	16922	0	0	3678	0
cre_d	8926	69980	312626	0	0	69980	0
czprob	929	3523	14173	229	0	3294	0
d6cube	415	6184	43888	0	0	6184	0
dbir2	18906	27355	1148847	0	0	27355	0
degen2	444	534	4449	0	0	534	0
degen3	1504	1818	26230	0	0	1818	0
degen4	4420	6711	107375	0	0	6711	0
grow07	140	301	2633	0	280	21	0
grow15	300	645	5665	0	600	45	0
grow22	440	946	8318	0	880	66	0
israel	174	142	2358	0	0	142	0
maros	847	1443	10006	35	0	1408	0
mod011	4481	10958	37425	1	1596	9361	0
nsct2	23003	14981	686396	0	0	14981	0
osa-07	1118	23949	167643	0	0	23949	0
osa-14	2337	52460	367220	0	0	52460	0
osa-30	4350	100024	700160	0	0	100024	0
perold	625	1376	6026	64	266	958	88
pilot_we	723	2789	9218	78	294	2337	80
rentacar	6804	9557	42019	650	179	8728	0
scagr_2	32847	34580	141757	0	0	34580	0
scrs_3	16545	17420	71401	0	0	17420	0
scsd6	147	1350	5666	0	0	1350	0
scsd8	397	2750	11334	0	0	2750	0
sctap2	1090	1880	8124	0	0	1880	0
sctap3	1480	2480	19734	0	0	2480	0
ship08l	778	4283	17085	0	0	4283	0
ship12l	1151	5427	21597	0	0	5427	0
stair	357	467	3857	82	6	373	6
sto27	14441	34114	114973	0	0	34114	0
stocfor2	2157	2031	9492	0	0	2031	0
stocfor3	16676	15695	74004	0	0	15695	0
sws	14310	12465	105480	0	0	12465	0
unicolns	5421	45569	168220	2	1449	44118	0
woodlp	244	2594	70216	0	0	2594	0
woodw	1098	8405	37478	0	0	8405	0

1. táblázat

Feladat	Induló duál inf. száma	Duál Ph-1 it. száma		Iteráció arányok		Megoldási stratégia		
		GDPO	HD ($k=1$)	H/G	G/H	Normálás I/N	Ind. bázis CB/LB	Devex I/N
25fv47	41	238	1033	4.34	0.23	I	LB	N
80bau3b	208	736	997	1.35	0.74	I	LB	I
agg	95	11	107	9.73	0.10	I	LB	N
agg2	171	13	109	8.38	0.12	I	LB	N
agg3	171	13	109	8.38	0.12	I	LB	N
baxter	3259	2687	4482	1.67	0.60	I	CB	I
bnl1	57	4	27	6.75	0.15	I	LB	I
bnl2	156	49	134	2.73	0.36	I	LB	I
boeing1	164	9	102	11.33	0.09	I	LB	N
cre_a	1156	476	1655	3.48	0.29	N	CB	I
cre_b	14503	4604	12281	2.67	0.37	N	CB	I
cre_c	1056	532	1559	2.93	0.34	N	CB	I
cre_d	10409	3479	14798	4.25	0.23	N	CB	I
czprob	1521	191	1959	10.26	0.10	I	LB	N
d6cube	2637	1209	6500	5.38	0.19	I	CB	I
dbir2	9210	9631	9848	1.02	0.98	I	LB	I
degen2	425	143	574	4.01	0.25	I	LB	I
degen3	1249	571	1945	3.41	0.29	I	LB	I
degen4	2697	1177	6792	5.77	0.17	I	LB	I
grow07	21	1	14	14.00	0.07	I	CB	N
grow15	45	1	14	14.00	0.07	I	CB	N
grow22	66	1	14	14.00	0.07	I	CB	N
israel	24	1	24	24.00	0.04	I	LB	N
maros	162	666	799	1.20	0.83	I	LB	N
mod011	4343	602	3753	6.23	0.16	I	CB	I
nsct2	11240	11507	11636	1.01	0.99	I	LB	I
osa-07	9201	114	3456	30.32	0.03	I	CB	I
osa-14	19695	141	3616	25.65	0.04	I	CB	I
osa-30	37495	68	7785	114.49	0.01	I	CB	I
perold	7	725	587	0.81	1.24	I	LB	N
pilot_we	91	580	1065	1.84	0.54	I	LB	N
rentacar	2	1778	1629	0.92	1.09	I	LB	N
scagr_2	8645	16676	27473	1.65	0.61	I	LB	I
scrs_3	4355	11635	12765	1.10	0.91	I	LB	I
scsd6	218	46	147	3.20	0.31	I	CB	N
scsd8	353	6	30	5.00	0.20	I	CB	N
sctap2	238	442	578	1.31	0.76	I	CB	I
sctap3	315	532	714	1.34	0.75	I	CB	I
ship08l	581	39	587	15.05	0.07	I	CB	N
ship12l	708	51	958	18.78	0.05	I	CB	N
stair	1	180	152	0.84	1.18	I	LB	N
sto27	11541	4879	6844	1.40	0.71	I	CB	I
stocfor2	639	1366	1552	1.14	0.88	I	LB	N
stocfor3	5077	10620	11848	1.12	0.90	I	LB	N
sws	2190	988	1694	1.71	0.58	I	CB	I
unicolns	43914	4975	50841	10.22	0.10	I	CB	I
woodlp	1057	30	1288	42.93	0.02	I	CB	N
woodw	1738	60	2520	42.00	0.02	I	CB	N

2. táblázat

	Felhasznált töréspontok száma										Iterációk
Feladat	1	2	3	4	5	6–10	11–20	21–50	50+	Max	Ph1-ben
25fv47	80	76	38	16	7	17	4	–	–	18	238
80bau3b	389	165	66	41	29	41	5	–	–	14	736
agg	–	1	–	2	–	6	2	–	–	18	11
agg2	1	1	–	3	1	4	–	3	–	48	13
agg3	1	1	–	3	1	4	–	3	–	48	13
baxter	1381	331	157	142	74	201	188	178	35	194	2687
bnl1	–	–	–	–	–	–	4	–	–	17	4
bnl2	17	18	4	–	–	–	10	–	–	19	49
boeing1	1	–	–	–	2	3	2	–	1	136	9
cre_a	79	62	39	27	30	67	79	57	36	474	476
cre_b	103	179	169	203	171	702	287	875	1915	3538	4604
cre_c	102	93	62	49	20	89	49	47	21	397	532
cre_d	108	105	151	109	133	548	656	733	936	3948	3479
czprob	46	71	32	12	8	5	2	4	11	172	191
d6cube	103	90	83	81	92	300	231	139	90	821	1209
dbir2	8851	587	130	34	22	6	1	–	–	13	9631
degen2	19	34	64	7	5	11	–	1	2	58	143
degen3	128	131	247	23	11	14	11	3	3	91	571
degen4	312	165	189	109	111	185	90	10	6	162	1177
grow07	–	–	–	–	–	–	–	1	–	21	1
grow15	–	–	–	–	–	–	–	1	–	45	1
grow22	–	–	–	–	–	–	–	–	1	66	1
israel	–	–	–	–	–	–	–	1	–	24	1
maros	258	200	97	44	24	34	8	1	–	32	666
mod011	260	107	64	46	31	52	17	10	15	917	602
nsct2	10974	298	85	35	19	59	31	6	–	26	11507
osa-07	37	24	3	6	1	2	4	5	32	4503	114
osa-14	70	18	4	2	1	2	3	6	39	9454	145
osa-30	18	3	4	4	2	2	1	6	39	3654	79
perold	414	156	47	18	21	38	15	14	2	154	725
pilot_we	341	84	39	16	11	42	36	9	2	103	580
rentacar	1737	33	6	–	–	1	1	–	–	11	1778
scagr_2	10625	5186	–	865	–	–	–	–	–	4	16676
scrs_3	8311	2995	298	23	8	–	–	–	–	5	11635
scsd6	2	2	4	2	2	6	14	10	4	100	46
scsd8	–	–	–	1	–	1	–	1	3	260	6
sctap2	65	98	58	64	26	83	28	17	3	55	442
sctap3	103	124	46	59	39	87	39	32	3	90	532
ship08	16	2	–	1	12	–	–	–	8	73	39
ship12	13	6	9	9	1	1	–	–	12	62	51
stair	147	30	1	–	1	–	1	–	–	14	180
sto27	612	725	916	520	484	1006	424	191	1	56	4879
stocfor2	583	450	179	99	30	25	–	–	–	10	1366
stocfor3	3730	3491	1604	761	443	546	44	1	–	21	10620
sws	367	332	12	85	31	71	70	–	–	17	988
unicolns	441	627	225	359	96	348	2659	220	–	34	4975
wood1p	–	–	–	–	–	–	5	8	17	588	30
woodw	–	2	–	2	4	7	7	15	23	801	60

3. táblázat

hányszor maximalizálta $f(t)$-t az első, a második, ..., az ötödik töréspont, illetve hányszor volt a maximalizáló töréspont 6 és 10, 11 és 20, 21 és 50 között, illetve 50-nél több. Az összevont része a táblázatnak sok érdekes esetet eltakar (főleg az 50+ részben). Ennek részbeni ellensúlyozására szerepel a „Max" feliratú oszlop, amelyik azt tünteti fel, hogy az összes iteráció során mennyi volt az egy lépésben felhasznált töréspontok maximális száma. Ha például megnézzük a `mod011` sorát, akkor azt látjuk, hogy az első fázisban megtett 602 iterációból (utolsó oszlop) 31 esetben fordult elő, hogy $f(t)$ az ötödik töréspontnál érte el a maximumát, továbbá volt olyan eset (Max), amikor a 917. töréspontig kellett elmenni a maximum eléréséhez.

A kiértékelés első szempontja az algoritmus helyességének az igazolása. Noha ez elméleti úton már megtörtént a 4.1. szakaszban, a kísérletek során szerzett tapasztalatok is azt mutatják, hogy GDPO képes helyesen megoldani a feladatokat.

GDPO hatásosságnak a kiértékeléséhez a 2. táblázatból indulunk ki. Már az első átnézés alapján látszik, hogy GDPO három eset kivételével jobb, mint HD. A H/G oszlop azt mutatja, hogy HD hányszor több iterációt végzett a duál megengedettség eléréséig, mint GDPO. Az 1-nél nagyobb számok jelzik azokat az eseteket, amikor GDPO volt a jobb, és ez a szám egyben a hatásosság mérőszáma is. Három esetben ez a szám 1-nél valamivel kisebb, jelezve, hogy ekkor HD volt a hatásosabb. A jobb áttekinthetőség érdekében elkészítettük a 4. sz. (tömörített) táblázatot annak a bemutatására, hogy GDPO hány esetben és hányszorosan volt jobb HD-nél.

Optimalizáló algoritmusok esetén 25% javulást általában jelentősnek szoktak minősíteni. Ha GDPO esetén csak a legalább 50%-os javulást tekintjük, akkor azt látjuk, hogy a 48 feladatból 35 esik ebbe a kategóriába (lásd 4. sz. táblázat). Feltűnő, hogy ezen belül 13 esetben a hatásosság több mint 10-szeresére növekedett. Különösen az `osa` feladatcsalád megoldása látszik kiemelkedő teljesítménynek, ahol a legjobb esetben (`osa-30`) a javulás 114-szeres.

Az első töréspontot használó algoritmusok (mint amilyen HD is) a duál nem-megengedettségek számát általában csak egyesével tudják csökkenteni, kivéve amikor duál degeneráció miatt ennél többet sikerül egy lépésben elérni. Bár GDPO csak a nem-megengedettségek összegében monoton, mégis képes a nem-megengedettségek számát is igen gyorsan csökkenteni. Az ebből a szempontból kiemelkedően jó példákat az 5. sz. táblázatban foglaltuk össze (összesen 20 feladat).

A 3. sz. táblázat azt tanúsítja, hogy $f(t)$ aktívan használja a definiált töréspontokat. Sőt, ennél több is kiderül. Elméletileg a lehető legjobb teljesítmény az, ha az összes nem-megengedettséget egyetlen iterációval ki lehet küszöbölni. A táblázatból látható, hogy valódi feladatokon GDPO ezt el is tudja érni. Ezek a feladatok: a `grow` család (`grow07`, `grow15` és `grow22`), valamint `israel`. A `grow` feladatokban viszonylag kevés type-2 változó van (21, 45 és 66), azonban az ezekhez tartozó összes duál logikai változó duál nem-megengedett a crash bázis esetén. Az első iterációban definiált $f(t)$ maximumát az összes (21, 45 és 66 [annyi, mint a type-2 változók száma]) definiált töréspont felhasználásával érte el, és ezáltal mindegyik duál logikai változó megengedett értékre került egyetlen lépésben. `israel` esetén az összes változó 2-es típusú, de itt is a maximális hatásossággal működött GDPO.

Javulás	Hány esetben
1.0 – 1.5-szeres	10
1.6 – 3.0-szoros	7
3.1 – 5.0-szörös	7
5.1 – 10.0-szeres	8
Több mint 10-szeres	13
Romlás	
0.8 – 1.0-szeres	3
Összes	48

4. táblázat. GDPO hatásossága az iterációk számának arányában mérve

Feladat	Induló duál inf. száma	GDPO iterációk
agg	95	11
agg2	171	13
agg3	171	13
bnl1	57	4
boeing1	164	9
grow07	21	1
grow15	45	1
grow22	66	1
israel	24	1
mod011	4343	602
osa-07	9201	114
osa-14	19695	141
osa-30	37495	68
scsd6	218	46
scsd8	353	6
ship08l	581	39
ship12l	708	51
unicolns	43914	4975
wood1p	1057	30
woodw	1738	60

5. táblázat. Duál infízibilitások számának különösen gyors eliminálása

6. Következtetések

A cikk célja a szimplex módszerre kidolgozott és GDPO-nak nevezett duál első fázis algoritmus [6, 5] elméleti és számítástechnikai elemzése, az algoritmus tulajdonságainak vizsgálata volt. Először magát az algoritmust mutattuk be röviden, hogy az elméleti tulajdonságok tárgyalása önmagában is érthető legyen. Ezután tértünk rá a számítástechnikai vizsgálatra, amit intenzív kísérleti munka előzött meg. A kísérleteket a nemzetközileg elfogadott valódi életből vett tesztfeladatokon végeztük el. Ezek méretben és komplexitásban széles kört ölelnek fel. GDPO-t összehasonlítottuk egy tipikus hagyományos duál első fázis algoritmussal. Kommerciális LP rendszerekkel való összehasonlításnak nem lett volna értelme a korábban már említett okok miatt.

A GDPO-val kapcsolatos elméleti és számítástechnikai vizsgálatok tapasztalatait az alábbiakban lehet összefoglalni.

1. GDPO a hagyományos duál első fázis algoritmusokat speciális esetként tartalmazza, és így azok általánosításának tekinthető.

2. GDPO a transzformált pivot sort többszörösen képes használni, ami által egy iterációban sok hagyományos iterációnak megfelelő előrehaladást képes elérni.

3. GDPO csak az infízibilitások összegében monoton, ami nagy rugalmasságot biztosít és lehetővé teszi megfelelő nagyságrendű pivot elem kiválasztását. Ez pedig kedvező numerikus tulajdonságot eredményez.

4. Duál degeneráció esetén GDPO-nak sokkal nagyobb esélye van nem-degenerált iterációt végrehajtani, mint más duál algoritmusoknak.

5. GDPO jól implementálható, és az iterációs sebesség alig érzékelhetően változik HD-vel szemben, ha a számítástudomány korszerű módszereit használjuk.

6. GDPO előnyös elméleti tulajdonságai a gyakorlatban rendszeresen érvényesülnek.

7. GDPO hatásosság tekintetében szinte mindig jelentősen felülmúlja a hagyományos duál első fázis módszert. Több valós esetben képes az elméleti maximumot is nyújtani, vagyis duál megengedetté tenni a megoldást egyetlen nem triviális iterációval.

8. GDPO kedvező működésének egyértelműen az a magyarázata, hogy minden lépésben a maximális előrehaladást valósítja meg, ami egy kiválasztott kilépő változó esetén lehetséges, és ami csak (akár nagyon) sok normál duál szimplex iterációval lenne elérhető. Sok töréspont felhasználása esetén ez hatalmas különbséget jelent.

Fentiek alapján levonható az a következtetés, hogy GDPO elméleti és gyakorlati szempontból egyaránt jelentős algoritmus, aminek feltétlen helye van a duál szimplex algoritmus korszerű eszköztárában.

Hivatkozások

[1] Dantzig, G. B., Maximization of a linear function of variables subject to linear inequalities, in: Koopmans, T. C. (ed.), *Activity analysis of production and allocation*, 339–347 (Wiley, New York, 1951).

[2] Fourer, R., Notes on the Dual Simplex Method, unpublished (1994, March).

[3] Karmarkar, N., A New Polynomial-Time Algorithm for Linear Programming, *Combinatorica* **4** (1984), 373–395.

[4] Maros, I., A Piecewise Linear Dual Procedure in Mixed Integer Programming, in: Giannesi, F., Komlósi, S. and Rapcsák, T. (eds.), *New Trends in Mathematical Programming*, 159–170 (Kluwer Academic Publishers, Boston, 1998).

[5] Maros, I., *Computational Techniques of the Simplex Method*, International Series in Operations Research and Management 61 (Kluwer Academic Publishers, Boston, 2003).

[6] Maros, I., A Piecewise Linear Dual Phase-1 Algorithm for the Simplex Method, *Computational Optimization and Applications* **26** (2003), 63–81.

[7] Roos, C., Terlaky, T. and Vial, J.-Ph., *Theory and Algorithms for Linear Optimization*, Discrete Mathematics and Optimization (Wiley, 1997).

[8] Terlaky, T. (ed.), *Interior Point Methods of Mathematical Programming*, Applied Optimization 5 (Kluwer Academic Publishers, Boston, 1996).

[9] Wolfe, Ph., A technique for resolving degeneracy in linear programming, *SIAM Journal of Applied Mathematics*, **11** (1963), 205–211.

(Beérkezett: 2005. július 22.)

MAROS ISTVÁN
DEPARTMENT OF COMPUTING
IMPERIAL COLLEGE
LONDON
i.maros@imperial.ac.uk

INVESTIGATING PHASE 1 OF THE DUAL SIMPLEX

ISTVÁN MAROS

The paper performs a theoretical and computational analysis of a new dual simplex algorithm GDPO that is based on a piecewise linear phase 1 objective function. It concludes that it is able to considerably outperform the traditional dual phase 1 methods. It offers enhanced numerically stability and more effectiveness in coping with degeneracy. Tests on 48 real life problems indicate that the theoretically possible improvements are very likely to materialize in practice thus making this algorithm a prime candidate for inclusion in any modern simplex solver.