sciendo

Műszaki Tudományos Közlemények vol. 11. (2019) 47–50.
https://doi.org/10.33895/mtk-2019.11.08
https://eda.eme.ro/handle/10598/31245
Angol nyelven: https://doi.org/10.33894/mtk-2019.11.08

EME MTSZ MTK

LINEÁRIS KOMPLEMENTARITÁSI FELADATRA VONATKOZÓ, SZÉLES KÖRNYEZETBEN MŰKÖDŐ, BELSŐPONTOS ALGORITMUS IMPLEMENTÁLÁSA

IMPLEMENTATION OF AN INTERIOR-POINT ALGORITHM FOR LINEAR COMPLEMENTARITY PROBLEM WORKING IN A WIDE NEIGHBORHOOD

Darvay Zsolt[1], Orbán Attila-Szabolcs[2]

Babeș-Bolyai Tudományegyetem, Matematika és Informatika Kar, Kolozsvár, Románia

[1] *darvay@cs.ubbcluj.ro*

[2] *orban.attila@yahoo.com*

Abstract

In this article, we study the interior-point algorithm for solving linear complementarity problems, published by Xiaouje Ma, Hongwei Liu, Jianke Zhang and Weijie Cong from the implementation point of view. The algorithm was implemented in C++ programming language, thus supporting the effectiveness of the method. Despite of the fact that the theoretical results refer only to the monotone linear complementarity problem, the practical testing showed that the algorithm also works well in more general cases.

Keywords*: interior-point algorithm, linear comlementarity problem, wide neighborhood, path-following algorithm, object-oriented technique.*

Összefoglalás

A cikkben a Xiaouje Ma, Hongwei Liu, Jianke Zhang és Weijie Cong által publikált lineáris komplementaritási feladatok megoldására vonatkozó útkövető belsőpontos algoritmust vizsgáljuk az implementáció szemszögéből nézve. Az algoritmust C++ programozási nyelvben implementáltuk, ezáltal alátámasztva a módszer hatékonyságát. Annak ellenére, hogy az elméleti eredmények csak monoton lineáris komplementaritási feladatra vonatkoznak, a gyakorlati tesztelés során arra is fény derült, hogy általánosabb esetekben is jól működik az algoritmus.

Kulcsszavak*: belsőpontos algoritmus, lineáris komplementaritási feladat, széles környezet, útkövető algoritmus, objektumorientált programozás.*

1. Bevezető

A lineáris optimalizálási feladatok megoldására különböző belsőpontos algoritmusokat vezettek be [1,2,3], melyek hatékonynak bizonyultak. Az egyes módszereket lineáris komplementaritási feladatra (LCP) is sikerül általánosítani [4], melyeket gyakran használnak mérnöki problémák megoldására. Az útkövető belsőpontos algoritmusok körében rövid, illetve hosszú lépéses változatokat különböztetünk meg. Mint kiderült, a rövid lépéses algoritmusok elméleti hatékonysága általában jobb, de a hosszú lépésesek a gyakorlatban jobban teljesítenek. Ai [5] vezette be az első lineáris optimalizálásra vonatkozó hosszú lépéses algoritmust (APF), amelynek az elméleti bonyolultsága megegyezik a legjobb rövid lépésesekével. Ai és Zhang [6] általánosította az Ai módszerét LCP-re. Ma, Liu, Zhang és Cong [7] úgy terjesztette

ki a fenti algoritmust (APF+), hogy két különböző lépést vezetett be. Az elsőt gyors lépésnek a másodikat biztonságos lépésnek nevezték el. Mindkettő az Ai által megadott széles környezetben dolgozik, de a gyors lépés esetén a vizsgált környezetet módosítjuk a paraméterek megváltoztatása által. A továbbiakban ezt az algoritmust vizsgáljuk, és tárgyaljuk az implementálásának a lehetőségeit.

2. A feladat leírása

Az LCP alakja:

$$\begin{cases} s = Mx + q, \\ x \geq 0, s \geq 0, x^T s = 0, \end{cases}$$

ahol $q \in \mathbb{R}^n$ és $M \in \mathbb{R}^{n\times n}$. Továbbá, feltételezzük, hogy az $M+M^T$ mátrix pozitív szemidefinit, tehát $x^T Mx \geq 0$ bármely $x \in \mathbb{R}^n$ esetén. Az M mátrix és a q vektor adottak, valamint az x és az s ismeretlenek.

A primál-duál belsőpontos algoritmusok igen hatékony módszernek bizonyultak az LCP-k megoldására. Ezen algoritmusok esetén feltételezzük, hogy az LCP megengedett megoldásainak halmaza $\mathcal{F}_{++}$ nem üres:

$$\mathcal{F}_{++} := \{(x,s) | s = Mx + q, \quad x > 0, \quad s > 0\}.$$

A centrális út, melyet a következőképpen definiálunk egy alapvető fogalom a primál-duál belsőpontos algoritmusok esetén:

$$C := \{(x,s) \in \mathcal{F}_{++} | xs = \mu e, \mu > 0\},$$

ahol xs jelöli az $x \in \mathbb{R}^n$ és $s \in \mathbb{R}^n$ elemenkénti szorzatát.

További jelölések: az $x \in \mathbb{R}^n$ vektor i-edik elemét x_i-vel jelöljük; e egy n-dimenziós vektor, amelynek az összes eleme 1; bármely $a \in \mathbb{R}^n$ számra $a^+ := \max\{a, 0\}$ és $a^- := \min\{a, 0\}$; $\|x\|$ az l_2 norma, $\|x\|_1$ pedig az l_1 norma. Az a^+ és a^- jelöléseket vektorra is kiterjesztjük.

3. A módosított algoritmus

Ai a következő módon definiálta az APF algoritmus számára a széles környezetet:

$$\mathcal{N}(\tau,\beta) := \{(x,s) \in \mathcal{F}_{++} \mid \|(\tau\mu e - xs)^+\|_1 \leq \beta\tau\mu \}$$

ahol $0 < \beta < 1$ és $\mu = (x^T s)/n$. Az APF+ algoritmushoz be kell vezetnünk a következőket:

$$\theta \in \left(0, \frac{1}{2}\right], \kappa \in (0,\theta), \beta \in [\beta_0, \beta_{max}],$$
$$0 < \beta_0 < \beta_{max} \leq \frac{1}{3}.$$

Feltételezzük, hogy adott az aktuális vektorokból alkotott (x,s) pár, mely eleme az $\mathcal{N}(\tau, \beta)$ környezetnek. Alkalmazunk egy gyors lépést, melyhez megoldjuk a következő egyenletrendszert:

$$\begin{cases} \Delta s^a = M\Delta x^a, \\ s\Delta x^a + x\Delta s^a = -xs. \end{cases} \quad (1)$$

A Δx^a és Δs^a meghatározása után úgy igyekszünk meghatározni az α lépéshosszt, hogy a kapott $x(\alpha) = x + \alpha\Delta x^a$ és $s(\alpha) = s + \alpha\Delta s^a$ vektorok az $\mathcal{N}(\tau, \theta\beta + (1 - \theta)\beta_{max})$ szélesebb környezetben legyenek. Tulajdonképpen a β paraméter értékét kissé megnöveljük. Ez az egyik sajátossága az APF+ algoritmusnak. A normalizált dualitási rés kiszámolása $\mu(\alpha)=(x(\alpha)^T s(\alpha)) / n$ összefüggés alapján történik.

A [7] cikkben a szerzők egy κ küszöbértéket adnak meg és a $\mu(\alpha) \leq \kappa\mu$ egyenlőtlenség segítségével döntik el, hogy szükség van-e biztonságos lépésre. Ha az egyenlőtlenség teljesül akkor a β paraméter új értéke $\theta\beta + (1 - \theta)\beta_{max}$ lesz, ellenkező esetben egy biztonságos lépést végzünk úgy, hogy a kapott pontok az eredeti $N(\tau,\beta)$ környezetben legyenek. Biztonságos lépés esetében megoldjuk a következő egyenlet-rendszert:

$$\begin{cases} \Delta s^p = M\Delta x^p, \\ s\Delta x^p + x\Delta s^p = \lambda(\tau\mu e - xs)^- + (\tau\mu e - xs)^+, \end{cases} \quad (2)$$

ahol a $\lambda = \dfrac{\|(\tau\mu e - xs)^+\|_1}{\|(\tau\mu e - xs)^-\|_1}$,

tehát $\lambda e^T(\tau\mu e - xs)^- + e^T(\tau\mu e - xs)^+ = 0$. Azonban a meghatározott $(\Delta x^p, \Delta s^p)$ pár nem adja meg a biztonságos lépés irányát, ebbe bele kell vennünk még a $(\Delta x^a, \Delta s^a)$ -t is. A [7] cikktől eltérően nem a $(\Delta x^a, \Delta s^a)$, illetve $(\Delta x^p, \Delta s^p)$ irányok lineáris kombinációjával dolgoztunk, hanem bevezettünk egy $0 < \gamma < 1$ állandót, amellyel a biztonságos lépést súlyozzuk. Tehát az α hosszúságú lépés az alábbi pontokat eredményezi:

$x(\alpha) = x + \alpha(\gamma\Delta x^p + \Delta x^a)$ és
$s(\alpha) = s + \alpha(\gamma\Delta s^p + \Delta s^a)$.

4. Az algoritmus

A továbbiakban a [7]-ben publikált algoritmus γ paraméterrel módosított változatát mutatjuk be.

Bemeneti paraméterek: kívánt pontosság $\varepsilon > 0$, $0 < \beta_0 < \beta_{max} \leq 1/3$, $\tau < 1/2$, $0 < \theta \leq 1/2$, $0 < \kappa < \theta$, $\gamma \in (0,1)$ és a kezdeti pont $(x^0, s^0) \in \mathcal{N}(\tau, \beta_0)$.

$(x, s) = (x^0, s^0)$; $\beta = \beta_0$;

while $x^T s > \varepsilon$ **do begin**
Meghatározzuk $(\Delta x^a, \Delta s^a)$-t az (1) alapján.
gyors lépés:
α = alphaFast($x, s, \Delta x^a, \Delta s^a, \beta, \beta_{max}$)
$x(\alpha) = x + \alpha \Delta x^a$; $s(\alpha) = s + \alpha \Delta s^a$;
$\mu(\alpha) = (x(\alpha)^T s(\alpha)) / n$;
if $\mu(\alpha) \leq \kappa\mu$ **then**
$\beta = \theta\beta + (1 - \theta)\beta_{max}$;
else begin
biztonságos lépés:
Meghatározzuk $(\Delta x^p, \Delta s^p)$ -t a (2) alapján.
α = alphaSafe(x, s, $\Delta x^a, \Delta s^a, \Delta x^p, \Delta s^p, \beta$);
$x(\alpha) = x + \alpha(\gamma \Delta x^p + \Delta x^a)$;
$s(\alpha) = s + \alpha(\gamma \Delta s^p + \Delta s^a)$;
$\mu(\alpha) = (x(\alpha)^T s(\alpha)) / n$;
end if
$(x, s) = (x(\alpha), s(\alpha))$; $\mu = \mu(\alpha)$;
end.

Megjegyezzük, hogy az alphaFast, illetve alphaSafe függvények úgy határozzák meg a lépés hosszúságát, hogy a kapott $(x(\alpha), s(\alpha))$ vektorpár az $\mathcal{N}(\tau, \theta\beta + (1 - \theta)\beta_{max})$, illetve $\mathcal{N}(\tau, \beta)$ környezetben legyen.

5. Implementálás

Az implementálás C++ programozási nyelvben történt, Visual Studio fejlesztői környezetben és alapjául veszi a [8] dolgozatban bevezetett kódot.

A megvalósítás során mindkét lépés esetén szembesültünk azzal a kérdéssel, hogy miként lehet meghatározni a lépés maximális hosszúságát úgy, hogy a kapott pontok a széles környezetben maradjanak.

A legjobb α lépéshossz a
$\min \mu(\alpha)$, $(x(\alpha), s(\alpha)) \in \mathcal{N}(\tau,\beta)$, $0 < \alpha \leq 1$,
optimalizálási feladat megoldása által adható meg az algoritmus minden lépésében [7].

E helyett a következőképpen jártunk el: előbb kiszámoltuk azt a maximális hosszúságú lépést, amely még nem sérti meg a megengedettségi feltételeket. Ezt követően ellenőriztük, hogy a kapott pontok benne vannak-e a környezetben. Amennyiben a feltétel nem teljesült feleztük a lépést és újra leellenőriztük azt, hogy a pontok benne vannak-e a környezetben. Mindezt addig ismételtük, ameddig a megfelelő hosszúságú lépést meg nem kaptuk.

6. Numerikus eredmények

Az algoritmust két monoton LCP-re teszteltük. Az első a következő (lcp01, lásd [9]):

$$M = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 0 & 1 \\ 1 & 0 & 1 & 2 \\ -1 & -1 & -2 & 0 \end{pmatrix}, q = \begin{bmatrix} -8 \\ -6 \\ -4 \\ 3 \end{bmatrix}.$$

A másodiknak megfelelő mátrix a [10] cikkben található. Az általunk vizsgált feladat (lcp02):

$$M = \left(m_{ij}\right)_{\substack{i=1..n \\ j=1..n}}, m_{ii} = 4i - 3, m_{ij} = 4i - 2, i \neq j$$

n = 10, q = [1, 5, 3, 6, 1, -7, 1, 8, 9, 1]^T.

6.1. A θ változtatásának vizsgálata

Az algoritmus minden lépésében megváltoztatjuk a centrális út környezetét, amely a β paraméterrel van jellemezve. Ennek értéke egy előre meghatározott alsó és felső határ között változhat. A módosítást a θ paraméter segítségével végezzük.

Megfigyelhetjük, hogy a θ csökkentése általában kevesebb iterációt eredményez (lásd az 1. és 2. táblázatot).

1. táblázat. *Eredmények lcp01-re*

Teszt sorszáma	θ ∈ (0, 1/2]	Iterációszám
1	0,04	19
2	0,07	20
3	0,2	20
4	0,5	20

2. táblázat. *Eredmények lcp02-re*

Teszt sorszáma	θ ∈ (0, 1/2]	Iterációszám
1	0,04	30
2	0,07	31
3	0,2	32
4	0,5	33

6.2. A γ változtatásának vizsgálata

A biztonsági lépésben bevezetettük a γ paramétert, mely a $(\Delta x^p, \Delta s^p)$ súlyozásáért felel, általa módosítható az $x(\alpha) = x + \alpha(\gamma \Delta x^p + \Delta x^a)$ és $s(\alpha) = s + \alpha(\gamma \Delta s^p + \Delta s^a)$ lépés. A következőkben vizsgáljuk, hogy a γ változtatása milyen hatással van az algoritmus kimenetelére.

Megállapítható, hogy ha nagyon kicsi a γ, akkor csökken a biztonsági lepés hatása, így több iterációt végzünk (lásd az 3. és 4. táblázatot).

3. táblázat. *Eredmények lcp01-re*

Teszt sorszáma	$\gamma \in (0, 1)$	Iterációszám
1	0,9	20
2	0,6	20
3	0,3	21
4	0,01	21

4. táblázat. *Eredmények lcp02-re*

Teszt sorszáma	$\gamma \in (0, 1)$	Iterációszám
1	0,9	32
2	0,6	32
3	0,3	33
4	0,01	34

7. Következtetések

A cikkben Ma, Liu, Zhang és Cong algoritmusát vizsgáltuk meg, mely két különböző lépést használva éri el rövid lépéses algoritmusok elméleti hatékonyságát. A megvalósítás szemszögéből nézve módosítottuk az algoritmust úgy, hogy egy γ paraméter segítségével súlyoztuk a biztonsági lépést.

A C++ programozási nyelvben írt kód segítségével sikerült alátámasztani az algoritmus hatékonyságát. Különböző monoton LCP-k esetén vizsgáltuk az iterációszám változását a θ, illetve γ paraméterek függvényében.

Annak ellenére, hogy az algoritmus monoton LCP-re vonatkozik a tesztelés során olyan feladatokra is jó eredményeket kaptunk, melyekre ez a tulajdonság nem teljesül. A **[11]**-ben közzétett 10×10 és 20×20 méretű elégséges mátrixokra is legtöbb 24 iteráció alatt sikerült megoldást találni. Megjegyezzük, hogy elsőként a **[12]** publikációban sikerült numerikus eredményeket elérni a **[11]**-beli feladatokra.

Eszerint érdemes a jövőben erre az általánosabb mátrixosztályra is megvizsgálni az algoritmus elméleti hatékonyságát.

Köszönetnyilvánítás

A szerzők köszönetüket fejezik ki az Erdélyi Múzeum-Egyesületnek a kutatási munkához nyújtott támogatásért.

Szakirodalmi hivatkozások

[1] Roos C., Terlaky T., Vial. J.-Ph.: *Theory and Algorithms for Linear Optimization*. Springer, NY, USA, 2005.

[2] Wright. S. J.: *Primal-Dual Interior-Point Methods*. SIAM, Philadelphia, USA, 1997.

[3] Ye. Y.: *Interior Point Algorithms, Theory and Analysis*. John Wiley & Sons, Chichester, UK, 1997/3. (1997)

[4] Kojima M., Megiddo N., Noma T., Yoshise A.: *A Unifed Approach to Interior Point Algorithms for Linear Complementarity Problems*. Lecture Notes in Computer Science 538, Springer Verlag, Berlin, Germany, 1991.

[5] Ai. W.: *Neighborhood-following algorithm for linear programming*. Sci. China serie A, 47. (2004) 812 – 820.

[6] Ai W., Zhang S.: *An O(√nL) iteration primal-dual path-following method, based on wide neighborhoods and large updates, for monotone LCP.* SIAM Journal on Optimization 16/2. (2005) 400–417. https://doi.org/10.1137/040604492

[7] Ma X., Liu H., Zhang J., Cong W.: *On superlinear and O(√nL) convergence of a path-following algorithm for monotone linear complementarity problems in a wide neighborhood.* Numerical Functional Analysis and Optimization, 38/5. (2017) 627–640. https://doi.org/10.1080/01630563.2017.1297824

[8] Darvay Zs., Takó I.: *Computational comparison of primal-dual algorithms based on a new software.* unpublished manuscript. (2012)

[9] Hock W., Shittkowski K.: *Test Examples for Nonlinear Programming Codes.* Lecture Notes in Economics and Mathematical Systems 187. Springer, Berlin (1981) https://doi.org/10.1007/978-3-642-48320-2

[10] Harker P. T., Pang J. S.: *A damped Newton method for linear complementarity problem.* In: Simulation an Optimization of Large Systems, Lectures on Applied Mathematics, AMS, Providence, RI, 26. (1990) 265–284.

[11] Morapitiye S.: *Sufficient Matrices*. (letöltve: 2019. február 9.). http://math.bme.hu/~sunil/su-matrices/

[12] Darvay Zs., Illés T., Povh J., Rigó P. R.: *Predictor-corrector interior-point algorithm for sufficient linear complementarity problems based on a new search direction*, manuscript. (2019)