社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－AL－94 (2)
2004／3／19

線形計画法の別解法についての実験報告

竹内 善和
近畿大学豊岡短期大学

山本 祥弘
鳥取大学工学部

抄録

ここで提案するアルゴリズムは単体法などと異なり。法線ベクトルを用いた幾何学的な考えに基づいている。

１９４７年に、G.B.Dantzingによって提案された単体法は数理計画の分野において大きな発展を遂げたが、その後線形計画問題はあまり進歩の跡がない。近年この分野での関心は、楕円体法や内点法のように、非線形の最適化問題に対する近似法に集中している。　ここでは、基本的な考え方と、アルゴリズムを示し、それによるいくつかの実行結果と単体法との計算時間の比較を発表する。

An experiment report about the another method of linear programming

Yoshikazu Takeuti
Kinki University
Toyooka junior College

Yoshihiro Yamamoto
Tottori University

Abstract

Algorithm to be suggested here is different from the simplex method and other,and based on the geometric idea that used a normal vector. The simplex method suggested in 1947 by G.B.Dantzing accomplished big development in a field of mathematical programming but, as for the linear programming problem, there is not a track of much progress afterwards.Like the ellipsoid method and the interior method, In late years interest in this field concentrates on an approximation method for a non-linear optimization problem. the basic idea and it's algorithm is proposed and comparison by result with simplex method is also presented.

1　目的

線形計画問題の解法は単体法、内点法などに代表されるが、複雑な手続きが必要であり、さまざまな改良がされている。ここで提案する方法は、最適解を与える端点をもとめる方法として、目的関数の超平面の法線ベクトルと、制約式から与えられる超平面の法線ベクトル間の角度、さらに、制約式から与えられる法線ベクトルたちの角度を指標とし、変数の個数だけ制約式を選び、連立方程式を解きながら求める端点に向かおうとするものである。　現在、最適解に確実に収束するアルゴリズムは得られていないが、こ

こでは現時点でのアルゴリズムを実行した結果を単体法と比較することによって、線形計画問題におけるこの方法の有効性を示すことにある。

2 準備

目的関数

$$z = c^T x, (z = \sum_{j=1}^{n} c_j x_j, c = (c_j), x = (x_j), j = 1, .., n) \cdots (1)$$

ここで、c は n 次元の定数ベクトル, x は n 次元の変数ベクトル
を制約式

$$Ax \leq b, (a_k^T x \leq b_k, A = (a_{kj}), b = (b_k), k = 1, 2, .., m) \cdots (2)$$

が定義する実行可能領域 D のもとで最大化する問題を取り扱う。
ただし、変数 x_jの非負条件 $x_j \geq 0$ は $-x_j \leq 0$ として制約式 (2) の中に含ませる
領域 D は空集合でないとする。

よく知られているように線形計画問題の最適解は実行可能領域の端点となっている。また、その端点は超平面の交点であり、目的関数から定義される超平面 $c^T x = p$ がその交点を通るならば、m 個の制約式から定義される超平面と超平面 $c^T x = p$ とのなす角度の最小値を与える超平面を、交点を構成する超平面の中から選んでみるというのは自然の考え方である。
$a_k^T x = b_k$で定義される超平面を A_kとし、A_kの法線ベクトル a_kと c のなす角度をθ_kとすれば

$$0 \leq \theta_k \leq \theta_j \Longleftrightarrow \cos\theta_k \geq \cos\theta_j, j = 1, ..., m$$

これは、

$$\frac{c^T a_k}{\|c\|\|a_k\|} \geq \frac{c^T a_j}{\|c\|\|a_j\|} \cdots (3)$$

と表せのるで、最適解を含む超平面をさがす指標として次の d_j を考える。

$$d_j = \frac{c^T a_j}{\|a_j\|} \cdots (4)$$

不等式 (3) は $d_k \geq d_j, (j = 1...m)$ と表せる。
また、実行可能領域の確定に貢献しない制約式が (2) のなかに存在する可能性があり、その原因は制約式の中の b_j の大きさに関係するので次の $\bar{d}_j$ を考える。

$$\bar{d}_j = \frac{\sum_{k=1}^{n} c_k a_{jk} - b_j}{\sqrt{\sum_{k=1}^{n} a_{jk}^2 + b_j^2}} \cdots (4)$$

$\bar{d}_j$ の役割は実行可能領域の確定に貢献しない制約式を除去するとき用いる。

3 アルゴリズム試案

$I = \{1, 2, ..., n, ..., m\}$ とおく。

Step1. $\bar{d}_j$、$j \epsilon I$ を求める

Step2. $\bar{d}_j$の最大を求めそれに対応する制約式を A_k とするとき a_{kj}を軸として掃きだす。j は $|a_{kj}|$ の最大に対応する。

Step3. k 行を除いて $(I = I - \{k\})step1, step2$ を繰り返す (n 回)。

Step4. それぞれの制約式を $A_1, A_2, ...A_n$ とし、その解 x^*を求める

Step5. 解 x^*がすべての制約式を満たすかどうかを調べる。

Step6. (A) もしすべての制約式を満たせば実行可能解である。最適解の判定は

$A_{(n)} = (a_1, a_2, ..., a_n), w = (w_1, w_2, ..., w_n)$　として

$A_{(n)}w = c$ を満たす解 w_iが $w_i \geq 0$ のとき x^*は最適解である。

負の w_jがあればそれらの最小をもとめそれに対応する A_jし、(2) から A_jを取り除いたものを (2), (1) を目的関数として $step1$ に戻る。

$(B)x^*$を満たさない制約式が存在すればそれらに対応する d_jの最大 d_kをもとめる。

$p_1 = \{A_1, A_2, ..., A_n\}$

$q_1 = \{$ 満たさない制約式の集合 $\} \cup p_1$ と定義する。

Step7. c と a_kを入れ替えた

$$g_j = \frac{\sum_{i=1}^{n} a_{ki}a_{ji}}{\|a_j\|}, j = 1...n$$

を計算し、g_jの最大をもとめ、それに対応する A_j と A_kを入れ替える。

$p_2 = (p_1 - \{A_j\}) \cup \{A_k\}$

Step8. $(A)p_u = p_v$となる u, v $(u < v)$ が存在したとき循環する。、このとき

(1) を目的関数、(2) の代わりを q_vとし $step1$ に戻る。

$(B)p_u \neq p_v$ならば $Step4$ に戻る。

4 実験結果

○実験目的

数式処理ソフト maple7(Waterloo maple) を用いたプログラムによる結果と maple7 に搭載されているのパッケージ (Simplex) による結果の計算時間の対比を行う。

○実験環境

Windows 版 maple7 を使用。(PC はスッペク 1.7GHZ、memory は 256Mbyte)

○データ作成条件

目的関数、制約式の係数は１０００から５０００までの整数とし、

乱数発生関数で求める。

○計算時間の計測方法

配列への代入が終了してからプログラム終了までを計測する。
計算が失敗した場合はそれを除いた計算例の時間の平均を採用。

table.computation result

	変数	制約式の個数	単体法でのx時間 (秒)	本アルゴリズムでの時間 (秒)
1	3	5	1以下	1以下
2	3	10	1以下	1以下
3	3	50	1以下	1以下
4	3	100	1以下	1以下
5	3	500	1.7	1.1
6	3	1000	3.2	3.2
7	3	5000	28	14
8	3	10000	77	34(1)
9	3	50000	1917	887

備考　＊上の *table* の時間は5回の計算時間の平均である。
＊(数字) 計算失敗した回数を示す.

5　考察

table は、単体法との計算時間の比較は提案した方法の方が優っているように思える。しかし、提案したアルゴリズムには問題が2つある。一つは、実行可能解が求まり、それが最適解でない場合、一個の制約式を無視しても最適解に影響しないとして破棄するようになっているが、未検証である。もう一つは循環にはいったときの q_j が最適解を含んでいない場合がある。実行例の8番目中に一個そのような存在する。一方 d_j と $\bar{d}_j$ の使い分けも課題である。提案したアルゴリズムでは Step4 までが $\bar{d}_j$, それ以外は d_j を用いている。最後に、この提案が単体法よりある部分では優っていることは明白である。今後はいま指摘した問題点を改良すべく、多くの問題、大きな問題を調べていきたい。