スパース処理技法を用いたカーマーカー法 による大規模線形計画問題の解法

正員 大堀 隆文 (北海道工大)
正員 大内 東 (北海道大)

1. ま え が き

線形計画問題 (Linear Programming: LP) の解法としては、1947 年に Dantzig によって発明された単体法 (シンプレックス法) が唯一の実用的な方法として約 40 年間にわたって用いられてきた⁽¹⁾。しかし、古くから知られているように、単体法では計算量の上限が「変数や制約式の個数の多項式オーダー」で押えられるという保証がない。単体法は、まれにはあるが極めて多くの反復を必要とする場合があり、計算量の面では指数オーダーの解法と言わざるを得ない⁽²⁾。従って、最悪の場合を評価の基準とする計算複雑度の立場からは、単体法が良い解法ではないことを意味している。

一方、単体法より良い解法を探す努力は様々な形で進められてきた。その一つが 1984 年に Karmarkar が発表した多項式オーダーの新解法 (以下、K 法と略記) である⁽³⁾。K 法のアルゴリズムは、基本的には射影変換された空間上でのポテンシャル関数最小化問題であり、最も時間のかかる部分は傾斜方向を求めるための計算である。従って、K 法を実際問題として効率良く計算機上で実現するためには、この部分の計算量を大幅に減少させなければならない。

いままでに、K 法の計算効率を上げるために様々な研究が行われてきたが、それらは大きく二つに分類される。一つは最大傾斜方向を他の方向で近似する方法であり⁽⁴⁾、もう一つは現実の大規模問題に特有のスパース (疎) 性を考慮に入れたスパース処理技法であ

る⁽⁵⁾。本論文のアプローチは後者に属し、任意のスパース構造をもつ大規模線形計画問題に対する K 法の効率的実現を目的とする。

K 法では、探索方向を求めるために各ステップで同じ構造の行列をもつ連立一次方程式を係数値だけを変えて幾度か計算しなければならない。このため、コレスキー分解用のサブルーチンをその都度呼んだのでは、そのルーチンへのデータのやり取りや零要素判定などに時間を要する。そこで、コレスキー法を使って何度も分解するのではなしに、行列の構造に合せて手計算でコレスキー分解するのと同じ方向のプログラムコードを生成し、係数値を変えてこのプログラムコードを何度も実行することを考える (コード生成法)⁽⁶⁾。

コード生成プログラム SYMBL はコレスキー分解を含めた K 法を直接実行するプログラム KMK を生成する。生成されたプログラム KMK では非零要素に対応した変数に関する式だけが現れる。従って、ループ処理の負担がない一方演算だけを実行すればよく、計算時間は圧倒的に少なくて済む。コード生成法では特定の構造をもった行列に適したプログラムを生成するので生成時間が余分にいるが、計算時間が大幅に少なくて済むため、K 法のように幾度か計算したいときに有利になる。しかも、いったん生成したプログラムをファイルに入れておけば、二度目の実行からは生成時間はいらない。非零要素が多くなると生成されるプログラムの長さは飛躍的に増大するが、処理が一方方向なので仮想記憶装置と結びついて有効な技法となっている。

本論文では、K 法におけるコード生成法の考え方、アルゴリズムおよびプログラミング環境を述べ、更にランダムに生成された大規模線形計画問題に対する数値実験結果を示す。

An Efficient Implementation of Karmarkar's Algorithm for Large Scale Linear Programs of Sparse Structure. By Takahumi Oohori, Member (Faculty of Engineering, Hokkaido Institute of Technology) & Azuma Ohuchi, Member (Faculty of Engineering, Hokkaido University).

大堀隆文: 正員, 北海道工業大学工学部電気工学科
大内 東: 正員, 北海道大学工学部情報工学科

2. Karmarkar 法

〈2.1〉 問題の定式化

問題 L : minimize $c^T x - \lambda$

subject to

$$x \in X = \{x \in E_+^n : Ax = b\} \dots\dots\dots (1)$$

ここで、 E^n は n 次元ユークリッド空間、 c は n 次元列ベクトル、 b は m 次元列ベクトル、 x は n 次元列ベクトル、 A は $m \times n$ 次元行列である。 λ は目的関数の最適値とする（一般には未知である）。また、 E_+^n は E^n の非負象限を表し、 $E_+^n = \{x \in E^n : x \geq 0\}$ となる。また、 T はベクトルおよび行列の転置を表す。

問題 L は次の仮定を満足しているものとする。

- (1) 実行可能内点 $x^0 > 0$ が存在する。
- (2) 目的関数の最適値 λ の下界値 λ^0 が既知である。
- (3) 最適解の集合が非空で有界である。
- (4) 係数行列 A の階数 (rank A) は m である。

仮定(4)は、K法で逆行列の代わりに A の一般化逆行列を用いると不要であるが、アルゴリズムの簡略化のために設けた⁽⁷⁾。

〈2.2〉 Karmarkar 法の概略

Karmarkar 法の本論文では単体から単体への射影変換を用いるため、原問題の制約集合 X を単体 $S_+ = \{x \in E_+^n : e^T x = 1, x \geq 0\}$ 上に閉じ込める変換を行っている。ここで、 $e = (1, 1, \dots, 1)^T$ である。本論文では、小島らに従い原問題をそのままの形で扱い、非負象限から単体への射影変換を用いる、いわゆる改訂 K 法の概略を述べる⁽⁸⁾。

改訂 K 法では、単体上で定義された Karmarkar によるポテンシャル関数の代わりに、次に示す正象限で定義されたポテンシャル関数を用いる。

$$\phi(x, \lambda) = (n+1) \ln(c^T x - \lambda) - \sum_{j=1}^n \ln x_j \dots\dots\dots (2)$$

簡単な計算によりわかるように、制約集合 X 上でポテンシャル関数を最小化することにより元の問題 L の最適解を求めることができる。従って、問題 L は次のポテンシャル最小化問題 P に変換される。

$$\begin{aligned} \text{問題 } P: & \text{ minimize } \phi(x, \lambda) \\ & \text{ subject to } x \in X \dots\dots\dots (3) \end{aligned}$$

小島は Karmarkar の理論に基づき、改訂 K 法が次の不等式を満足する可能解と下界値の列 $\{x^k, \lambda^k\}$ を作することを示した。

$$\phi(x^{k+1}, \lambda^{k+1}) < \phi(x^k, \lambda^k) - 1/5$$

$$(k=0, 1, 2, \dots) \dots\dots\dots (4)$$

簡単な計算によりアルゴリズムの一次収束性を示す次式が導かれる。

$$\begin{aligned} & (c^T x^k - \lambda^k) / (c^T x^0 - \lambda^0) \\ & < \text{const exp}\{-k/5(n+1)\} \\ & (k=1, 2, \dots) \dots\dots\dots (5) \end{aligned}$$

改訂 K 法では、 x^k を問題 L の正の可能解とするとき、次の E_+^n から S_+ への射影変換 T および逆変換 T^{-1} を利用する。

$$\begin{aligned} \hat{y} = \begin{bmatrix} y_0 \\ y \end{bmatrix} &= T(x) : y_0 = (n+1) - e^T y, \\ y &= (n+1)(D^k)^{-1}x / \{e^T(D^k)^{-1}x + 1\} \\ &\dots\dots\dots (6) \end{aligned}$$

$$x = T^{-1}(y_0, y) : T^{-1}(y_0, y) = D^k y / y_0 \dots\dots\dots (7)$$

ここで、 y_0 はスカラー、 y は n 次元列ベクトル、 D^k は対角行列で次の成分をもつ。

$$D^k = \text{diag}\{x_1^k, x_2^k, \dots, x_n^k\} \dots\dots\dots (8)$$

問題 P に射影変換 T を適用すると、次に示す変換されたポテンシャル最小化問題 P' を得る。

問題 P' : minimize

$$\begin{aligned} \phi'(y_0, y, \lambda^k) &= (n+1) \ln(c^T D^k y - y_0 \lambda^k) \\ &- \sum_{j=0}^n \ln \hat{y}_j - \sum_{j=1}^n \ln x_j^k \end{aligned}$$

subject to

$$\begin{aligned} \hat{y} \in Y &= \{\hat{y} \in E_+^{n+1} : -by_0 + AD^k y = 0, \\ &y_0 + e^T y = n+1\} \dots\dots\dots (9) \end{aligned}$$

現在の探索点 x^k は射影変換により単体 S_+ の中心 e に移るので、この点でポテンシャル関数 $\phi'(y_0, y, \lambda^k)$ の実行可能傾斜方向を求める。この方向に一定幅進むことにより関数 $\phi'(y_0, y, \lambda^k)$ ばかりではなく、元の関数 $\phi(x, \lambda^k)$ が一定量減少し、これが K 法の多項式性を実現している。 Y' を点 e における実行可能方向の集合とする。

$$\begin{aligned} Y' &= \{\hat{y} \in E^{n+1} : -by_0 + AD^k y = 0, \\ &y_0 + e^T y = 0\} \dots\dots\dots (10) \end{aligned}$$

また、 P^k を E_+^{n+1} から Y' への直交射影行列、 $\nabla \phi'$ を ϕ' の傾斜ベクトルとすると、理論的に実行可能傾斜方向は次式で表現される。

$$d^k = -P^k \nabla \phi'(e) \dots\dots\dots (11)$$

(11)式は簡単な計算によりベクトル $-P^k c^k$ と一致することがわかる。ここで、 $c^k = [-\lambda^k, c^T D^k]^T$ である。K 法の多項式性を保証するためには、探索方向に一定幅進むことが必要であるが、実際のアルゴリズム上は、以下に示すように一次元探索をするほうが効率が良い。

〈2.3〉 改訂 K 法のアロリズム

〔Step 0〕 (初期化)

$x^0 > 0$: 問題 L の初期可能解

$\lambda^0 < \lambda$: 問題 L の目的関数の初期下界値

$\varepsilon > 0$: 終了判定のための精度

$k = 0$: 繰返し回数

〔Step 1〕 (終了判定)

$c^T x^k - \lambda^k < \varepsilon$ ならば終了。

〔Step 2〕 (下界値の更新)

目的関数の新しい下界値 $\lambda^{k+1} (> \lambda^k)$ を計算する。

詳しくは文献 (4) または (8) を参照。

〔Step 3〕 (探索方向の計算)

変換されたコストベクトル c^k の実行可能部分空間 Y' への直交射影 d^k を求める。

$$d^k = -P^k c^k \\ = -[I - B^k B^{kT} (B^k B^{kT})^{-1} B^k - e e^T / (n+1)] c^k \quad \dots\dots\dots (12)$$

ここで、 $B^k = [-b, A D^k]$ である。

〔Step 4〕 (ポテンシャル関数の最小化)

関数 $\phi'(e + \alpha d^k / \|d^k\|)$ を $\alpha \in (0, \alpha^*)$ に関して最小化し最小点を α^k とする。ここで、 $\alpha^* = \max \{ \alpha \in E^1 : e + \alpha d^k / \|d^k\| \geq 0 \}$ とする。次の探索点は次のようになる。

$$y = \begin{bmatrix} y_0 \\ y \end{bmatrix} = e + \alpha^k d^k / \|d^k\| \quad \dots\dots\dots (13)$$

〔Step 5〕 (射影逆変換)

$$x^{k+1} = T^{-1}(y_0, y) = D^k y / y_0 \quad \dots\dots\dots (14)$$

〔Step 6〕 $k = k + 1$ とおき〔Step 1〕へ戻る。

〈2.4〉 初期可能解の求め方 (Phase 1)

いままでは、問題 L に対して正の初期可能解 x^0 の存在を仮定してきた。ここでは、Karmarkar の考えに基づき初期可能解を求める方法を述べる⁽³⁾。実行可能解が存在するとき目的関数の最小値が 0 となるような人工的な線形計画問題を考える。

問題 I : $\min x_{n+1}$

$$\text{subject to } Ax - b - x_{n+1}(Ax^{(0)} - b) = 0 \\ x \geq 0, x_{n+1} \geq 0 \quad \dots\dots\dots (15)$$

ここで、 $x^{(0)} \in E_+$ は任意のベクトルである。初期可能解 $(x^{(0)}, 1)$ を出発点として問題 I に K 法を適用すると、元の問題の可能解 $(x > 0, x_{n+1} = 0)$ に収束する。

3. スパース処理技法の適用

〈3.1〉 改訂 K 法のスパース性

改訂 K 法で最も時間のかかるのは〔Step 3〕の射影

行列 P^k の計算である。この部分の計算は次の順序で計算すると効率が良い。

(1) 行列の乗算 1: $B^k B^{kT} \dots\dots\dots 0(m^2 n)$ の算術演算

(2) 行列の乗算 2: $B^k c^k \dots\dots\dots 0(mn)$ の算術演算

(3) 連立一次方程式を解き、 $x = (B^k B^{kT})^{-1} B^k c^k$ を求める。

$(B^k B^{kT})x = B^k c^k \dots\dots\dots 0(m^3)$ の算術演算

(4) 行列の乗算 3: $B^{kT} (B^k B^{kT})^{-1} B^k c^k \dots\dots\dots 0(mn)$ の算術演算

二次の項を無視すると各反復で $[0m^2(m+n)]$ の算術基本演算を要し、全体として K 法の計算複雑度は理論的に $0(m^2 n(m+n)L)$ (L は精度に関する定数) となる。従って、実際に効率の良い改訂 K 法を実現するためには、この部分の計算量を大幅に減らさなければならない。幸いなことに、現実の大規模線形計画問題において、 A または $B^k B^{kT}$ はスパース性をもつので様々なスパース行列手法を適用し、計算の高速化が図れる。しかし、著者らは問題および K 法に特有の構造に着目して、より効率の良い手法を提案する⁽⁶⁾。

上記の行列演算で繰返し用いられる行列 B^k と $B^k B^{kT}$ は、 D^k の値が繰返しごとに変化するために係数の値が変わるが構造の変わらない行列である。従って、(1), (2), (4) 項では行列の乗算、(3) 項では方程式を解くためのコレスキー分解のプログラムコードを B^k 行列の非零構造を用いて前もって生成しておく。K 法の各反復では、プログラムコードに値を代入するだけの反復処理や分岐処理の負担のない単純処理となるので、高速化が実現できると考えられる。代償としての前処理、プログラムコード生成は一度だけでよいので、それほど負担にはならないと思われる。

〈3.2〉 コレスキー分解のコード生成法 行列の乗算のプログラムコード生成はそれほど困難ではないので、ここでは次の連立一次方程式を解くためのコレスキー分解のプログラムコード生成の方法を述べる。

$$(B^k B^{kT})x = B^k c^k \quad \dots\dots\dots (16)$$

以下、説明の都合上、上記方程式を一般形 $Ax = b$ と置く。コレスキー法では、方程式の対称性を考慮に入れて初めに次のように LL^T 分解する。

$$A = LL^T \quad \dots\dots\dots (17)$$

LL^T 分解の p 段目における各要素は次式で計算される。

$$l_{pp} = \sqrt{a_{pp} - \sum_{v=1}^{p-1} l_{pv}^2} \quad \dots\dots\dots (18)$$

$$l_{ip} = \left(a_{ip} - \sum_{v=1}^{p-1} l_{iv} l_{pv} \right) / l_{pp} \\ (i = p+1, \dots, m) \quad \dots\dots\dots (19)$$

次に方程式 $LL^T x = b$ を中間変数 y を用いて前進代入と後退代入により解く。

$$y_i = \left(b_i - \sum_{v=1}^{i-1} l_{iv} y_v \right) / l_{ii} \quad (i=1, \dots, m) \dots\dots\dots (20)$$

$$x_i = \left(y_i - \sum_{v=i+1}^m l_{vi} x_v \right) / l_{ii} \quad (i=m, \dots, 1) \dots\dots\dots (21)$$

ただし、便宜上 Σ の上限が下限より小さいときは式の値を 0 とする。

p 段目において、 A 行列のスパース性を利用した非零要素のコード展開方法を以下に述べる。(19)式において、もし a_{ip} または Σ の中の積の少なくとも一つが 0 でなければ l_{ip} の式をコード展開する。 Σ の中の $l_{p\alpha} \neq 0$ の項だけを保存し、これは i に独立なので各 p に対して一度だけ行う。次に、選ばれた項の中で $l_{i\alpha} \neq 0$ のものだけを式に展開する。

コード生成プログラム SYMBL は上記の手続きを実行し、プログラムコード KMK を生成する。二次元配列上の非零要素 a_{ij} および l_{ij} はそれぞれ LL^T 分解の自然な順序でラベル付けられた一次元配列に圧縮して記憶される。

〈3.3〉コード生成法のプログラミング環境 改訂 K 法におけるコード生成法はやや特殊なプログラ

図 1 コード生成による改訂 K 法のためのプログラミング環境

Fig. 1. Programming environment for revised Karmarkar method by code generation.

```

MODULE SPARSECHLS001;
define CHLS001;
const datamax=30250;datamaxfillin=60500;
type realvec1=array[0..datamax] of double;
      realvec2=array[0..datamaxfillin] of double;
var f      : extern realvec1;
      a      : extern realvec2;

procedure CHLS001;extern;
procedure CHLS001;
begin
  g[1]:=sqrt(f[1]);
  g[2]:=-f[2]/g[1];
  g[3]:=f[3]/g[1];
  g[4]:=-f[4]/g[1];
  g[5]:=-f[5]/g[1];
  g[6]:=-f[6]/g[1];
  g[7]:=-f[7]/g[1];
  g[8]:=-f[8]/g[1];
  g[9]:=sqrt(f[9]-g[2]*g[2]);
  g[10]:=(-g[3]*g[2])/g[9];
  g[11]:=(-g[4]*g[2])/g[9];
  g[12]:=(f[10]-g[5]*g[2])/g[9];
  g[13]:=(-g[6]*g[2])/g[9];
  g[14]:=(-g[7]*g[2])/g[9];
  g[15]:=(f[11]-g[8]*g[2])/g[9];
  g[16]:=sqrt(f[12]-g[3]*g[3]-g[10]*g[10]);
  g[17]:=(f[13]-g[4]*g[3]-g[11]*g[10])/g[16];

```

図 2 コード生成法によるコレスキー分解 PASCAL コードの例

Fig. 2. Example of Cholesky factorization PASCAL program by code generation

ミング環境を必要とする。なぜならば、それは与えられた線形計画問題の構造データ(零・非零行列)と数値データを別々に扱うからである。著者らは、そのような環境を APOLLO DOMAIN ワークステーション上で実現した。しかし、著者らのアルゴリズムは特殊な装置を必要としないので、他のシステム上でも容易に実現できる。また、プログラミング言語として PASCAL を用いたが、他の言語、例えば FORTRAN, C などを用いても容易にアルゴリズムを実現できる。

図 1 は改訂 K 法のためのプログラミング環境を示す。図 2 は SYMBL によって生成されたコレスキー分解の PASCAL プログラムコードの最初の部分を示す。

〈3.4〉密ベクトルの処理 以前にも述べたように、現実の線形計画問題の制約行列は非常に疎であるが、多くの非零要素からなる列(密ベクトル)が含まれる可能性が幾つかある。

- (1) Phase 1 における人工変数 x_{n+1} の係数ベクトル $(Ax^{(0)} - b)$ (〈2.4〉節参照)
- (2) 右辺ベクトル b
- (3) 変数結合形制約式
- (4) min-max 形目的関数の場合
- (5) その他特殊構造行列

改訂 K 法の (Step 3) における $B^k B^{kT}$ 行列は以下に示すように、元の行列 A の中に一本でも密ベクトルが

存在すれば密行列になる。

$$B^*B^{*T} = bb^T + A(D^*)^2A^T \\ = bb^T + \sum_{j=1}^n (x_j^*)^2 a_j a_j^T \dots\dots\dots (22)$$

ここで、 a_j は A の第 j 列ベクトルである。このように、任意の密ベクトルは B^*B^{*T} 行列のスパース性を破壊してしまうという欠点をもつが、この問題点は以下のように Sherman-Morrison 公式を適用すれば解決できる。 g を B^* 行列の中の密ベクトルとすると

$$(B^*B^{*T})^{-1} = (BB^T + gg^T)^{-1} \\ = (BB^T)^{-1} - \frac{(BB^T)^{-1}gg^T(BB^T)^{-1}}{1 + g^T(BB^T)^{-1}g} \\ \dots\dots\dots (23)$$

もし、 B がスパースであれば、スパース性を考慮して BB^T を前もってコレスキー分解しておき、以下の手順で方程式を解けばよい。

- (1) BB^T のコレスキー分解 LL^T を求める。
- (2) 方程式 $(LL^T)y = g$ を前進および後退代入により解く。
- (3) 方程式 $(LL^T)x = B^*c^* - gy^TB^*c^*/(1 + g^Ty)$ を (2) 項と右辺だけが異なる前進および後退代入により解く。

4. 数値実験

上記で述べたコード生成法による改訂 K 法の数値実験結果を述べる。すべての実験は、APOLLO DOMAIN DN 570 上で倍精度演算により行われた。DN 570 は 8 MB の主記憶と 86 MB のハードディスクをもつ 1.5 MIPS の処理速度をもつワークステーションである。

いずれのテスト問題も現実の大規模問題に近い構造をもたせるために、係数行列をスパースかつ帯行列とした。

〈4.1〉 テスト問題 1 Avis-Chvátal のランダム線形計画問題⁽²⁾をもとに、スパース性を考慮に入れた次の問題を考える。

$$\text{問題 } T1: \text{ maximize } c^T x \\ \text{subject to } Ax \leq b, x \geq 0 \dots\dots (24)$$

ここで、 c と b は 1 と 2 の間の一様乱数を要素とするベクトルである。 A は $m \times m$ のランダムスパース帯行列である。すなわち、帯の内側（ここでは帯幅を 20 に固定した）では、係数 a_{ij} は 0 または 1 と 2 の間の一様乱数値をもち帯の外側では 0 の値をとる。 A の対角要素は非有界を避けるため 1 に固定してある。問題 $T1$ は正の可能解 $x = \varepsilon$ ($0 < \varepsilon \ll 1$) をもつので

表 1 テスト問題 $T1$ の数値実験結果
(列平均非零数 = 3)

Table 1. Computational results for problem $T1$ (each column has 3 nonzeros on average).

問題の 大きさ (m)	シンプレックス法		カーマーカー法					比	
	反復回数	実行時間 (s)	コード生成時間	コンパイル時間	反復回数	実行時間 K_1	全時間 K_2	$\frac{S}{K_1}$	$\frac{S}{K_2}$
100	78	5	6	153	17	12	171	0.4	0.03
200	153	10	14	314	18	23	351	0.4	0.03
300	248	27	23	476	19	33	532	0.8	0.05
400	340	48	35	656	20	45	736	1.1	0.07
500	414	67	47	797	20	54	898	1.2	0.07

表 2 テスト問題 $T1$ の数値実験結果
(列平均非零数 = 6)

Table 2. Computational results for problem $T1$ (each column has 6 nonzeros on average).

問題の 大きさ (m)	シンプレックス法		カーマーカー法					比	
	反復回数	実行時間 (s)	コード生成時間	コンパイル時間	反復回数	実行時間 K_1	全時間 K_2	$\frac{S}{K_1}$	$\frac{S}{K_2}$
100	107	13	12	363	19	18	393	0.7	0.0
200	296	134	28	771	22	37	836	3.6	0.2
300	448	369	45	1,186	21	52	1,283	7.1	0.3
400	571	653	66	1,631	23	76	1,773	8.6	0.4
500	723	1,344	92	2,041	23	92	2,225	14.6	0.6

Phase 1 を必要としない。

列平均非零数を 3 と 6 のそれぞれの場合、問題のサイズ m をパラメータ ($m=100, 200, 300, 400, 500$) として数値実験を行った結果を表 1 と表 2 に示す。比較のために、スパース処理技法を用いた逆行列の積形式改訂単体法の計算結果を合せて載せてある。表には、 K 法と単体法の計算時間 (秒)、繰返しの回数などが含まれ、いずれも 3 回の試行の平均値である。表中の K_1 は K 法の実行時間、 K_2 はコード生成、コンパイルおよび実行時間の総和を表す。

いずれの表においても、全計算時間で比較すると単体法のほうが効率がよいが、実行時間では非零数が 3 のとき $m=400$ 、非零数が 6 のとき $m=200$ で計算時間が逆転している。また、非零数が増えるときの計算量の増加率は、単体法のほうが圧倒的に大きい。従って、単体法のスパース処理技法は非零の割合が小さい場合は有効に働くが、非零数が増加するにつれて効果が弱くなる。一方、 K 法は非零数が増えなくても単体法ほど極端に計算量は増えないので、安定な解法と言える。

〈4.2〉 テスト問題 2 小島らのランダム線形計画問題⁽⁴⁾をもとに、スパース性を考慮に入れた次の問題を考える。

表 3 テスト問題 T2 の数値実験結果
(列平均非零数 = 3)

Table 3. Computational results for problem T2 (each column has 3 nonzeros on average).

問題の大きさ (m)	シンプレックス法		カーマーカー法						比	
	反復回数	実行時間 (s)	コード生成時間	コンパイル時間	反復回数	実行時間 K_1	全時間 K_2		$\frac{S}{K_1}$	$\frac{S}{K_2}$
100	265 (138)	52 (15)	6	138	15 (2)	11 (1)	155		5	0.3
200	553 (295)	222 (78)	15	285	17 (2)	20 (3)	320		11	0.7
300	818 (437)	611 (195)	25	427	18 (3)	30 (5)	482		20	1.3
400	1,132 (599)	1,011 (376)	38	585	18 (3)	39 (6)	662		26	1.5
500	1,387 (730)	1,891 (483)	54	738	18 (3)	50 (8)	842		38	2.2

表 4 テスト問題 T2 の数値実験結果
(列平均非零数 = 6)

Table 4. Computational results for problem T2 (each column has 6 nonzeros on average).

問題の大きさ (m)	シンプレックス法		カーマーカー法						比	
	反復回数	実行時間 (s)	コード生成時間	コンパイル時間	反復回数	実行時間 K_1	全時間 K_2		$\frac{S}{K_1}$	$\frac{S}{K_2}$
100	309 (154)	86 (30)	12	289	16 (2)	13 (2)	314		7	0.3
200	620 (314)	561 (181)	28	594	17 (2)	25 (4)	647		22	0.9
300	959 (464)	1,919 (593)	50	904	17 (3)	37 (6)	991		52	1.9
400	1,315 (622)	4,432 (1,122)	78	1,242	18 (3)	50 (9)	1,370		89	3.2
500	1,639 (804)	8,069 (2,252)	110	1,586	18 (3)	64 (11)	1,760		126	4.6

問題 T2: maximize $c^T x$

subject to $Ax = b, x \geq 0, \dots (25)$

ここで、 c は 1 と 2 の間の一様乱数を要素とするベクトルである。 A は $m \times 2m$ のランダムスパース帯行列であり、問題 T1 と同じ性質をもつ行列である。 b は各行の係数ベクトルのユークリッドノルムとした。問題 T2 は正の初期可能解を探すために、Phase 1 を必要とする。

問題 T1 と同様に、列平均非零数を 3 と 6 のそれぞれの場合、問題のサイズ m をパラメータ ($m=100, 200, 300, 400, 500$) として数値実験を行った結果を表 3 と表 4 に示す。表中の () の中の数字は Phase 1 の値を表す。

いずれの表においても、実行時間では圧倒的に K 法のほうが効率が良いうえ、全計算時間でも m が 300 以上のとき K 法のほうが有利である。列非零数が 6 で $m=500$ の場合、実行時間が 126 倍、全計算時間で 4.6

倍もの差が出た。テスト問題 1 と 2 の違いは () の中の数字からわかるように、K 法では Phase 1 に要する時間が単体法と比べて極端に少ないからであると考えられる。単体法が初期基底を作るために制約式の個数だけの人工変数を必要とするのに対して、K 法では 1 個で十分である (〈2.4〉節参照)。

以上まとめると、本論文の手法は、スパースな帯行列において、(1) Phase 1 を必要とする、(2) 列非零数が多い、(3) 規模が大きい問題に対して有効に働くことがわかった。

5. 結 論

著者らは改訂 K 法に対して、線形計画問題のスパース構造を最大限に利用したコード生成アルゴリズムを開発した。本手法が成功するかぎには、コードを実行することによって得られる計算時間の減少がコード生成にかかる前処理時間を上回ることである。スパースな帯行列に対する数値実験結果によると、問題の規模が小さく非零要素が極端に少ない場合は、単体法が K 法を上回ることもあるが、それ以外のときは K 法が有利である。また、Phase 1 の処理も K 法が有利であり、問題の規模が大きくと列非零数が多くなるにつれて、その差が開くことがわかった。しかし、問題のスパース性がなくなると生成されるプログラムコードが膨大になり、全計算時間に占めるコンパイル時間の割合が大きくなることが問題点として残る。

また、線形計画法を実際に業務用に利用しているところでは、同じ構造をもつ問題を繰返し解くことが多いので、本手法が一層有利になると思われる。非線形計画問題や整数計画問題を解くときに、線形計画問題をサブルーチンとして繰返し解く場合にも同様の利点が生じると思われる。

問題点および今後の課題として次のものが挙げられる。

- (1) スパースデータをより効率的に処理するためのリスト構造の採用とプログラムの改良
- (2) コレスキー分解時に新たに生じる非零数 (fill-in) の抑制。
- (3) 他の内点法 (Adler-Karmarkar 法⁽⁹⁾、伊理-今井法⁽¹⁰⁾ など) へのコード生成法の適用。
- (4) 生成コード用特殊コンパイラの作成。
- (5) 現実問題に対するより広範な数値実験。

最後に、本論文のプログラム作成および数値実験に御協力下さった北海道工業大学電気工学科技官 鳴海一彦氏に深く感謝いたします。

(昭和 63 年 5 月 25 日受付)

文 献

- (1) G.B. Dantzig: Linear Programming and Extensions (1963) Princeton University Press, Princeton, NJ
- (2) D. Avis, et al.: "Note on Bland's Pivoting Rule", *Mathematical Programming Study*, **8**, 24 (1978)
- (3) N. Karmarkar: "A New Polynomial-time Algorithm for Linear Programming", *Combinatorica*, **4**, 373 (1984)
- (4) M. Kojima, et al.: "An Efficient Implementation of Karmarkar's New LP Algorithm", Technical Report, No. B-180, Dept. of Information Sciences, Tokyo Inst. of Tech. (1986)
- (5) J.A. Tomlin: An Experimental Approach to Karmarkar's Linear Programming Algorithm (1985) Manuscript, Keron Inc. Mountain View, CA
- (6) F.G. Gustavson, et al.: "Symbolic Generation of an Optimal Crout Algorithm for Sparse Systems of Linear Equations", *JACM*, **17**, 87 (1970)
- (7) D.M. Gay: "A Variant of Karmarkar's Linear Programming Algorithm for Programs in Standard Form", *Mathematical Programming*, **37**, 81 (1987)
- (8) 小島: 「Karmarkar 法の改良について」, 第 6 回数値計画シンポジウム論文集, p. 243 (昭 60)
- (9) I. Adler, et al.: "An Implementation of Karmarkar's Algorithm for Linear Programming", Working Paper, OR Center, U. of California (1986)
- (10) M. Iri, et al.: "A Multiplicative Barrier Function Method for Linear Programming", *Algorithmica*, **1**, 455 (1986)
- (11) 大堀, 他: 「LU 分解のスパース処理技法によるカーマーカ一法の高速化」, 昭 62 電気関係北海道支部連大