A-021

凹最小化問題に対する Falk-Soland の分枝限定法に関する一考察

A Consideration on the Falk-Soland Branch-and-Bound Method for Concave Minimization Problems

斉藤 恵一 † Keiichi Saito　施 建明 ‡ Jianming Shi　田中 章 † Akira Tanaka　河口 万由香 † Mayuka F. Kawaguchi　宮腰 政明 † Masaaki Miyakoshi

1 序論

凹最小化問題 (Concave Minimization Problem) とは，有界な凸閉集合上で目的関数である多変数凹関数を最小化するという問題である．目的関数が凹関数である場合，複数の局所的最適解が存在する．全ての局所的最適解を列挙することによって大域的最適解は見つかるが，問題の次元あるいは制約条件の式が増えると全ての候補を列挙することは困難となる．

Tuy はこの問題を解く二つのアルゴリズムを提案した [1]．一つは実行可能領域のなかにある大域的最小点を含まない部分を削り落としていく凹性カット法 (concavity-cut method)，もう一つは整数計画問題・組合せ最適化問題で使われる分枝限定法 (branch-and-bound method) である．その後に，凸計画法にも使われる外部近似法 (outer approximation method) も加わるが，60 年代に開発されたこの三つの手法が今も非凸計画法の主役である．また，Tuy による大域的最適化アルゴリズムの 5 年後には Falk-Soland の分枝限定法 (Falk-Soland Branch-and-Bound Method)[2] が発表されている．このように，凹最小化問題を解くための大域的最適化アルゴリズムが次々と開発され，比較的大規模な問題でも現実的な時間のうちに解決することができるようになった．

本稿では，Falk-Soland の分枝限定法のアルゴリズムを取り上げる．実際にこのアルゴリズムを実装し，様々な凹最小化問題に対する計算機実験を行い，その結果について考察する．

2 分離可能な凹最小化問題

Falk-Soland の分枝限定法のアルゴリズムが適用できる凹最小化問題は以下のように記述される．

$$\left|\begin{array}{ll} \text{最小化} & f(\boldsymbol{x}) \equiv \sum_{j=1}^{n} f_j(x_j) \\ \text{条　件} & \boldsymbol{x} \in D \end{array}\right. \qquad (1)$$

ここで，$D \subset \boldsymbol{R}^n$ は凸閉集合，$f : \boldsymbol{R}^n \rightarrow \boldsymbol{R}$ は凹関数，$\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^{\mathsf{T}}$ は n 次元実数ベクトルである．また目的関数 f が $\boldsymbol{x}$ の成分 x_j それぞれの凹関数の和で表されなければならない．このとき，f は分離可能 (separable) であるという．(1) のような分離可能な凹最小化問題はネットワーク流問題などに現れる．

3 Falk-Soland の分枝限定法

3.1 アルゴリズムの概要

まず，実行可能領域 D 上における各変数 x_j の下界値 l_j と上界値 u_j を求め，D を含む領域 M を定める．その M を，いくつかの領域 M^k に分割する．分割するときは M における各辺の長さ $L_j = |u_j - l_j|$ $(j = 1, 2, \ldots, n)$ を計算し，そのなかの最も長い辺 $L_{j_0} = \max\{ L_j \mid j = 1, 2, \ldots, n\}$ をとって分割する．また計算の効率性を考えると，多くの分割を行うよりも 2 分割して，領域 M^1 と M^2 を形成する方が望ましい．そして分割によって形成された領域と D の共通部分で下界値 $\overline{f}_k$ を計算し，その時点での暫定値 f^0 との比較により選んだ領域を捨て去るか (**限定操作**) あるいは 2 つに分割するか (**分枝操作**) を決定する．ここで使われる暫定値 f^0 とは，それまでに得ている最良な実行可能解 (暫定解) を代入して求められる目的関数値である．分枝操作のところでそれまでに得ている暫定値のなかで最小な値を選び，それを新しい暫定値として更新する．このような手順を繰り返しながら最適解が含まれる領域を絞り込んでいく．

3.2 アフィン関数導入による下界値計算

目的関数が凹関数，D が有界な凸閉集合である場合，問題の最適解は必ず D の端点に存在する [3]．そのため，D の端点をすべて調べ，その中から最適値を達成する点を探せばよい．しかし，次元が増えることによって問題の変数や制約条件の式が増え，その結果端点の個数も指数関数的に増加することが知られており，端点をすべて列挙することは困難である [4]．また分枝限定法では，各領域で求める下界値 $\overline{f}_k$ が重要な値である．この下界値を求める方法はアルゴリズムの効率を大きく左右し，また凹関数である f から下界値 $\overline{f}$ $(\overline{f} = \min_k\{\overline{f}_k\})$ を求めることは困難であるため，下界値 $\overline{f}$ をいかにして求めるかが問題となる．そこで，目的関数 f の分離可能性を利用して，下界値 $\overline{f}$ を求めるために，以下のアフィン関数 g_j を導入する．

$$g_j(x_j) = \frac{f_j(u_j{}^k) - f_j(l_j{}^k)}{u_j{}^k - l_j{}^k} x_j + \frac{u_j{}^k f_j(l_j{}^k) - l_j{}^k f_j(u_j{}^k)}{u_j{}^k - l_j{}^k} \qquad (2)$$

このアフィン関数 g_j を導入して線形計画問題を作り，単体法 (場合によっては二段階単体法) でその最適値を求める．その最適値を f の下界値 $\overline{f}_k$ としその最適解 $\overline{\boldsymbol{x}}_k$ は問題 (1) の実行可能解となるので，$f(\overline{\boldsymbol{x}}_k) < f^0$ ならば暫定値を $f^0 = f(\overline{\boldsymbol{x}}_k)$ と更新することができる．もちろん，ここで得られた $\overline{f}_k$ は問題 (1) の目的関数値ではない．分枝限定法の分枝操作と暫定値の更新を繰り返しながら，f の最適値に近づけていき f の最適値を求めるのである．このように難しい凹最小化問題を緩和して沢山の易しい問題を解くという方法を利用することにより問題を解決することができる．

4 プログラムの構成

4.1 領域の定義と分割

領域を定義する点については，すべての点の座標を保持する必要はなく，最も遠い距離にある 2 点の座標を保持するだけでよい．このことを利用し，保持すべき点の座標を格納するため

† 北海道大学大学院情報科学研究科 CS 専攻
‡ 室蘭工業大学情報工学科

の配列として 3 次元配列を用意した．3 次元配列には以下のような形式で要素を格納している．

$$\begin{array}{ccccc} \text{d2[0][0][0]} & \leftarrow & x_0 \quad y_0 & \rightarrow & \text{d2[0][0][1]} \\ \text{d2[0][1][0]} & \leftarrow & x_1 \quad y_1 & \rightarrow & \text{d2[0][1][1]} \\ \vdots & & \vdots \quad \vdots & & \vdots \\ \text{d2[0][N-1][0]} & \leftarrow & x_{N-1} \quad y_{N-1} & \rightarrow & \text{d2[0][N-1][1]} \end{array}$$

$(x_0, x_1, \ldots, x_{N-1})$，$(y_0, y_1, \ldots, y_{N-1})$ のそれぞれで 1 点の座標を表している．また 3 次元配列の左側にある添字が領域番号の数字を表し，残り 2 つの添字が座標を格納するための 2 次元配列を表している．この 3 次元配列を利用すれば，配列に格納する要素数を少なくすることができる．

領域の分割については，分割する前の領域を定める 2 点の座標を保持しておき，分割における座標の特徴をうまく利用することで行っている．

4.2 アルゴリズムに関する工夫

このアルゴリズムは『ある領域を選び，その領域における下界値を計算し，暫定値との比較を行う』という操作を繰り返すことから，一連の操作を再帰的に行うよう設定した．そのため，領域を探索する順番については再帰的な操作を繰り返すということで，領域番号の順番通りとした．

また各領域で得られる下界値と暫定値との比較により領域の分割を行うが，その領域には最適解がなく，分割する必要がないことが後になってわかる場合がある．よってすぐに領域の分割を行わず，探索する全ての領域の下界値を計算し，暫定値を更新した後に比較を行い，領域の分割を行うかを決定する．

5 計算機実験

C 言語によりプログラムを実装し，以下の点を検証すべく計算機実験を行った．実験に用いた計算機環境は表 1 の通りである．

- 非負条件を除いた制約条件の式，次元の増加による平均実行時間，領域の平均分割回数の変化
- アルゴリズムにおける有限回での収束性，プログラムの終了条件について

表 1 実験に用いた計算機環境

OS	WindowsXP
CPU / メモリ	Pentium(R) 4 CPU 2.53GHz / 0.99GB
ソフトウェア	Microsoft Visual Studio .NET 2003

なお，設定した凹最小化問題における目的関数は 2 次関数または線形関数である．目的関数を構成する変数の係数はランダムに乱数で発生させ，各変数 x_j は非負条件をもつこととした．また，Falk-Soland の分枝限定法では有限回でのアルゴリズムの収束が保証されていないため，許容誤差 ε を導入してプログラムの終了条件を定めた．

6 実験結果と考察

平均実行時間，領域の平均分割回数の変化を図 1 に示す．(a) は非負条件を除いた制約条件の式，(b) は次元に対する結果である．

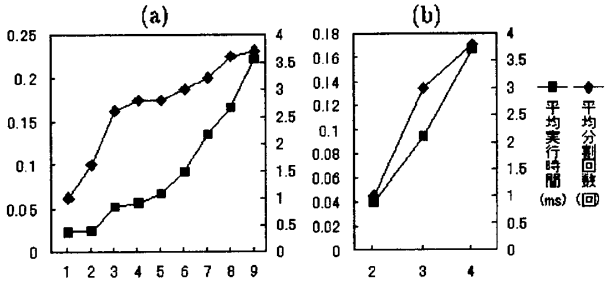

図 1 平均実行時間，領域の平均分割回数の変化

平均実行時間，領域の平均分割回数がともに増加しており，(a)，(b) を比較しても次元の増加による影響が大きいことがわかる．制約条件の式，次元の増加にしたがって実行可能領域 D の端点の個数も多くなっていくが，その端点の増え方には違いがある．そのため最適値を達成する点の候補も変わってくるため，図 1 に見られるような違いが生じると考えられる．また，問題によっては D の構造が複雑になるときもあり，最初に定義する領域 M の端点と D の端点が一致しないものが多い場合は領域の分割回数が増える傾向にあると考えられる．さらに下界値計算に使われるアフィン関数によっては下界値 $\overline{f}_k$ とそれまでの暫定値 f^0 との差がかなり大きい場合もあり，目的関数 f の最適値を見つけるための領域の絞り込み回数が多い．

次に，アルゴリズムにおける有限回での収束性について述べる．有限回でのアルゴリズムの収束が保証されていないため，許容誤差 ε を導入してプログラムの終了条件を定めた．しかし今回の実験を通して有限回で収束する例があることがわかった．領域で得られる下界値とその時点での暫定値が一致して探索する領域がないという判断でプログラムが終了する場合である．この場合，最後の段階で実行可能領域の端点を通るような領域の分割を行っており，得られる解は厳密な最適解であり最適値も真の最適値であることがいえる．もちろん，ε による終了条件に依存する場合もある．この場合は厳密な最適解ではなく大域的近似解となり，大域的に見ても真の最適値との誤差が ε しかもたない点となるはずである．

参考文献

[1] Tuy, H.(1964), "Concave programming under linear constraints," *Soviet Math.* 5, pp.1437-1440.

[2] Falk, J.E. and R.M. Soland(1969), "An algorithm for separable nonconvex programming problems," *Manag. Science* 15, pp.550-569.

[3] 久保幹雄，田村明久，松井知己 (2002)，応用数理計画ハンドブック，朝倉書店，pp.674-675.

[4] David Avis, Luc Devroye(2000), "Estimating the number of vertices of a polyhedron", *Information Processing Letters* 73, pp.137-143.

[5] 久野誉人 (1995)，非凸計画問題≠解けない問題～分枝限定法による大域的最適化～，オペレーションズ・リサーチ 1995 年 5 月号，pp.232-233.