筑波大学大学院博士課程

システム情報工学研究科修士論文

単体上における非凸関数の大域的最適化

千葉 竜介

修士（工学）

（コンピュータサイエンス専攻）

指導教員　久野 誉人

2017年3月

概要

本論文では単体と呼ばれる領域の上での最適化を研究する．単体であることの条件はとても単純であり，単体上の最適化問題は，多様な実問題から考えられた経緯がある．目的関数は非凸関数としているが，非凸関数の最適化は凸関数の最適化に比べて困難であるとされている．本論文では非凸関数の中でも単調関数を扱う．単調関数とは変数の値を増やすと関数値も必ず増大もしくは減少するような関数である．このような単調関数を単体上で最適化する．最適化手法は分枝限定法を用い，単体をグリッド化した上で実行した．さらに，その手法が多項式時間アルゴリズムであることを示し，計算実験によりアルゴリズムの有用性を確認した．

目 次

第 1 章　序論 **1**
1.1 記号の定義 1
1.2 構成 1

第 2 章　単体上の単調関数最適化問題 **3**
2.1 単調関数 3
2.2 単位単体 4
2.3 単体上の単調関数の性質 4

第 3 章　関連研究 **7**
3.1 分枝限定法 7
3.2 単体のグリッド化 7
3.3 単体の分割 9
3.3.1 分割の手順 9
3.3.2 部分問題のグリッド数 11
3.3.3 分割数の性質 12

第 4 章　提案手法 **13**
4.1 分割する次元の選択 13
4.2 単体の保持 14
4.2.1 二分木の構成 14
4.2.2 二分木から問題を生成する方法 15
4.3 低次元の単体の処理 16
4.4 アルゴリズム 16
4.5 並列化 18
4.6 計算量の解析 18

第 5 章　数値実験 **19**
5.1 条件 19
5.2 実験結果 19

第 6 章　結論 **20**

謝辞 **21**

参考文献 **22**

図 目 次

2.1 $(0.7, 0.3, 0), (0, 1, 0), (0, 0.3, 0.7)$ を頂点に持つような単体の下界を与える点 . 6

3.1 $c = 1, n = 3, m = 3$ としてグリッド化された単体 $G(c, k, m)$ 8
3.2 2 つに分割されたグリッド化された単体 11

4.1 $n = 3, m = 3$ の問題を解いたときの二分木 T の例 15

第1章　序論

本論文で扱う，単体上の最適化問題は単純な制約条件を持ち，多様な実問題からこのような問題が考えられた経緯がある．単体上の最適化手法は De Klerk [1] により，目的関数が連続である場合や二次多項式である場合，または緩和について論じられている．本研究は非凸関数を目的関数とするが，一般に非凸関数の最適化は凸関数の最適化よりも困難であるとされている．特に非凸関数の中でも，変数の値を増やすと必ず関数値も増加もしくは減少するような関数である単調関数を目的関数として扱う．目的関数と制約条件がともに単調関数であるような問題の最適化手法は Tuy [2] などにより研究されているが，本論文で扱う問題である単調関数の単体上での最適化はこれまであまり研究されていない．

単体上の単調関数を分枝限定法で最適化する手法を提案する．Bomze, De Klerk [3] により研究された単体のグリッド化を実行した上でその単体を分割し，単調関数の単体上での下界を利用したアルゴリズムを構築した．また，その手法の計算量が多項式であることがわかった．さらに，計算実験により本アルゴリズムの有用性を確認した．

1.1　記号の定義

表記	意味
$\mathbb{R}$	実数全体の集合
$\mathbb{R}^n$	n 次元実数ベクトル全体の集合
$\mathbb{R}^n_+$	すべての要素が非負である n 次元実数ベクトル全体の集合
$\mathbb{Z}$	0 を含む自然数全体の集合
$\mathbb{Z}^n$	n 次元自然数ベクトル全体の集合
e_i $(e_i \in \mathbb{R}^n)$	i 番目の要素が 1 でそれ以外が 0 であるような単位ベクトル
x_i $(x \in \mathbb{R}^n)$	ベクトル x の i 列目の要素
$x < y$ $(x, y \in \mathbb{R}^n)$	$x_i < y_i$ $(i = 1, ..., n)$
$x \leq y$ $(x, y \in \mathbb{R}^n)$	$x_i \leq y_i$ $(i = 1, ..., n)$

1.2　構成

第 1 章では序論を述べ，記号の定義を行った．第 2 章では本論文で扱う問題の定義を行い，その性質について論じる．次に第 3 章で関連研究の説明を行い，第 4 章で提案手法の説明を

する．第 5 章では数値実験の結果を紹介し，第 6 章で結論を述べる．

第2章　単体上の単調関数最適化問題

本章では，本論文で扱う問題の説明をする．以下のような単調関数 f の最小化問題を考える．

$$\left|\begin{array}{ll} \text{最小化} & f(x) \\ \text{条件} & \displaystyle\sum_{i=1}^{n} x_i = 1 \\ & x_i \geq 0, i = 1, ..., n \end{array}\right.$$

変数 $x \in \mathbb{R}^n$ の各要素の和が 1 であることと非負であることが制約条件とされている．このような条件を満たす x の集合は単体と呼ばれており，様々な研究がなされている．まずは単調関数と単体を定義し，それらの性質について論ずる．

2.1　単調関数

単調関数の定義を行う．単調関数とは，単調増加関数または単調減少関数のことであり，入力の増加に対して関数値も常に増加または減少する関数である．単調増加関数を以下のように定義する．

定義 1. $I \subseteq \mathbb{R}^n$ の関数 $f : I \to \mathbb{R}$ は任意の $x, x' \in I$ に対して以下が成り立つ場合に単調増加関数であるという．

$$x < x' \Rightarrow f(x) \leq f(x')$$

一方，単調減少関数は以下のように定義できる．

定義 2. $I \subseteq \mathbb{R}^n$ の関数 $f : I \to \mathbb{R}$ は任意の $x, x' \in I$ に対して以下が成り立つ場合に単調減少関数であるという．

$$x < x' \Rightarrow f(x) \geq f(x')$$

ここで I は n 次元実数空間全体や区間，離散的な空間が考えられる．

定理 1. $f : I \to \mathbb{R}$ が単調増加関数であることと，$-f$ が単調減少関数であることは同値である．

証明. f は単調増加関数であるから，任意の $x, x' \in I$ について以下が成り立つ.

$$\begin{aligned} & x < x' \Rightarrow f(x) \leq f(x') \\ \iff \ & x < x' \Rightarrow -f(x) \geq -f(x') \end{aligned}$$

これは $-f$ が単調減少関数であることを示し，f が単調増加関数であることと，$-f$ が単調減少関数であることが同値であることが証明された. □

定理 1 より，関数値を -1 倍する操作によって単調増加と単調減少が切り替わり，単調性は失われないことがわかる．さらに，最適化問題において，ある関数 f を最大化することは $-f$ を最小化することと同じであるため，関数値を -1 倍することで最大化問題を最小化問題に変換することができる．ここからは単調増加関数の最小化問題についてしか論じないが，この通り最大化問題は最小化問題に変換することができ，単調性は保存されることから，一般性は失わない.

2.2 単位単体

定義 3. 以下のような Δ_n を単位単体という.

$$\Delta_n \triangleq \left\{ x \in \mathbb{R}_+^n \;\middle|\; \sum_{i=1}^{n} x_i = 1 \right\}$$

n について具体的に考えてみると，$n = 2$ の単体は線分，$n = 3$ の単体は平面上の正三角形，$n = 4$ の単体は三角錐となることがわかる.

2.3 単体上の単調関数の性質

単調増加関数 $f : \mathbb{R}_+^n \to \mathbb{R}$ の単位単体上における性質について考える．まず，単体の定義から以下が成り立つ.

定理 2. 単体上の任意の点 $x, y \in \Delta_n$ について $x < y$ が成り立つことはない.

証明. 背理法で示す．$x < y$ となるような x, y が Δ_n 中に存在すると仮定する.
$x < y$ であるから，それぞれの要素について以下が成り立ち，

$$\begin{aligned} & x_i < y_i \ , i = 1, ..., n \\ \Longrightarrow \ & \sum_{i=1}^{n} x_i < \sum_{i=1}^{n} y_i \end{aligned}$$

これは $x, y \in \Delta_n$ つまり，$\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i = 1$ に矛盾する.
よって，$x < y$ が成り立つような単体上の点 x, y は存在しないことが証明された. □

定理 2 から，単調増加関数の唯一の性質である $x < y \Rightarrow f(x) < f(y)$ を利用できる点が単体上には存在しないことがわかり，単体上だけを見ると特に性質のない任意の関数であるように見える．よって，単体上における単調関数の最適化は一般的な大域的最適化問題を解くことと同程度に困難であると予想される．しかし，単体以外の領域の情報を利用し，単体上の関数値の下界を得ることができる．

定理 3. 以下のような要素を持つ $x_{LB} \in \mathbb{R}^n_+$ は単体の任意の部分領域 $A \subseteq \Delta_n$ における単調増加関数 $f : \mathbb{R}^n_+ \to \mathbb{R}$ の関数値の下界 $f(x_{LB})$ を与える．

$$(x_{LB})_i = \min_{y \in A} y_i \ , \ i = 1, ..., n$$

証明. x_{LB} の生成方法より，$A \backslash \{x_{LB}\}$ の任意の要素 x について $x_{LB} < x$ が成り立つ．
また，f は単調増加関数であったから，$\forall x, y \in \mathbb{R}^n_+, x < y \Rightarrow f(x) < f(y)$ が成り立ち，

$$\forall x \in A \backslash \{x_{LB}\}, f(x_{LB}) < f(x)$$

以上により，$f(x_{LB})$ は単体の任意の部分領域 A における単調増加関数 f の下界となる． □

x_{LB} の算出をするためには単体の部分領域 A における変数の最小値を次元ごとに求める必要がある．A の形状がわからない場合，この処理は一般の最適化問題を解くことであるから困難である．ただし，特に A が単体である場合，つまり A が元の単体の部分単体となっている場合，部分単体のすべての頂点を調べるだけで x_{LB} の算出は完了する．なぜなら単体は凸であるから，ある次元の最小の変数値は必ず頂点が持つ．よって，A が n 次元単体であった場合，n 個の頂点を調べるだけで x_{LB} の算出は完了し，その計算量は $O(n)$ である．

例えば，$(0.7, 0.3, 0), (0, 1, 0), (0, 0.3, 0.7)$ を頂点に持つような $n = 3$ の単体は Δ_3 の部分単体であるが，この単体における x_{LB} は，それぞれの次元の最小値を集めた $(0, 0.3, 0)$ である．図 2.1 は Δ_3 が薄い青色，部分単体が青色，x_{LB} が赤い点で描画されている図である．

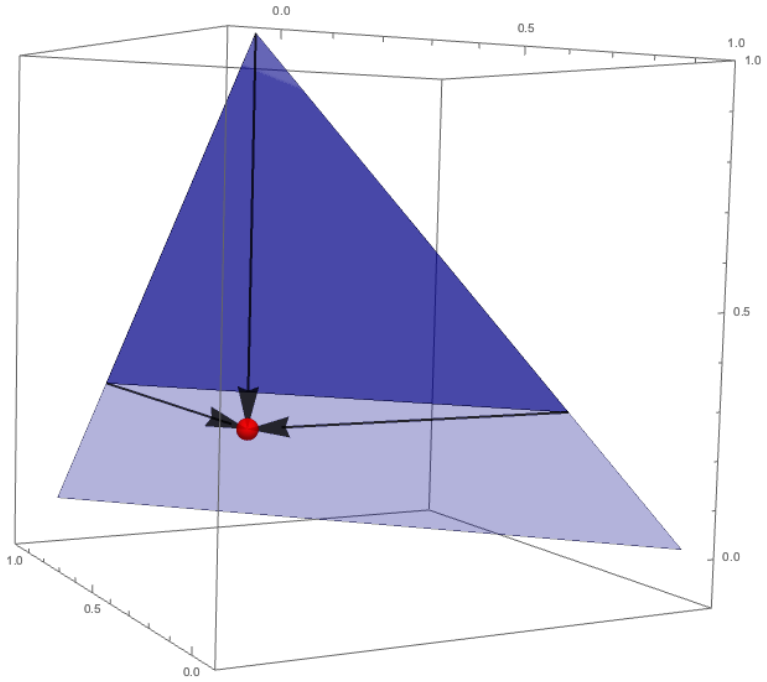

図 2.1: $(0.7, 0.3, 0), (0, 1, 0), (0, 0.3, 0.7)$ を頂点に持つような単体の下界を与える点

第3章　関連研究

3.1　分枝限定法

ここでは一般的な分枝限定法について説明する．分枝限定法とは，大域的最適化の汎用アルゴリズムであり，広く利用されている．実行可能領域を分割し，それぞれの分割された領域の下界値，上界値と暫定最適値を比較することで最適解が存在し得ない領域を削除し，探索する領域を絞り込む．この操作を繰り返して効率的に探索する最適化手法である．

最小化問題を解く場合を考えながらアルゴリズムの説明を行う．分枝限定法は大きく分けて分枝・限定の 2 種類の操作から成る．分枝操作では，実行可能領域の集合からある領域 S を選択し，$S_1 \cup S_2 \cup ... \cup S_n = S, S_1 \cap S_2 \cap ... \cap S_n = \phi$ を満たす n 個の部分領域に分割する．分割の方法や分割数はアルゴリズムによって異なり，問題に対して適した手法で分割することが重要である．次に限定操作では，それぞれの部分領域について下界値を求める．その下界値が，他の領域の上界値よりも大きい場合や，暫定最適値よりも大きい場合はその部分領域に最適解が存在し得ないため，その領域の探索を打ち切る．ある領域の探索を打ち切り，削除することを刈り込みと呼ぶ．領域の刈り込みを効率的に行うには，よい下界，上界を得ること，探索中に素早くよい暫定最適値を見つけることが重要である．以上の操作をすべての領域を刈り込むまで繰り返し実行し，終了時点の最適値を問題の最適値として出力する．

3.2　単体のグリッド化

de Klerk ら [4] により行なわれている，グリッド化した単体上での最適化手法を説明をする．グリッド化とは，実行可能領域に格子を考え，格子の交差点だけを抽出し，実行可能領域の内部に等間隔に存在する点群を得ることである．そして，得られた点群の中で最も良い点を探す．グリッド化によって解候補の数は減少し，解きやすくなる．

単体のグリッド化の定義を行う．$m \in \mathbb{Z}$ を利用して，n 次元の単位単体の各辺を m 分割して得られるグリッド化された単位単体は以下のように表せる．

$$\{\, x \in \Delta_n \mid mx \in \mathbb{Z}^n \,\}$$

これまでは単位単体のみを扱ってきたが，正の実数 c を利用して，各辺の長さが c であるような単体 $c\Delta_n$ をグリッド化したもの $G(c, n, m)$ を以下のように定義する．

定義 4.

$$G(c, n, m) \triangleq c\Delta_n \cap \{\, x \in \mathbb{R}^n \mid mx \in \mathbb{Z}^n \,\}$$

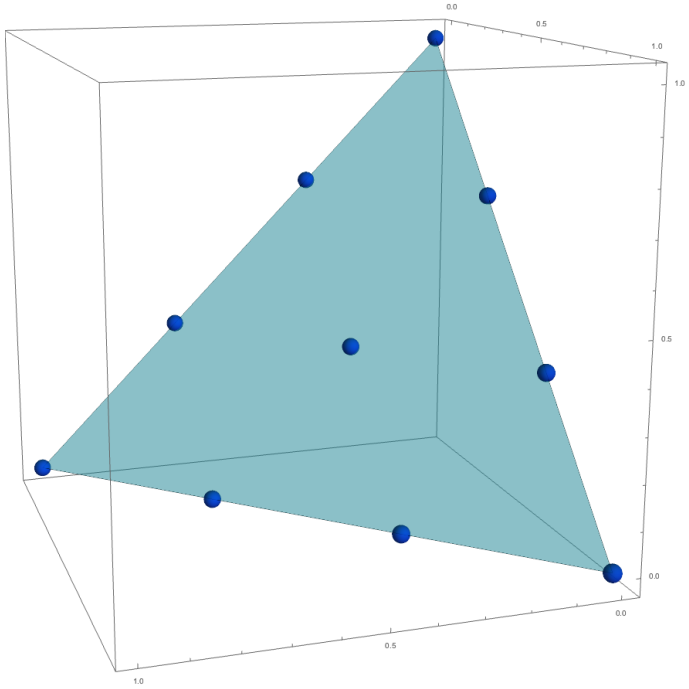

図 3.1: $c = 1, n = 3, m = 3$ としてグリッド化された単体 $G(c, k, m)$

ここで，$c = 1$ としたとき，$G(1, n, m)$ は n 次元の単位単体の各辺を m 等分してグリッド化したものと同じであることがわかる．

さらに，単体をグリッド化したときのグリッドの総数について，以下が成り立つことがわかっている．

定理 4. グリッド化された単位単体 $G(1, n, m)$ が持つ点の数は二項係数を利用した以下の数となる．

$$|G(1, n, m)| = \binom{n + m - 1}{m}$$

これは m を固定した場合の n の多項式であり，グリッド化した単体上を全探索しても n の多項式オーダーでアルゴリズムが完了することがわかる．しかし，n, m を大きくした場合の $\binom{n+m-1}{m}$ はとても大きい数であり，例えば $n = 10, m = 100$ の場合，生成される点の数はおよそ 8.5×10^{12} であり，全探索は難しいと容易に想像できる．

図 3.1 は $c = 1, n = 3, m = 3$ としてグリッド化した単体の図である．単体 Δ_3 を薄い青色で，グリッド $G(1, 3, 3)$ を青色で描画している．グリッドを数えると 10 個であることがわかり，これは $\binom{3+3-1}{3}$ と一致することが確認できる．

3.3 単体の分割

本節では，単体の分割手法について説明する．分枝限定法の分枝操作では実行可能領域をいくつかの部分領域に分割するが，本問題の実行可能領域は単体であるから，単体を分割する手法を考える．さらに，分枝操作は繰り返し行われるため，分割された部分領域も同様にすべて単体となっている必要がある．

3.3.1 分割の手順

外崎ら [5] による単体の分割手法は，グリッド化された単体を，1 次元少ない部分単体と，1 列少ない部分単体の 2 つに分割する方法である．この手法を具体的に説明する．

まずは本研究で扱う問題，グリッド化された単位単体上の最小化問題 P を以下のように定義する．

$$P \left| \begin{array}{ll} \text{最小化} & f(x) \\ \text{条件} & x \in G(1,n,m) \end{array} \right.$$

$mb \in \mathbb{Z}^n$ を満たすような非負ベクトル $b \in \mathbb{R}^n_+$ を利用して c を次のように与える．

$$c \equiv 1 - \sum_{j=1}^{n} b_j > 0$$

このとき次元のインデックスの部分集合 $K = \{j_1, ..., j_k\} \subset \{1, ..., n\}$ を選択すると，問題 P の部分問題を以下のように定義することができる．

$$P(K,b) \left| \begin{array}{lll} \text{最小化} & f(x) & \\ \text{条件} & \displaystyle\sum_{j=1}^{n} x_j = 1 & \\ & mb_j \le mx_j \in \mathbb{Z}, & j \in K \\ & x_j = b_j, & j \notin K \end{array} \right.$$

K に選択されなかった次元の変数値は b の値に固定され，K に選択された次元の変数のみを変数とするような問題となる．$y \in G(c,k,m)$ として変数を取り直すと，$P(K,b)$ は以下の問題と等価である．

$$\left| \begin{array}{lll} \text{最小化} & f(x) & \\ \text{条件} & y \in G(c,k,m) & \\ & y_i = x_{j_i} - b_{j_i}, & j = 1, ..., k \\ & x_j = b_j, & j \notin K \end{array} \right.$$

この問題は各辺の大きさが c である k 次元のグリッド化された単体上の問題である．さらに定義より，$G(c,k,m)$ は各辺を mc 個に分割したグリッドの集合であり，定理 4 より以下が成り立つ．

$$|G(c,k,m)| = \binom{k+mc-1}{mc}$$

次に，K から一つのインデックス j_k を選択し $P(K,b)$ を二つの部分問題に分割する．

$$P(K, b+e_{j_k}/m) \left| \begin{array}{lll} \text{最小化} & f(x) & \\ \text{条件} & \displaystyle\sum_{j=1}^{n} x_j = 1 & \\ & mb_j \le mx_j \in \mathbb{Z}, & j \in K\backslash\{j_k\} \\ & mb_{j_k}+1 \le mx_{j_k} \in \mathbb{Z} & \\ & x_j = b_j, & j \notin K \end{array} \right.$$

$$P(K\backslash\{j_k\}, b) \left| \begin{array}{lll} \text{最小化} & f(x) & \\ \text{条件} & \displaystyle\sum_{j=1}^{n} x_j = 1 & \\ & mb_j \le mx_j \in \mathbb{Z}, & j \in K\backslash\{j_k\} \\ & x_j = b_j, & j \notin K \\ & x_{j_k} = b_{j_k} & \end{array} \right.$$

同様にして，新しい単体上の変数 y を利用すると，それぞれ以下のように書き直すことができる．

$$\left| \begin{array}{lll} \text{最小化} & f(x) & \\ \text{条件} & y \in G(c-1/m,k,m) & \\ & y_i = x_{j_i} - b_{j_i}, & i = 1,...,k-1 \\ & y_k = x_{j_k} - b_{j_k} - 1/m & \\ & x_j = b_j, & j \notin K \end{array} \right.$$

$$\left| \begin{array}{lll} \text{最小化} & f(x) & \\ \text{条件} & y \in G(c,k-1,m) & \\ & y_i = x_{j_i} - b_{j_i}, & i = 1,...,k-1 \\ & x_j = b_j, & j \notin K \\ & x_{j_k} = b_{j_k} & \end{array} \right.$$

$P(K, b+e_{j_k}/m)$ は $P(K,b)$ からグリッドが一列減った問題であり，$P(K\backslash\{j_k\},b)$ は次元が 1 次元減った問題である．以上のようにグリッド化された単体を，一列減った単体と一次元

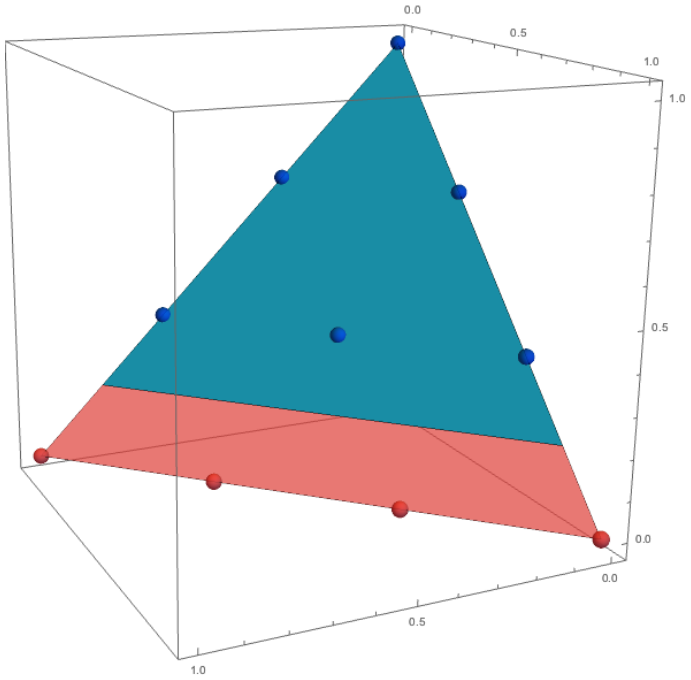

図 3.2: 2 つに分割されたグリッド化された単体

減った単体に分割することができた．もともと単体には一列という概念は存在しないが，グリッド化をしたことによってこのような操作が可能になった．

図 3.2 はグリッド化した単体を本手法で分割した図である．青い点の集合は一列減った単体 $P(K, b + e_{j_k}/m)$ を表し，赤い点の集合が一次元減った単体 $P(K\backslash\{j_k\}, b)$ を表している．元の単体は $n = 3$ の単体であったが，赤い点の集合は $n = 2$ の単体，つまり線分となっており，一次元減っていることがわかる．また，青い点の集合を見てみると，こちらは次元に変化はなく，変わらず $n = 3$ の単体であるが，グリッドが一列削られていることがわかる．

3.3.2 部分問題のグリッド数

分割された問題 $P(K, b+e_{j_k}/m), P(K\backslash\{j_k\}, b)$ の定義域のグリッド数について以下がわかっている．

$$|G(c-1/m, k, m)| = \binom{k+mc-2}{mc-1}$$

$$|G(c, k-1, m)| = \binom{k+mc-2}{mc}$$

また，二項係数の性質として以下が知られている．

$$\binom{k+mc-1}{mc} = \binom{k+mc-2}{mc-1} + \binom{k+mc-2}{mc}$$

ゆえに，以下が成り立つことが確認できる．

$$|G(c,k,m)| = |G(c-1/m,k,m)| + |G(c,k-1,m)|$$

この分枝操作を繰り返すと，すべての部分問題の実行可能領域はただ一点のみを含む単体になり，その総数は $\binom{k+mc-1}{mc}$ となる．

3.3.3 分割数の性質

分枝限定法において問題は繰り返し部分問題に分割されるが，問題をノードに持つ木構造を考えることができる．分割される前の問題を親ノードとして，分割された部分問題を子ノードとすると，アルゴリズムが進行するにつれて木は子ノードを増やして大きくなる．本章で説明した分割によって生成される木を T とすると，問題は 2 分割されるため T は 2 分木である．さらに，問題を完全に分割し切った場合，部分問題の実行可能領域はただ一点を持つ単体となるが，この部分問題は T の末端の葉ノードにあたる．そしてこの二分木 T のノード数について以下のことが言える．

定理 5. T の総ノード数は $2\binom{n+m-1}{m} - 1$ である

証明. 帰納的に示す．$r = \binom{n+m-1}{m}$ とおく．$q < r$ であるような q に対して，q を葉ノードの総数とする二分木の総ノード数は $2q-1$ であると仮定する．
次に T の根ノードを親に持つような二つの部分木を T_1, T_2 とし，それらの葉ノードの数を r_1, r_2 とすると，$r = r_1 + r_2$ が成り立つ．さらに，$r_1, r_2 \leq r-1$ であるから仮定より，T_1, T_2 の総ノード数は $2r_1 - 1, 2r_2 - 1$ である．
よって，T の総ノード数は

$$(2r_1 - 1) + (2r_2 - 1) + 1 = 2(r_1 + r_2) - 1 = 2r - 1$$

□

第4章　提案手法

本章では，単体上の単調関数を分枝限定法で最適化する提案手法の説明をする．分枝限定法において，領域の分割方法，下界の良さはアルゴリズムの速度に対してとても重要であり，本研究では単調関数の下界に定理 3 で得られるものを利用し，分割方法は 3.3 節で説明したもの利用する．

4.1　分割する次元の選択

分枝限定法において，分割された部分問題が良い関数値を持つことは，その部分領域が良い下界を持つことにつながり，その下界によってその領域を削除しやすくなる．さらに同時に，良い暫定解を得ることにもつながるため，部分問題が良い関数値を持つことはアルゴリズムの早い収束に対して重要である．本節では，部分領域が良い関数値を持つような 3.3 節における次元のインデックス j_k の選び方，つまり，問題を分割する次元の決定方法について論じる．

3.3 節の問題 $P(K,b)$ では，元の変数 x の代わりに $y \in G(c,k,m)$ となるような変数 y が取り直されている．$G(c,k,m)$ は次元のインデックス j_k を選択することで二つの部分単体 $G(c,k-1,m)$, $G(c-1/m,k,m)$ に分割される．選択された j_k 次元の値について詳しく見てみると，$G(c,k-1,m)$ では j_k 次元の変数値 x_{j_k} は b_{j_k} に固定されており，これに対応する $y \in G(c,k,m)$ における変数値 y_k は最小値 0 となる．

本問題で扱っている関数は単調増加関数であるから，ある次元 k に着目したとき，y_k が大きくなれば関数値は単調に増加し，逆に y_k が小さくなれば関数値は単調に減少する．よって，関数値に対して大きなインパクトを与えている次元 k を探し出し，次元 k について単体を分割すると，部分問題 $P(K\backslash\{k\},b)$ の次元 k の変数値は最小値に固定され，関数値は小さくなると予想できる．

次に，そのような大きなインパクトを与えている次元 k の探し方を考える．$G(c,k,m)$ の頂点の集合を考えると，それらは c 倍した k 次の単位行列 I の列を取り出したものとなっている．つまり，k 個の頂点の要素は以下のようになっている．

$$(c,0,0,...,0),(0,c,0,...,0),(0,0,c,...,0),...,(0,0,0,...,c)$$

ある次元 i に着目すると，i 番目の頂点の次元 i の値は c で，それ以外の頂点の次元 i の値は 0 である．ここで，全ての頂点について関数値を調べ，最も悪い関数値を持った頂点を j 番目の頂点であるとする．ここで，関数の単調性を考えてみると，頂点の次元 j の値のインパク

トが大きいために関数値が悪くなっているのではないかと予想できる．そこで，よい関数値を持った部分領域を切り出すために，次元 j について単体を分割し，部分問題の次元 j の変数値を最小値に固定する．

分割元の単体を入力として受け取り，本手法によって分割する次元を決定するプログラムの疑似コードを Algorithm 1 に示した．

Algorithm 1 Decide the cutting dimension of simplex

```
1:  function DecideCuttingDimension(simplex)
2:      Initialize worst_value
3:      Initialize dimension
4:      for each vertex of simplex do
5:          function_value := ObjectiveFunctionValue(vertex)
6:          if function_value ≥ worst_value then
7:              worst_value := function_value
8:              dimension := GetDimension(vertex)
9:          end if
10:     end for
11:     return dimension
12: end function
```

4.2 単体の保持

本節では分枝限定法の分枝操作によって分割された部分単体の，プログラム上での扱い方について説明する．3.3.3 節で述べた通り，問題の分割は木構造を持つ．アルゴリズムは問題 P を根ノードとして始まり，アルゴリズムが進行するにつれて木のノード数は指数的に増大する．分枝限定法ではすべての葉ノードの問題を分割する必要があるが，大きなサイズの問題を解く場合，コンピュータのメモリ上に全ての葉ノードのオブジェクトを保持し続けることは困難となる．この問題を解決するために，すべての葉ノードを保持するのでなく，分割の情報を表現する二分木 T を保持し，葉ノードが表す部分問題はそのつど二分木の情報から生成する．

4.2.1 二分木の構成

プログラム中における二分木の作り方を説明する．二分木のノードは部分問題を表現し，問題を分割した際に単体を分割した次元をノードに保存する．$P(K, b)$ を分割すると二つの部分問題が生成されるが，グリッドが一列減った部分問題 $P(K, b + e_{j_k}/m)$ は左の子ノードに，一次元減った部分問題 $P(K\backslash\{j_k\}, b)$ は右の子ノードに挿入する．以上のルールに従って二分

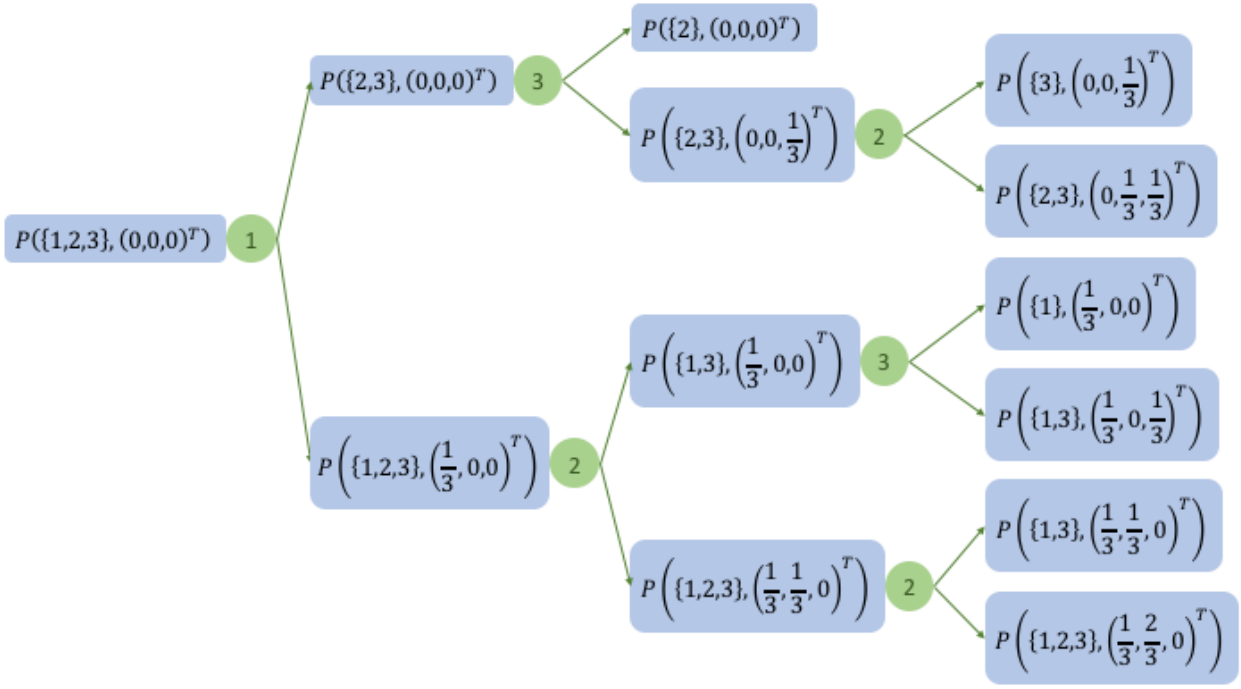

図 4.1: $n = 3, m = 3$ の問題を解いたときの二分木 T の例

木を構成することで，二分木に保持された情報から，ある葉ノードの部分問題における単体の形状を復元することができる．

$n = 3, m = 3$ の問題を解いた際の二分木 T の構造の例を図 4.1 に示した．アルゴリズムの開始状態は $K = \{1, 2, 3\}$, $b = (0, 0, 0)^T$ である．緑色の丸に囲まれた数字は単体を分割した次元を表し，グリッドが一列減った問題は下側の子ノード，次元が減った問題は上側の子ノードとして書いた．ここで，実際にプログラムが保持するのは図における緑色の部分のみ，つまり単体を分割した次元と二分木の構図のみであり，問題や単体のオブジェクトは保持しない．どのようにこの二分木からノードが表す問題を生成するかを次で説明する．

4.2.2 二分木から問題を生成する方法

二分木の情報から，葉ノードが表す部分問題の単体を生成する．調べたい葉ノードから根ノードまで，親ノードをたどる．その際，左右どちらの子ノードであるかの情報と分割された次元を取得し，その二つの情報から葉ノードが表す部分問題を一意に再現できる．ある葉ノードを入力とし，そのノードが表現する部分問題の実行可能領域単体を返す疑似コードを Algorithm 2 に示した．この処理の計算量は，ノードが最も深い場所にある場合に最悪となるが，木の深さは高々 $n + m$ であるから，$O(n + m)$ である．

Algorithm 2 Regenerate the simplex from the binary tree

```
1:  function REGENERATESIMPLEX(node)
2:      simplex := GENERATEUNITSIMPLEX()
3:      while parent := PARENT(node) do
4:          cutting_dimension := binary_tree[parent]
5:          if node is left child node then
6:              simplex := CUTONEROW(simplex, cutting_dimension)
7:          else if node is right child node then
8:              simplex := CUTONEDIMENSION(simplex, cutting_dimension)
9:          end if
10:         node := parent
11:     end while
12:     return simplex
13: end function
```

4.3 低次元の単体の処理

アルゴリズムが進行するにつれ，分割された問題の次元は小さくなる．ここで分割の情報を保持する二分木 T を考えてみると，木の末端に近い部分には低次元の部分問題しか存在せず，木の末端に近いということはそれらの数がとても多いことが容易に想像できる．よって，低次元の問題を効率良く処理することはアルゴリズムの速度に対して重要である．

外崎ら [5] による研究では単体が 1 次元の単体 (点) になるまで分割を行っていた．例えば 2 次元の単体 (線分) を分割するとき，その単体は一次元減った単体 (点) と一点が取り除かれた単体 (線分) に分割される．下界によって刈られる場合を除き，この処理を単体の分割数回 (m 回) 繰り返すことによってやっと全ての単体が点となりアルゴリズムが終了する．分割を繰り返すことでアルゴリズムのオーバーヘッドはかさむため，低次元の問題は別の単純なアルゴリズムで探索し，速度を向上したい．低次元の問題一つ一つにおける実行時間の改善は小さいものであるが，その数はとても多いため，アルゴリズム全体として速度の大幅な改善が期待される．

分割が進み，問題が 2 次元，つまり実行可能領域が線分になった場合は単純に全てのグリッドの関数値を評価する．この処理は，単体が m 分割されている場合は $O(m)$ の処理となる．疑似コードを Algorithm 3 に示した．

4.4 アルゴリズム

これまでに説明した処理をまとめた疑似コードを Algorithm 4 に示した．これは単調増加関数を単体上で最小化するアルゴリズムである．

Algorithm 3 Line search

function LINESEARCH($line$)
    for each $point$ of $line$ **do**
        $function\_value$:= OBJECTIVEFUNCTIONVALUE($point$)
        if $function\_value < current\_optima$ **then**
            $current\_optima := function\_value$
        end if
    end for
end function

Algorithm 4 Optimize monotonic function on simplex

Initialize $binary\_tree$
Initialize $current\_optima$
i := 0
while $true$ **do**
    $nodes := i$ th row of $binary\_tree$
    if $nodes$ are empty **then**
        break
    end if
    for each $node$ of $nodes$ **do**
        $simplex$ = REGENERATESIMPLEX($node$)
        $lower\_bound$:= CALCULATELOWERBOUND($simplex$)
        if $current\_optima \leq lower\_bound$ **then**
            continue
        end if
        if $simplex$ is a line **then**
            LINESEARCH($simplex$)
            continue
        end if
        if $current\_optima \geq$ some vertices of $simplex$ **then**
            Replace $current\_optima$
        end if
        $cutting\_dimension$ = DECIDECUTTINGDIMENSION($simplex$)
        $binary\_tree[node] := cutting\_dimension$
    end for
end while
return $current\_optima$

4.5 並列化

本アルゴリズムの並列化について述べる．分割の情報を保持する二分木 T における，同一列のノードには相互に依存性はないため，列ごとに処理が完了していることを保証すればアルゴリズムを並列化できる．Algorithm 1 による分割する次元の計算や，Algorithm 3 による線形探索を並行して実行できるため，アルゴリズム全体に対して速度の向上が図れるかもしれない．

ただし，$G(1, n, m)$ を実行可能領域とする問題に対して，Algorithm 1 のオーダーは高々 $O(n)$ で，Algorithm 3 は高々 $O(m)$ であるため，それほど重い処理ではない．よって問題の大きさによっては，大きな速度の改善が見られない，もしくはスレッドの切り替えのオーバーヘッドがかさみむしろ改悪することも考えうる．

具体的には，Algorithm 4 における 10 - 23 行目を並列化できる．

4.6 計算量の解析

Algorithm 4 の計算量を考える．まずはサブモジュールとして使っている処理を解析する，10 行目で二分木から単体を生成している処理は 4.2.2 節で説明した通り高々 $O(n+m)$，11 行目で下界を与える点を計算する処理は 2.3 節の通り $O(n)$，16 行目の線形探索は Algorithm 3 を利用して高々 $O(m)$，最後に，22 行目の分割する次元を決定する処理は Algorithm 1 により単体の頂点を調べ上げれば完了するため $O(n)$ である．以上により，m を固定したときサブモジュールの計算量は高々 $O(n)$ であることがわかった．さらに，定理 5 により，分割の情報を保持する二分木 T の総ノード数は $\binom{n+m-1}{m}$ であることがわかっており，これは n に対する多項式である．多項式の多項式は多項式である性質から，Algorithm 4 全体の計算量は n の多項式であることがわかった．

第5章　数値実験

TODO: 書く

5.1　条件

5.2　実験結果

第6章　結論

TODO: 書く

謝辞

TODO: 書く

参考文献

[1] E De Klerk, The complexity of optimizing over a simplex, hypercube or sphere: a short survey, Central European Journal of Operations Research 16.2 (2008), pp. 111-125.

[2] H. Tuy, Monotonic optimization: Problems and solution approaches, SIAM Journal on Optimization 11.2 (2000), pp. 464-494.

[3] IM Bomze, and E De Klerk, Solving standard quadratic optimization problems via linear, semidefinite and copositive programming, Journal of Global Optimization 24.2 (2002), pp. 163-185.

[4] E De Klerk, D Den Hertog, and G. Elabwabi, On the complexity of optimization over the standard simplex, European journal of operational research 191.3 (2008), pp. 773-785.

[5] 外崎真造, 非凸 2 次制約付き配合計画問題のロバスト最適化, 筑波大学システム情報工学研究科修士学位論文 (2010)