

## 2 レベル 0-1 計画問題に対する 遺伝的アルゴリズムを用いた Stackelberg 解の改良型計算手法

丹 羽 啓 一

### 1 はじめに

現実の意思決定状況では、階層的な構造をもつ組織に複数の意思決定者 (DM: decision maker) が存在し、それぞれの目的を最適化するように順次、あるいは同時に意思決定を行う状況が考えられる。このような問題は 2 レベル計画問題として定式化される [18]。2 レベル計画問題では、上位レベルの DM が先に決定し、下位レベルの DM は上位レベルの DM の決定を知った上で自己の目的関数を最適化するように決定する。上位レベルの DM はこの仮定の下で自己の目的関数を最適化するように決定する。この決定の組のことを Stackelberg 解と呼ぶ。本論文では、上位レベルと下位レベルにそれぞれ一人の DM が存在し、各 DM の決定変数がすべて 0-1 変数であるような 2 レベル 0-1 計画問題を扱う。

離散変数を含む 2 レベル計画問題に対する研究を概観すると Bard and Moore は 2 レベル 0-1 計画問題 [3] と 2 レベル混合整数計画問題 [2] の Stackelberg 解を導出するために分枝限定法に基づくアルゴリズムを提案している。Wen and Yang [19] は上位レベルの決定変数として 0-1 変数を、下位レベルの決定変数に連続変数をもつ 2 レベル計画問題の Stackelberg 解を得るための計算手法を提案している。

一方、自然界のシステムの適応過程を説明し、生物における進化のメカニズムを模擬する遺伝的アルゴリズム (GA: Genetic Algorithms) は、GA

に関する国際会議やGoldbergの著書[7]によって注目を集め、最適化、適応、学習の方法論として注目を集めている[4, 17]。また、GA は様々な組合せ最適化問題に対して適用され、その有効性が報告されている[15, 16]。

GA を用いた2レベル計画問題の研究として、Anandalingam ら[1]は、2レベル線形計画問題のStackelberg 解を得るための解法を提案している。また、著者ら[10, 11]は、2レベル分権システムに対する0-1計画問題のStackelberg 解を導出するために、坂和らによって提案された2重構造文字列を個体表現に採用した計算手法を提案している。さらに、この計算手法を拡張することで目的関数や制約式に含まれるパラメータのあいまい性がファジィ数[5]として特性づけられるファジィパラメータを含む2レベル計画問題[13, 14]や多目的環境下における2レベル計画問題[9, 12]のStackelberg 解を導出するための計算手法を提案している。

著者らによって提案された計算手法は、全数探索によって得られた厳密なStackelberg 解を得ることが可能であり、また、全数探索を適用することが不可能な問題においても比較的短時間で良好な近似Stackelberg 解を導出することが可能である。しかしながら、この計算手法には、二つの問題点が存在した。その一つは、計算手法に組み込まれているGAの個体群において個体の多様性を維持するような試みがなされていないということである。もう一つは、アルゴリズムの終了条件がGAの打ち切り世代にのみ依存しており、個体群の収束状況に応じた終了条件を計算手法に組み込むことができていないということである。

このような状況の下で、本論文では、2レベル0-1計画問題に焦点をあて、著者らによって提案されたStackelberg 解の計算手法の問題点を改善することで改良型の計算手法を提案する。具体的には、近似Stackelberg 解の精度を高めるために個体群の多様性を保つ効果のあるシェアリングを導入し、さらに個体群の収束状況に応じてアルゴリズムを終了するための条件を追加する。また、上位レベルのDMの決定を扱うGAの個体群内に

重複して発生する  $\mathbf{x}$  に対して繰り返し下位レベルの DM の最適反応  $\mathbf{y}(\mathbf{x})$  を GA によって求めるのではなく、過去に上位レベルの DM の決定として扱った  $\mathbf{x}$  とその最適反応  $\mathbf{y}(\mathbf{x})$  を一定数記憶させておき、後の世代において同一の  $\mathbf{x}$  が現れたときには、記憶させておいた  $\mathbf{y}(\mathbf{x})$  の情報を取り出すことで、下位レベルの最適反応を計算するために適用している GA の回数を減少させるようなアルゴリズムを提案する。提案する手法の有効性を検証するために数値実験を行い、解の精度と計算時間の両面において従来の計算手法と比較、検討する。

## 2 2 レベル 0-1 計画問題

簡略化のために上位レベルの DM と下位レベルの DM をそれぞれ DM1 と DM2 で表す。DM1 と DM2 の目的関数をそれぞれ  $z_1(\mathbf{x}, \mathbf{y})$ ,  $z_2(\mathbf{x}, \mathbf{y})$  とし、 $\mathbf{x}=(x_1, \dots, x_{n_1})^T$ ,  $\mathbf{y}=(y_1, \dots, y_{n_2})^T$  をそれぞれ DM1 と DM2 の決定変数ベクトルとする。ただし、上付添字  $T$  は転置を表す。目的関数の係数ベクトルをそれぞれ  $\mathbf{c}_1=(c_{11}, \dots, c_{1n_1})$ ,  $\mathbf{d}_1=(d_{11}, \dots, d_{1n_2})$ ,  $\mathbf{c}_2=(c_{21}, \dots, c_{2n_1})$ ,  $\mathbf{d}_2=(d_{21}, \dots, d_{2n_2})$  で表し、制約式の係数行列を  $m \times n_1$  行列  $A$ ,  $m \times n_2$  行列  $B$  で表す。制約式の右辺定数ベクトルを  $\mathbf{b}=(b_1, \dots, b_m)^T$  とする。このとき、2 レベル 0-1 計画問題は一般に以下のように定式化される。

$$\left. \begin{array}{ll} \underset{\mathbf{x}}{\text{maximize}} & z_1(\mathbf{x}, \mathbf{y}) = \mathbf{c}_1 \mathbf{x} + \mathbf{d}_1 \mathbf{y} \\ \text{where } \mathbf{y} \text{ solves} & \\ \underset{\mathbf{y}}{\text{maximize}} & z_2(\mathbf{x}, \mathbf{y}) = \mathbf{c}_2 \mathbf{x} + \mathbf{d}_2 \mathbf{y} \\ \text{subject to} & A\mathbf{x} + B\mathbf{y} \leq \mathbf{b} \\ & \mathbf{x} \in \{0, 1\}^{n_1}, \mathbf{y} \in \{0, 1\}^{n_2} \end{array} \right\} \quad (1)$$

本論文では、簡単化のため  $A$ ,  $B$ ,  $\mathbf{b}$ ,  $\mathbf{c}_1$ ,  $\mathbf{c}_2$ ,  $\mathbf{d}_1$ ,  $\mathbf{d}_2$  の各成分は正であると仮定する。

2 レベル 0-1 計画問題において Stackelberg 解が選択される過程は次のように表すことができる。DM1 と DM2 は互いに相手と自己の目的関数

と制約条件を知っており、DM1 は先に決定  $\mathbf{x}$  を選択し、DM2 は DM1 の決定  $\mathbf{x}$  を十分知った上で自己の目的関数を最大にする決定  $\mathbf{y}$  を選択する。すなわち、DM1 の決定を  $\hat{\mathbf{x}}$  とするとき、DM2 は  $\hat{\mathbf{x}}$  をパラメータとした 0-1 計画問題

$$\left. \begin{array}{ll} \underset{\mathbf{y}}{\text{maximize}} & z_2(\hat{\mathbf{x}}, \mathbf{y}) = \mathbf{d}_2 \mathbf{y} + \mathbf{c}_2 \hat{\mathbf{x}} \\ \text{subject to} & B\mathbf{y} \leq \mathbf{b} - A\hat{\mathbf{x}} \\ & \mathbf{y} \in \{0, 1\}^{n_2} \end{array} \right\} \quad (2)$$

を解き、最適解  $\mathbf{y}(\hat{\mathbf{x}})$  を  $\mathbf{x}$  に対する最適反応として選択する。これを前提とし、DM1 も自己の目的関数を最大にする決定  $\mathbf{x}$  を選択する。Stackelberg 解を解の概念として採用する問題では各 DM の間には互いに決定を拘束するような合意が存在しないことが前提となる。言い換えれば彼らの関係は非協力であるといえることができる。

次に、問題(1)の Stackelberg 解を計算する上で重要な概念を定義する。

1) 実行可能領域：

$$S = \{(\mathbf{x}, \mathbf{y}) \mid A\mathbf{x} + B\mathbf{y} \leq \mathbf{b}, \mathbf{x} \in \{0, 1\}^{n_1}, \mathbf{y} \in \{0, 1\}^{n_2}\} \quad (3)$$

2) DM1 が  $\mathbf{x}$  を選択したときの下位レベルの実行可能領域：

$$S(\mathbf{x}) = \{\mathbf{y} \in \{0, 1\}^{n_2} \mid B\mathbf{y} \leq \mathbf{b} - A\mathbf{x}, \mathbf{y} \in \{0, 1\}^{n_2}\} \quad (4)$$

3) DM1 の決定空間：

$$S(X) = \{\mathbf{x} \in \{0, 1\}^{n_1} \mid A\mathbf{x} + B\mathbf{y} \leq \mathbf{b} \text{ をみたす } \mathbf{y} \in \{0, 1\}^{n_2} \text{ が存在する}\} \quad (5)$$

4) DM1 が  $\mathbf{x} \in S(X)$  を選択したときの DM2 の合理的応答集合：

$$R(\mathbf{x}) = \left\{ \mathbf{y} \in \{0, 1\}^{n_2} \mid \mathbf{y} \in \arg \max_{\mathbf{y} \in S(\mathbf{x})} z_2(\mathbf{x}, \mathbf{y}) \right\} \quad (6)$$

5) 誘導領域：

$$IR = \{(\mathbf{x}, \mathbf{y}) \mid (\mathbf{x}, \mathbf{y}) \in S, \mathbf{y} \in R(\mathbf{x})\} \quad (7)$$

6) Stackelberg 解：

$$\{(x, y) \mid (x, y) \in \arg \max_{(x, y) \in IR} z_1(x, y)\} \quad (8)$$

このような 2 レベル 0-1 計画問題(1)の Stackelberg 解を導出するための手法としては、Bard and Moore によって提案された分枝限定法に基づく計算手法や著者らによって提案された遺伝的アルゴリズムに基づく計算手法が存在する。

### 3 Stackelberg 解の計算方法

本節では、2 レベル 0-1 計画問題の Stackelberg 解を導出するために著者らによって提案された遺伝的アルゴリズムに基づく計算手法（簡略化のために以下では、従来法と表記する）について解説し、従来法の問題点を述べた後、遺伝的アルゴリズムに基づく改良型計算手法について提案する。

#### 3.1 遺伝的アルゴリズムに基づく Stackelberg 解の計算方法

ここでは、著者らによって提案された従来法について解説する。従来法は、坂和らによって提案され、0-1 計画問題や整数計画問題の近似解を得るための手法としてその有効性が示されている 2 重構造文字列を個体表現として採用した GA[15]を導入している。しかしながら、この個体表現をそのまま用いただけでは、2 レベル 0-1 計画問題を扱うことができないため、DM1 の決定に対応する部分文字列と DM2 の決定に対応する部分文字列から構成される 2 重構造文字列を用いることとし、それぞれの部分個体ごとに遺伝的操作を行うことで DM1 の決定と DM2 の決定を扱っている。

ここで、DM1 の決定を扱う GA を上位レベル GA と呼び、DM2 の決定を扱う GA を下位レベル GA と呼ぶことにする。

次に、従来法のアルゴリズムについて要約したものを示す。

### 従来法のアルゴリズム

手順 1 上位レベル GA の世代数を  $t: = 0$  とし, 初期個体をランダムに  $N_u$  個生成する。

手順 2 上位レベル GA の各個体  $x$  に対して, 手順 2-1 から手順 2-3 の下位レベル GA の操作を適用し, DM2 の最適反応  $y(x)$  を求める。

手順 2-1 下位レベル GA の個体  $y$  を  $N_l$  個ランダムに生成し, 下位レベル GA の初期個体群とする。

手順 2-2 上位レベル GA の個体として与えられた  $x$  と下位レベル GA によって生成された  $y$  を用いて DM2 の目的関数値を計算し, その値を用いて個体を再生する。

手順 2-3 もし, 現在の反復回数が予め設定された打ち切り世代数  $M_l$  を超えていれば, 手順 3 に進む。そうでなければ, 下位レベル GA の各個体に対して交叉, 突然変異を適用し, 手順 2-2 に戻る。

手順 3 下位レベル GA の操作によって求められた DM2 の最適反応  $y(x)$  と与えられた上位レベル GA の個体  $x$  の組  $(x, y(x))$  を用いて DM1 の目的関数値を計算し, 各個体の適合度の値を求める。

手順 4 上位レベル GA の現在の反復回数が予め設定された打ち切り世代数  $M_u$  に到達すれば, アルゴリズムを終了し, それまでに得られた個体の中で適合度の値が最も大きい個体を最適な個体  $(x, y)$  とみなす。そうでなければ, 手順 5 に進む。

手順 5 手順 3 で求めた上位レベル GA の各個体の適合度の値を用いて再生を行い, さらに交叉と突然変異を適用する。その後,  $t: = t + 1$  として手順 2 に戻る。

上述したアルゴリズムの詳細を以下に示す。

### コーディングとデコーディング

0-1 計画問題を解く際, GA で用いられる個体表現に 2 進文字列がある

[8, 7]。しかしながらこの表現を用いただけでは、制約を満たさない実行不可能な個体を生成する可能性があり、GA の性能を劣化させる恐れがあることが知られている。従来法では、0-1 計画問題の実行可能解のみを生成するために後述するデコーディングアルゴリズムと組み合わせて、図 1 に示される 2 重構造文字列を用いた個体表現を導入している。

← $x$ に関する個体 →			← $y$ に関する個体 →		
$i_x(1)$	...	$i_x(n_1)$	$i_y(1)$	...	$i_y(n_2)$
$S_{ix(1)}$	...	$S_{ix(n_1)}$	$S_{iy(1)}$	...	$S_{iy(n_2)}$

図 1 : 2 重構造文字列を用いた個体表現

ここで、 $S_{ix(m)} \in \{0, 1\}$ ,  $i_x(m) \in \{1, \dots, n_1\}$  とし、 $m \neq m'$  に対して  $i_x(m) \neq i_x(m')$  とする。同様に  $S_{iy(m)} \in \{0, 1\}$ ,  $i_y(m) \in \{1, \dots, n_2\}$  とし、 $m \neq m'$  に対して  $i_y(m) \neq i_y(m')$  とする。また、2 重構造文字列において、 $i_x(m)$ ,  $i_y(m)$  と  $S_{ix(m)}$ ,  $S_{iy(m)}$  はそれぞれ解ベクトルの要素の添字とその値を表す。

しかしながらこの方法を用いただけでは、必ずしも実行可能解のみを生成することができない。そこで、手順 1, 手順 3 および手順 2-2 において、実行不可能な解を除去するために次のような手続きを用いる。

#### 手順 1 と手順 3 における実行可能な $x$ を生成するアルゴリズム

手順 1  $m=1$ ,  $\Sigma=\mathbf{0}$  を設定する。ここで、 $a_{.l}$  は行列  $A$  の  $l$  番目の列ベクトルを表す。

手順 2 もし、 $s_{ix(m)}=1$  なら、 $m=m+1$  とし、手順 3 に進む。 $s_{ix(m)}=0$  ならば、 $m=m+1$  とし、手順 4 に進む。

手順 3 もし、 $\Sigma + \mathbf{a}_{ix(m)} \leq \mathbf{b}$  なら、 $x_{ix(m)}=1$ ,  $\Sigma = \Sigma + \mathbf{a}_{ix(m)}$  に設定する。そうでなければ、 $x_{ix(m)}=0$  と設定する。

手順 4 もし、 $m > n_1$  なら、アルゴリズムを終了し、2 重構造文字列によって表された GA の個体の表現型として  $x$  をみなす。そうでなければ、手順 2 に戻る。

従来法の手順 2-2 については以下に示す手続きを用いる。

#### 手順 2-2 における実行可能な $y$ を生成するアルゴリズム

- 手順 1  $m=1$ ,  $b' = b - Ax$ ,  $\Sigma = 0$  とする。ただし,  $x$  は従来法のアルゴリズムにおける手順 1 もしくは手順 3 で得られた実行可能な  $x$  とする。
- 手順 2 もし,  $s_{iy(m)} = 1$  なら  $m = m + 1$  とし, 手順 3 に進む。 $s_{iy(m)} = 0$  ならば,  $m = m + 1$  とし, 手順 4 に進む。
- 手順 3  $\Sigma + b_{iy(m)} \leq b'$  なら,  $y_{iy(m)} = 1$ ,  $\Sigma = \Sigma + b_{iy(m)}$  と設定する。そうでなければ,  $y_{iy(m)} = 0$  と設定する。ここで,  $b_{iy(m)}$  は行列  $B$  の  $i_y(m)$  番目の列ベクトルを表す。
- 手順 4 もし,  $m > n_2$  なら, アルゴリズムを終了し, 2 重構造文字列によって表された GA の個体の表現型として  $y$  をみなす。そうでなければ, 手順 2 に戻る。

#### 再生

まず, 下位レベル GA の再生方法について述べる。手順 2-2 において, 与えられた上位レベルの決定変数  $x$  の値と下位レベル GA の個体をデコーディングすることで得られた  $y$  の値を DM2 の目的関数  $z_2(x, y)$  に代入することで各個体の関数評価値を得る。その後, 線形スケーリングを用いることで各個体の適合度の値を導出し, さらに, 期待値エリート選択を適用することによって次世代に残す個体を決定する。

次に, 上位レベル GA の再生方法について述べる。手順 1 ならびに手順 3 において, 上位レベル GA の個体をデコーディングすることによって得られた  $x$  の値と下位レベル GA を適用することで得られた  $y(x)$  の値を DM1 の目的関数  $z_1(x, y(x))$  に代入することで各個体の関数評価値を得る。その後, 線形スケーリングを適用することによって各個体の適合度の値を計算し, その値に基づいて期待値エリート選択を適用することによって次世



代に残す個体を決定する。

### 交叉と突然変異

2 重構造文字列の個体に対して、通常の 1 点交叉や多点交叉を行えば、子における添字  $i_x(m)$ ,  $i_x(m')$ ,  $m \neq m'$  または,  $i_y(m)$ ,  $i_y(m')$ ,  $m \neq m'$  が同じ添字番号をもつような実行不可能な個体を発生させる可能性がある。このような問題点は巡回セールスマン問題やスケジューリング問題に対して遺伝的アルゴリズムを適用する際に指摘されており、これを回避する方法として部分一致交叉 (partially matched crossover: PMX) が考案されている。従来法では、坂和らによって提案された 2 重構造文字列を扱うために修正された PMX を用いる。また、交叉オペレータを適用するかどうかを決定する際には、あらかじめ設定された確率  $p_c$  を用いる。

#### PMX の手続き

- 手順 1 2 重構造文字列によって表された二つの個体  $\mathbf{s}_1$ ,  $\mathbf{s}_2$  において、二つの交叉点をランダムに決定する。
- 手順 2 PMX にしたがって、 $\mathbf{s}_1$  と  $\mathbf{s}_2$  の上段の文字列と共に対応する下段の文字列を並べ換え  $\mathbf{s}'_1$  と  $\mathbf{s}'_2$  を生成する。
- 手順 3  $\mathbf{s}'_1$  と  $\mathbf{s}'_2$  の二つの交叉点間の下段の文字列を交換することで 2 重構造文字列において修正された PMX を適用した後の子  $\mathbf{s}''_1$  と  $\mathbf{s}''_2$  を得ることができる。

突然変異オペレータは、遺伝的アルゴリズムにおいて局所的なランダムサーチの役割を担っていると考えられる。2 重構造文字列では、変数の添字列が変数の優先度を表し、また、0-1 変数列は、0-1 変数の値そのものを表していることから、一つの文字列の中に異なる性質の文字列が共存しており、各文字列に対して突然変異を適用する必要がある。従来法では、それぞれの文字列に対して突然変異オペレータを適用しており、添字列に

対しては、逆位を用い、0-1変数列に対しては、ビット反転を導入している。次に、突然変異オペレータを個体に適用する際には、まず、突然変異確率  $p_m$  にしたがってその個体に突然変異が適用されるか否かを決定する。さらに、突然変異が適用される場合については、突然変異選択用定数  $M_{pum}$  にしたがって逆位を適用するかビット反転を適用するかを決定している。

### 突然変異の手続き

- 手順1 2重構造文字列によって表された個体  $s$  に対して、乱数  $r_m$  を発生させる。 $r_m \leq M_{pum}$  であれば、ランダムに0-1変数列の1点を選び、ビット反転を行い、 $s'_1$  を得る。そうでなければ、手順2に進む。
- 手順2 ランダムに添字列の2点を選び、2点間の部分文字列に対して逆位を適用し、 $s'_2$  を得る。

## 3.2 遺伝的アルゴリズムを用いた Stackelberg 解の計算手法の改善

従来法を用いることによって比較的短い時間で比較的良好な近似 Stackelberg 解を導出することが可能になった。しかしながら、従来法にはアルゴリズム上の問題点が複数存在することから、それらを解決することによって近似 Stackelberg 解の精度を改善し、さらに不要な計算を削減することが可能である。

以下では、従来法におけるアルゴリズム上の問題を二点指摘し、それらを検証した後、改善のための方策を導き出す。

1. 上位レベル GA の世代が進むと GA の性質上、個体群内において個体が一つもしくは複数の個体の周辺に収束することが想定される。つまり、個体群内に同じ  $x$  の値を表現型に持つ個体が複数個存在するようになることが想定されるわけであるが、探索の初期にこの現象が生じると個体が局所解に収束し、最適解を得ることができないといった

問題が起きる。また、従来法のアルゴリズムでは、重複して生じている  $\boldsymbol{x}$  に対して、繰り返し、最適反応  $\boldsymbol{y}(\boldsymbol{x})$  を求めており、不要な計算を行っているといえる。

2. 手順 2-3 や手順 4 で示した通り、従来法では、アルゴリズムの終了判定が GA の打ち切り世代数のみに依存しており、個体の収束状況に対応した終了判定が導入されていない。

これらの問題については、特に上位レベル GA の個体群が世代を経るごとにどのように推移しているのかを検証する必要がある。

問題点の検証

問題点を検証するために表 1 に示すような規模で制約の本数が 5 本であるような 2 レベル 0-1 計画問題を用意する。ここで、2 レベル 0-1 計画

表 1：数値実験で用いる問題の規模と制約の強さ

問題	DM1 が扱う変数	DM2 が扱う変数	制約の強さ
A	5	5	I (50%)
			II (70%)
			III (90%)
B	10	10	I (50%)
			II (70%)
			III (90%)
C	15	15	I (50%)
			II (70%)
			III (90%)
D	20	20	I (50%)
			II (70%)
			III (90%)

問題の  $A, B, \boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{d}_1, \boldsymbol{d}_2$  の各成分は  $[10, 99]$  の整数の乱数を発生させることで値を設定し、 $\boldsymbol{b}$  の成分  $b_i$  については、

$$b_i = r_i \left( \sum_{j=1}^{n_1} a_{ij} + \sum_{k=1}^{n_2} b_{ik} \right), i=1, \dots, m \quad (9)$$

にしたがって値を設定する。ただし、 $a_{ij}$  は行列  $A$  の  $ij$  成分を表し、 $b_{ik}$  は行列  $B$  の  $ik$  成分を表す。ここで、 $r_i$  は制約の強さの度合いを表しており、制約を強く設定した問題 (I) については  $[0.45, 0.55]$  の乱数の値を発生させ、制約を中程度に設定した問題 (II) については、 $[0.65, 0.75]$  の値を発生させる。また、制約を弱く設定した問題 (III) については、 $[0.85, 0.95]$  の乱数の値を発生させる。 $b_i$  の値は小数点以下を四捨五入し、整数値として設定する。また、問題 A から D については係数値の異なる問題を 3 種類ずつ用意する。

次に、これらの問題の Stackelberg 解を導出するために従来法を適用する。その際、上位レベル GA と下位レベル GA のパラメータを個体群サイズ 100, 打ち切り世代数 100, 交叉率 0.7, 突然変異率 0.01 に設定する。また、数値実験には 1.40 GHz の CPU を搭載し、Windows XP を OS とした

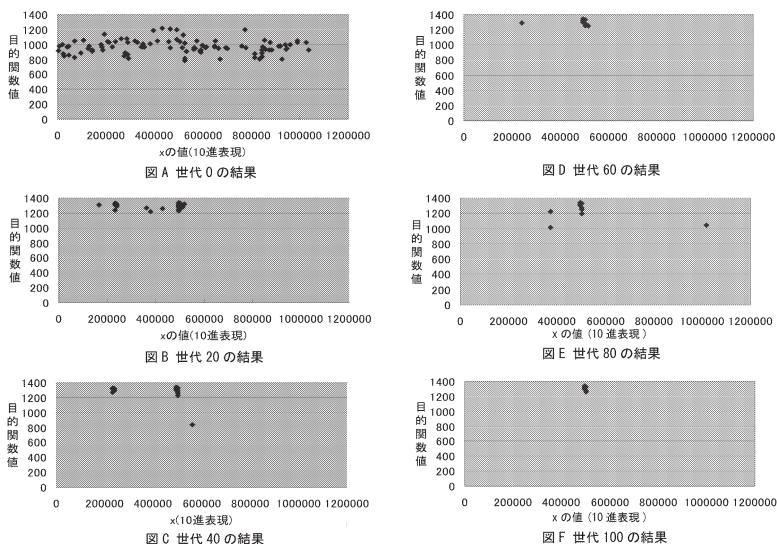


図 2：個体と目的関数値の分布図

コンピュータを用い、コンパイラとして Visual C++ 6.0を採用する。シミュレーションの試行回数は、各問題に対して10回ずつとする。

得られた結果の代表的な例として問題D-Iの実験結果を図2に示す。図の横軸は、上位レベルGAの個体を表しており、上位レベルGAの個体の表現型  $x$  を2進数の数値として捉え、その値を基数変換によって10進表現化したものである。また、図の縦軸はDM1の目的関数値を表す。

図2-Aの初期個体群では、多様な個体が存在するのに対し、20世代を経た段階の個体群の状態を表す図2-Bにおいては、個体が5つのグループに収束していることがわかる。さらに、40世代経過後の図2-Cでは、一つの個体の例外を除いて二つのグループに収束が進み、60世代、80世代と経過するごとに100世代経過時に得られている最良解の周辺に多くの個体が収束していることがわかる。

この結果より、上位レベルGAの打ち切り世代よりも前の世代で上位レベルGAの個体群が一定の個体に収束していることがわかる。このことから問題点1と2が実際のアルゴリズムで生じていることがいえる。

そこで、まずは、問題点1で指摘している個体群に属する個体の収束について考慮する。GAでは、適合度の値に基づいて次世代に残す個体を再生するため、世代が進むにつれて個体群中に適合度が高く、似通った構造の遺伝子をもつ個体が多く存在するようになる。この性質は、探索の後期においてGAの局所探索の能力を高める効果を発揮するのだが、探索の初期においては、初期収束を引き起こし、個体群の多様性を維持できず、良好な解を導出できないといった問題を生じさせる。そこで、個体群内の個体の多様性を保ち、初期収束を起こさないという目的からGoldbergら[6]によって提案されたシェアリングを導入する。シェアリングは多くの点が集中している部分における関数値には比較的小さな重みをかけるのに対して、孤立している点における関数値には相対的に大きな重みをかけることにより、関数値の分布の相対的な均一化を図る操作である。数学的にはコード化された文字列  $s_i$ ,  $s_j$  間の距離  $d_{ij}=d(s_i, s_j)$  を考えることにより

$$(1) \quad 0 \leq sh(d) \leq 1, \forall d \in [0, \infty)$$

$$(2) \quad sh(0) = 1$$

$$(3) \quad \lim_{d \rightarrow \infty} sh(d) = 0$$

を満たす関数  $sh(\cdot)$  のことをシェアリング関数と呼び, Goldberg らは特に式(10)で表されるようなシェアリング関数の導入を推奨している。

$$sh(d) = \begin{cases} 1 - \left( \frac{d}{\sigma_{\text{share}}} \right)^\alpha & d < \sigma_{\text{share}} \text{ のとき} \\ 0 & \text{その他} \end{cases} \quad (10)$$

ここで,  $\sigma_{\text{share}}$  と  $\alpha$  は定数である。

個体  $\mathbf{s}_i$  の適合度関数を  $f(\mathbf{s}_i)$ , 個体群サイズを  $N$  として, 距離とシェアリング関数が選ばれると, シェアリングにより次のように適合度を修正する。

$$f(\mathbf{s}_i) := \frac{f(\mathbf{s}_i)}{m_i} \quad (11)$$

$$m_i = \sum_{j=1}^N sh(d(\mathbf{s}_i, \mathbf{s}_j)) \quad (12)$$

提案する改良型の計算手法では, 式 (10)のシェアリング関数を採用し, 個体間の距離  $d_{ij}$  の計算にハミング距離[17]を用いる。ただし, 個体間の距離を計算する際には, 遺伝子型の個体ではなく表現型の個体を用いる。この理由としては, 個体の適合度の値を算出する際に表現型の個体を用いていることから表現型の個体に対してシェアリングを導入した方が個体群の多様性を保つ効果が高いと見込まれるからである。

次に, 上位レベル GA において扱う個体  $\mathbf{x}$  が同じ世代の個体群内に複数存在する場合や異なる世代の個体群内に再び同じ個体が出現した場合に下位レベル GA を用いて繰り返し同じ問題(2)を解き, 同じ最適反応  $\mathbf{y}(\mathbf{x})$  を求めている点について考慮する。

これを回避するためには, 単純に考えると上位レベル GA の操作で得られたすべての  $\mathbf{x}$  に対して, 下位レベル GA で求めた最適反応  $\mathbf{y}(\mathbf{x})$  を記憶

させておき、過去に扱ったことのある  $x$  が個体として現れた際には記憶させておいた最適反応  $y(x)$  を取り出すことで下位レベル GA を適用せずに済ませることができる。しかしながら、問題の規模が増大すると上位レベル GA で扱ったすべての  $x$  に対して最適反応の情報を記憶させておくことはメモリの容量の制約や保存してある  $x$  を検索するために要する時間上の制約から困難であるといえる。そこで、図 3 のようなデータ構造の記憶領域を設け、 $x$  に対する最適反応  $y(x)$  の一部を記憶させる。

ここで、 $x^i, i=1, \dots, x\_max$ , は過去に上位レベル GA の個体として扱われた  $x$  の値を表し、 $y_j^i, i=1, \dots, x\_max, j=1, \dots, y\_max$  は下位レベル GA によって求めた  $x^i$  に対する最適反応の値を表す。 $x\_max$  は保存しておく DM1 の決定  $x$  の数を表し、 $y\_max$  は保存しておく DM2 の最適反応  $y(x)$  の数を表す。また、 $z_1(x^i, y_j^i(x^i))$  と  $z_2(x^i, y_j^i(x^i))$  は保存してある  $x^i$ ,

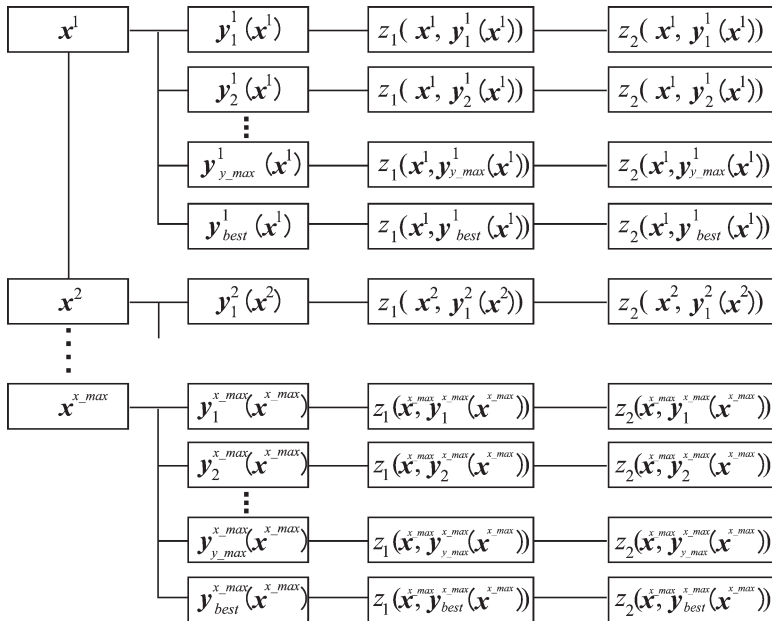


図 3 :  $x$  と  $y(x)$  を保存するための記憶領域

$y_j^i(x^i)$  を DM1 と DM2 の目的関数に代入した値を表し、 $y_{best}^i(x^i)$  は  $z_1(x^i, y_j^i(x^i))$  の値を最も大きくするような  $y_j^i(x^i)$  を表す。

さらに次に示すようなアルゴリズムを用いることによって、下位レベル GA の適用回数を減少させ、不要な計算を削減する。

#### 最適反応 $y(x)$ の保存と下位レベル GA の回避手続き

- 手順 1 上位レベル GA の個体  $\bar{x}$  が保存してある  $x^i$  に存在すれば手順 2 に進む。存在しなければ、さらに  $x^i$  の数が  $x\_max$  に達しているかどうかを確認し、達していれば手順 3 に進む。達していなければ、手順 4 に進む。
- 手順 2  $y_j^i$  が  $y\_max$  に達していれば、保存してある  $y_{best}^i(x)$  を最適反応として上位レベル GA に返し、アルゴリズムを終了する。達していなければ、手順 4 に進む。
- 手順 3 保存してある  $x^i$  の中で  $z_1(x^i, y_{best}^i(x^i))$  の値が最も小さいものを選択し、その  $x$  の値を  $x^k$  とする。下位レベル GA を適用することで  $\bar{x}$  に対する最適反応  $y(\bar{x})$  を得た後、 $x^k$  の記憶領域にそれらを保存し、アルゴリズムを終了する。
- 手順 4 下位レベル GA を適用することによって  $\bar{x}$  に対する最適反応  $y(\bar{x})$  を得た後、それらの値を保存し、アルゴリズムを終了する。

問題点 2 のアルゴリズムの終了条件については、次のような方法を用いて対処することができる。上述したように問題点 1 を解決するための手法としてシェアリングを導入した。その際、個体群に属するすべての個体間の距離  $d_{ij}$  を算出していることから個体の収束度合いを世代ごとに測ることが可能になった。そこで、この距離を用いて、ある一つの個体が個体群全体の個体に占める割合を計算し、その割合が  $c_e$  以上になった場合、個体群がある一つの個体に収束したものと判断し、アルゴリズムを終了するという方法である。改良型の計算手法では打ち切り世代による終了条件に加



えてこの終了条件も採用する。

このような手法を取り入れることで従来法のアルゴリズムを改良する。  
改良後の計算手法のアルゴリズムを要約して以下に示す。

### Stackelberg 解の改良型計算手法のアルゴリズム

手順 1 上位レベル GA の世代数を  $t:=0$  とし、初期個体をランダムに  $N_u$  個生成する。

手順 2 上位レベル GA の各個体  $x$  に対して、下位レベル GA の回避手続きを実施するか否かを判定する。実施する場合は保存してある下位レベルの  $y(x)$  を最適反応とし、手順 3 に進む。実施しない場合は手順 2-1 から手順 2-3 の下位レベル GA の操作を適用し、下位レベルの最適反応  $y(x)$  を求める。

手順 2-1 下位レベル GA の個体  $y$  を  $N_l$  個ランダムに生成し、下位レベル GA の初期個体群とする。

手順 2-2 上位レベル GA の個体として与えられた  $x$  と下位レベル GA によって生成された  $y$  を用いて DM2 の目的関数値を計算し、その値を用いて個体を再生する。

手順 2-3 現在の反復回数が予め設定された打ち切り世代数  $M_l$  を超えているか、もしくは、個体の収束度合いが予め設定された  $C_{el}$  を超えた場合、手順 3 に進む。そうでなければ、下位レベル GA の各個体に対して交叉、突然変異を適用し、手順 2-2 に戻る。

手順 3 下位レベル GA の操作によって求められた下位レベルの最適反応  $y(x)$  と与えられた上位レベル GA の個体  $x$  の組  $(x, y(x))$  に関する情報を保存し、さらに、それらの値を用いて DM1 の目的関数値を計算し、各個体の適合度の値を求める。

手順 4 上位レベル GA の現在の反復回数が予め設定された打ち切り世代数  $M_u$  に到達しているか、もしくは、個体の収束度合いが予め設定

された値  $C_{eu}$  を超えていれば、アルゴリズムを終了する。その際、それまでに得られた個体の中で適合度の値が最も大きい個体を最適な個体  $(x, y)$  とみなす。そうでなければ、手順5に進む。

手順5 手順3で求めた上位レベルGAの各個体の適合度の値を用いて再生を行い、さらに交叉と突然変異を適用する。その後、 $t:=t+1$  として手順2に戻る。

#### 4 数値例

従来法と改良型の計算手法を解の精度と計算時間の面で比較するために数値例を示す。数値例には3.2節で述べた問題の一つを用い、従来法と改良型の計算手法に含まれるGAのパラメータを次のように設定する。まず、従来法における上位レベルGAと下位レベルGAのパラメータを個体群サイズ100、交叉率0.9、突然変異率0.02、打ち切り世代数100に設定する。次に、改良型の計算手法においても個体群サイズ、交叉率、突然変異率、打ち切り世代数の各パラメータは従来法と同じ値に設定する。シェアリングの定数  $\sigma_{share}$  をDM1もしくはDM2が扱う変数の10%に該当する整数値に設定する。ただし、10%の値が整数にならない場合、小数点以下

表2：解の精度の比較

問題	制約の 強さ	改良型の計算手法				従来法				全数 探索
		最良値	最悪値	平均値	分散	最良値	最悪値	平均値	分散	
A	I	250	250	250.0	0.00	250	250	250.0	0.00	250
	II	323	323	323.0	0.00	323	323	323.0	0.00	323
	III	404	404	404.0	0.00	404	404	404.0	0.00	404
B	I	650	650	650.0	0.00	650	650	650.0	0.00	650
	II	877	877	877.0	0.00	877	877	877.0	0.00	877
	III	1071	1071	1071.0	0.00	1071	1071	1071.0	0.00	1071
C	I	1078	1078	1078.0	0.00	1078	1078	1078.0	0.00	1078
	II	1377	1377	1377.0	0.00	1377	1377	1377.0	0.00	1377
	III	1727	1727	1727.0	0.00	1727	1724	1726.4	1.44	1727
D	I	1251	1251	1251.0	0.00	1251	1218	1246.2	108.36	—
	II	1570	1559	1561.2	13.96	1570	1559	1564.2	20.36	—
	III	1909	1896	1907.7	15.21	1909	1876	1905.7	98.01	—

を四捨五入する。また、 $\alpha$  を 0.25 に設定する。さらに、ある世代において、個体群全体におけるある一つの個体の占める割合が 90% 以上に達した場合、個体群は収束したものとして判断し、GA の操作を打ち切る。この手続きを実施するために  $C_{eu}$  および  $C_{el}$  の値を 0.9 に設定する。最後に、上位レベル GA にのみ設定する必要のあるパラメータについて述べる。そのパラメータとして過去に上位レベル GA で扱った DM1 の決定  $\mathbf{x}^i$  を保存しておく数  $x\_max$  と  $\mathbf{x}^i$  に対する最適反応  $\mathbf{y}_j^*(\mathbf{x}^i)$  を保存しておく数  $y\_max$  がある。これらの値をそれぞれ 100 と 5 に設定する。

次に実験環境について説明する。実験には 2.80GHz の CPU を搭載し、OS を Windows XP としたパソコンを用いた。また、コンパイラとして Microsoft Visual C++ 6.0 を使用した。

試行回数については、問題ごとに従来法および改良型の計算手法を 10 回ずつ実行した。その結果を表 2 から表 5 に示す。

表 2 より 30 変数以下の規模の問題では、従来法、改良型の計算手法ともにすべての試行において最適な Stackelberg 解を導出できていることがわかる。また、40 変数の規模の問題においても改良型の計算手法は従来法と同じ最良解を得られているだけでなく、分散の値が小さいことから安定

表 3：計算時間の比較

問題	制約の 強さ	改良型の計算手法				従来法				全数 探索
		最大値	最小値	平均値	分散	最大値	最小値	平均値	分散	
A	I	8.34	8.00	8.22	0.01	303.67	296.78	301.03	6.46	0.00
	II	14.50	14.36	14.44	0.00	300.59	299.94	300.20	0.03	0.00
	III	15.45	15.30	15.39	0.00	301.47	301.13	301.30	0.01	0.00
B	I	1019.83	969.83	993.10	271.29	499.52	498.50	499.06	0.11	0.45
	II	1190.97	1143.06	1161.87	207.32	496.25	494.83	495.23	0.15	0.50
	III	1177.58	1119.25	1153.64	311.28	501.41	496.45	497.28	2.06	0.52
C	I	1921.96	1865.88	1893.02	285.63	699.19	688.11	696.48	8.93	566.73
	II	2001.53	1977.30	1991.26	42.29	692.08	683.81	688.76	4.42	652.38
	III	1997.99	1973.99	1981.32	55.02	693.31	689.44	690.78	2.01	665.81
D	I	2536.64	2473.38	2513.73	345.91	893.38	889.45	891.19	2.03	—
	II	2614.17	2593.69	2602.37	32.14	907.33	895.58	899.00	14.98	—
	III	2489.74	2464.50	2474.97	41.02	909.74	900.42	903.03	8.87	—

表 4：下位レベル GA の回避回数

問題	制約の 強さ	改良型の計算手法			
		最大値	最小値	平均値	分散
A	I	9985	9985	9985.0	0.00
	II	9945	9945	9945.0	0.00
	III	9940	9940	9940.0	0.00
B	I	3123	2831	2949.5	7919.85
	II	2438	2120	2313.3	9262.61
	III	2509	2117	2288.6	13721.44
C	I	484	254	389.3	4404.61
	II	345	235	289.6	912.04
	III	246	167	210.1	813.29
D	I	128	64	83.8	461.96
	II	86	11	60.7	379.81
	III	259	133	184.8	1331.96

して良好な解を得ていることがわかる。次に、表 3 の計算時間の結果から改良型の計算手法は従来法に比べて問題の規模が大きくなるにつれて急速に計算時間が増大していることがわかる。このことは問題 A から問題 C において顕著である。これは、表 4 の結果から明らかなように変数が増えるにつれて下位レベル GA を回避した回数が減少しているために生じている現象であると考えられる。また、改良型の計算手法において多くの計算時間がかかっている理由として、次の二点が挙げられる。一点目はシェアリングの導入によって世代ごとに全個体間の距離を計算する必要が生じたことであり、もう一点は上位レベル GA の個体  $x$  が記憶領域に保存されているかどうかを検索する必要が生じたためであると考えられる。最後に、表 4 の結果より、問題の規模が大きくなるにつれて、下位レベル GA の回避回数が減少していること、ならびに表 5 の結果より、改良型の計算手法は従来法よりも最良解を導出した世代が遅いことからシェアリングの導入によって上位レベル GA の個体群の多様性が保たれ、急速な個体群の収束を避けることが可能になったと考えられる。逆に、個体群の多様性が保たれるようになったことから下位レベル GA を回避する手続きの効果が薄まったものとみなすことができる。

表 5：最良解が得られた世代の比較

問題	制約の強さ	改良型の計算手法				従来法				全数探索
		最大値	最小値	平均値	分散	最大値	最小値	平均値	分散	
A	I	0	0	0.0	0.00	0	0	0.0	0.00	—
	II	0	0	0.0	0.00	0	0	0.0	0.00	—
	III	0	0	0.0	0.00	0	0	0.0	0.00	—
B	I	7	0	3.8	4.36	4	0	3.0	1.80	—
	II	17	3	6.9	24.09	6	2	3.7	2.01	—
	III	14	0	6.1	15.89	8	0	3.7	5.41	—
C	I	67	1	25.8	335.76	52	6	18.6	253.44	—
	II	51	8	29.5	180.45	15	5	9.0	8.40	—
	III	65	1	31.8	518.96	14	4	8.7	7.81	—
D	I	75	19	40.0	244.20	41	7	19.3	91.21	—
	II	100	21	43.0	581.80	93	6	35.3	794.41	—
	III	89	18	47.3	534.61	31	11	17.2	33.96	—

## 5 おわりに

本論文では、二人の DM が相互に独立で、協力する動機がなく、彼らの決定には順序があり、後で決定する DM は先に決定した DM の決定を十分認識した上で自己の決定を行うような 2 レベル 0-1 計画問題に焦点をあて、著者らによって提案されていた Stackelberg 解の計算手法の問題点を改善することで改良型の計算手法を提案した。具体的には、個体群の多様性を維持し、導出する解の精度を高めるためにシェアリングを導入し、さらに個体群の収束状況に応じてアルゴリズムを終了するための条件を追加した。また、上位レベル GA の個体群内に重複して発生した DM1 の決定  $x$  に対して繰り返し下位レベル GA を適用することで最適反応  $y(x)$  を求めるのではなく、過去に上位レベル GA で扱った  $x$  とその最適反応  $y(x)$  を一定数記憶させておき、後の世代において同一の  $x$  が現れたときには、記憶させておいた  $y(x)$  の情報を取り出すことで、下位レベル GA の適用回数を減少させるアルゴリズムを提案した。提案した手法の有効性を検証するために数値実験を行い、解の精度と計算時間の両面で従来法と比較し、解の精度の点では比較的良好な結果を得ることができた。

## 参 考 文 献

- [1] G. Anandalingam, R. Mathieu, C.L. Pittard, and N. Sinha: "Artificial intelligence based approaches for solving hierarchical optimization problems," in: Sharda, Golden, Wasil, Balci and Stewart (eds.), *Impacts of Recent Computer Advances on Operations Research*, North-Holland, pp. 289-301 (1989).
- [2] J. Bard and J. Moore: "The mixed integer linear bilevel programming problem," *Operations Research*, Vol. 38, pp. 911-921 (1990).
- [3] J. Bard and J. Moore: "An algorithm for the discrete bilevel programming problem," *Naval Research Logistics*, Vol. 39, pp. 419-435 (1992).
- [4] L. Davis (ed.): *Handbook of Genetic Algorithms*, Van Nostrand Reinhold (1991). 嘉数也訳, 遺伝的アルゴリズムハンドブック, 森北出版 (1994).
- [5] D. Dubois and H. Prade: *Fuzzy Sets and Systems: Theory and Applications*, Academic Press, New York (1980).
- [6] D.E. Goldberg and J. Richardson: "Genetic algorithms with sharing for multimodal function optimization," *Proceedings of the Second International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, pp. 41-49 (1987).
- [7] D.E. Goldberg: *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison Wesley, Massachusetts (1989).
- [8] D.E. Goldberg and R. Lingle: "Alleles, loci, and the traveling salesman problem," *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, Lawrence Erlbaum Associates, Hillsdale, NJ, pp. 154-159 (1985).
- [9] K. Niwa, K. Kato, M. Sakawa and I. Nishizaki: "Interactive fuzzy programming using Partial information about preference through genetic algorithms for multiobjective two-level integer programming problems", *Joint 2nd International Conference on Soft Computing and Intelligent Systems and 5th International Symposium on Advanced Intelligent Systems Conference Proceedings*, (2004).
- [10] 丹羽, 西崎, 坂和: "遺伝的アルゴリズムを用いた2レベル分権システムに対する0-1計画法", 日本ファジィ学会誌, Vol. 10, No. 4, pp. 735-742 (1998).
- [11] 丹羽, 西崎, 坂和: "分権的2レベル0-1計画問題に対する遺伝的アルゴリズムを用いたStackelberg-Nash解の計算方法", 日本ファジィ学会誌, Vol. 11, No. 5, pp. 808-815 (1999).
- [12] K. Niwa, I. Nishizaki and M. Sakawa: "Multiobjective two-level zero-one programming problems through genetic algorithms", *Proceedings of the Second*

- Asia-Pacific Conference on Industrial Engineering and Management Systems*, pp. 493-496 (1999).
- [13] K. Niwa, I. Nishizaki and M. Sakawa: "Computational methods for two-level 0-1 programming problems with fuzzy parameters through genetic algorithms", *1999 IEEE International Fuzzy Systems Conference Proceedings*, Vol. 3, pp. 1510-1515 (1999).
- [14] K. Niwa, I. Nishizaki and M. Sakawa: "Computational methods for two-level integer programming Problems with fuzzy parameters through genetic algorithms", *Proceedings of KES'02*, Vol. 2, ed. by L. C. Jain and R. K. Jain, pp. 1247-1251 (2002).
- [15] 坂和, 加藤, 砂田, 園田: "改良型遺伝的アルゴリズムによるファジィ多目的組合せ最適化," 日本ファジィ学会誌, Vol. 6, pp. 177-186 (1994).
- [16] 坂和, 加藤, 砂田, 園田: "改良型遺伝的アルゴリズムによる対話型ファジィ満足化手法," 日本ファジィ学会誌, Vol. 7, pp. 361-370 (1995).
- [17] 坂和, 田中: 遺伝的アルゴリズム, 朝倉書店(1995).
- [18] K. Shimizu, Y. Ishizuka and J. F. Bard: *Nondifferentiable and two-level mathematical programming*, Kluwer Academic Publishers (1997).
- [19] W. P. Wen, and Y. H. Yang: "Algorithms for solving the mixed integer two-level linear programming problem", *Computers and Operations Research*, Vol. 17, pp. 133-142 (1990).