動的計画法複合アルゴリズムと 最適低推力ミッション軌道生成への応用†

花岡 照明*

A Hybrid Dynamic Programming Algorithm and Its Application to Optimal Low-Thrust Mission Trajectory Generation

Teruaki HANAOKA*

A hybrid dynamic programming algorithm is proposed, which is based on the forward dynamic programming combined with the branch-and-bound technique. This hybrid algorithm solves nonlinear optimal control problems with complicated state and control constraints several orders faster than the conventional dynamic programming. This method overcomes the Bellman's the "curse of dimensionality". Basically, the proposed algorithm consists of interactions similar to Darwinian evolution through the cycles of selection, multiplication through the state space quantization.

The most important idea is that this algorithm successively improves not only the optimal solution but also the accuracy of the lower bounds of the cost function which have great influence upon the number of computations as well as computation regions. To this end, the hybrid algorithm is carried out with the successive iterations under the further quantization of each state and control variable. The second idea is that this algorithm is carried out with parallel computations starting from the approximated local solutions obtained through each iteration step.

This paper assesses convergence and accuracy of the hybrid algorithm as well as the number of computations. An application of this algorithm is studied for the optimal low-thrust mission trajectory generation, in which, the Sun encounter mission is considered, and only Venus is taken as the swung-by planet if some swingby is requested. The results show that the proposed hybrid algorithm is quite efficient for the large-scale problems with the complicated constraints such as the Sun encounter mission.

Key Words: dynamic programming, optimal control, branch-and-bound, low-thrust, swingby

1. はじめに

今日、航空宇宙をはじめ、多くの分野で、システムに対する要求は複雑かつ多様化してきている。一方、これらの要求を含んだ複雑な問題を容易に扱うことのできる計算法の開発が強く望まれている。

本論文では、複雑な非線形最適制御問題を統一的に解く計算法について考察する。

動的計画法¹⁾は、基本的には、どのような複雑なシステ

ムに対しても数値的に取り扱うことのできる手段としてよく知られている。さらに、動的計画法を用いると、大域的最適解(以下、大域解と表記)やフィードバック構造をもつ解が得られること、制約条件や非線形の取扱いが容易であること、状態の遷移やコスト関数の表現が表関数であってもよいなど、数多くの望ましい性質を利用できる。しかしながら、動的計画法は、「Bellmanの次元の呪い」の問題²⁾、すなわち、状態変数の次元数の増加に伴って計算数が指数関数的に増加し、特別な問題を除き、計算実行上、求解が困難、という致命的な欠陥を持つため、次元の高い問題への適用は、きわめて難しいとされてきた。

動的計画法の実行で必要となる計算数とメモリーサイズを低減させる研究として、様々な手法が提案されてい

† 第31回宇宙科学技術連合講演会で発表(1987・10)

* 東京大学工学部 東京都文京区本郷 7-3-1

* Faculty of Engineering, University of Tokyo, Bunkyo-ku, Tokyo

(Received July 25, 1994)

(Revised September 6, 1995)

る²⁾。Alekseev³⁾とMorin and Marsten⁴⁾は、離散系の問題に対し、最適解の候補になり得ない状態を削除するために動的計画法と分枝限定法を併用した。一方、連続系の問題に対する主要な技法として、Larson and Korsak⁵⁾の(state) incremental dynamic programming法とJacobson and Mayne⁶⁾のdifferential dynamic programming法が報告されている。前者の問題点は、真の解への収束を保証しないことと、状態遷移関数が、制御変数に関し可逆関数であることを要求することである。一方、後者では、初期制御方策の仮定を必要とし、解は局所的最適解(以下、局所解と表記)となるという問題があることや、微係数計算などの計算負荷の問題もあるとされている。他方、Woźniakowski⁷⁾は、多変数積分問題での次元の呪いを解決する方法として、「平均時基準」の概念を導入することで、計算可能になることを理論的に示している。Hanaoka and Tanabe⁸⁾およびHanaoka⁹⁾では、再突入飛行体の軌道最適化や航空機の経路最適化などの実際の連続系に対して、動的計画法と分枝限定法を併用した、動的計画法複合アルゴリズム(以下、基本形複合アルゴリズムと表記)を適用し、計算数を大幅に削減させることに成功している。その中では、アルゴリズムの実行で必要となるコストの下界値を、システムの特長性を利用して計算する方法が提案されている。

今回の複合アルゴリズムでは、この基本形複合アルゴリズムに対し、計算上の工夫として、コストの下界値を精度よく推定する繰り返し論理と、局所解を避ける並列処理を組み入れた算法(以下、拡張複合アルゴリズムと表記)を提案する。それらによって数値計算上の負荷(計算数とサイズ)を大幅に削減し、局所解を回避することができる。繰り返し論理では、計算数を左右する下界値の推定精度を向上させている。一方、並列処理は、複数の最適経路候補を効率的に取り扱い、局所解へ陥ることを避けている。

拡張複合アルゴリズムは、ダーウィン進化、すなわち、淘汰、増殖、突然変異の3つの繰り返し過程、の一部をまねたような算法であり、従来型動的計画法¹⁾とは異なった構成法に基づいている。淘汰過程は、後述の「ブロックを使った優先順位計算」に、増殖過程は、量子化した制御の適用による「経路群の繰り返し生成」に対応する。突然変異の組入れは、本稿では扱わない。本論文で述べる拡張複合アルゴリズムの計算メカニズムは、分散的に、ばらばらに存在する経路群を、最適な経路へと集約させるもので、その理論的根拠を前向き動的計画法^{4),8)}に置いている。

本論文では、拡張複合アルゴリズムの計算数、収束性、

解の精度について調べている。また、実際問題への本法の適用性を調べるため、金星とのスウィングバイを考慮した、低推力太陽探査軌道の生成問題が考察されている。

2. 最適制御問題と動的計画法

いま、制御システムは、

$$\mathbf{x}_{k+1} = \mathbf{g}_k(\mathbf{x}_k, \mathbf{u}_k) \quad (k=0, 1, 2, \dots, N) \quad (1)$$

で記述されるものとする。ここで、 $\mathbf{x}_k$ と $\mathbf{u}_k$ を、それぞれ、 n 次元状態ベクトルおよび m 次元制御ベクトルとする。また、 k を段変数の添え字、 $\mathbf{g}_k$ を n 次元ベクトル値関数とする。このとき、最小化したい評価関数を

$$J = \sum_{i=0}^{N-1} L_i(\mathbf{x}_i, \mathbf{u}_i) + \Phi_N(\mathbf{x}_N) \quad (2)$$

とすれば、最適制御問題は

$$\text{Minimize } J = \sum_{i=0}^{N-1} L_i(\mathbf{x}_i, \mathbf{u}_i) + \Phi_N(\mathbf{x}_N) \quad (3)$$

$$\text{Subject to } \mathbf{x}_{k+1} = \mathbf{g}_k(\mathbf{x}_k, \mathbf{u}_k) \quad (k=0, \dots, N-1) \quad (4)$$

$$\mathbf{x}_0 = \mathbf{c}_0 \quad (5)$$

$$\mathbf{x}_N \in \Omega_F \quad (6)$$

$$\mathbf{x}_k \in X_k \quad (k=1, \dots, N-1) \quad (7)$$

$$\mathbf{u}_k \in U(\mathbf{x}_k) \quad (k=0, \dots, N-1) \quad (8)$$

となる。ただし、 $L_k(\mathbf{x}_k, \mathbf{u}_k)$ ($k=0, 1, \dots, N-1$)を1段当りのコスト関数、 $\Phi_N(\mathbf{x}_N)$ を最終段のコスト関数とする。また、 X_k と $U(\mathbf{x}_k)$ を、それぞれ k 段での許容状態集合と許容制御集合とする。(5)と(6)式は、それぞれ、初期条件と終端条件であり、 Ω_F ($\Omega_F \subset X_N$)を終端条件を満たす集合とする。

上記の問題を従来型(後向き)動的計画法¹⁾を用いて解くには、各段 k ($k=0, 1, \dots, N$)でのすべての許容状態 $\mathbf{x}_k \in X_k$ に対して動的計画法の再帰関数方程式を適用し、それらの状態 $\mathbf{x}_k$ の最適制御 $\mathbf{u}_k^*$ を計算する。この再帰関数方程式は、 k 段以後の最小コスト関数を

$$J_k(\mathbf{x}_k) = \min_{\mathbf{u}_k, \mathbf{u}_{k+1}, \dots, \mathbf{u}_{N-1}} \left\{ \sum_{i=k}^{N-1} L_i(\mathbf{x}_i, \mathbf{u}_i) + \Phi_N(\mathbf{x}_N) \right\}$$

と定義すると、Bellmanの最適性の原理¹⁾から、

$$J_N(\mathbf{x}_N) = \Phi_N(\mathbf{x}_N)$$

$$J_k(\mathbf{x}_k) = \min_{\mathbf{u}_k \in U(\mathbf{x}_k)} \{ L_k(\mathbf{x}_k, \mathbf{u}_k) + J_{k+1}(\mathbf{g}_k(\mathbf{x}_k, \mathbf{u}_k)) \} \quad (k=0, \dots, N-1) \quad (9)$$

となる。この(9)式から最適方策を解析的に求めることは多くの場合難しいので、通常、数値的に解くことになる。

(9)式を解くには、通常、各状態変数 x_i^j ($i=1, 2, \dots, n$)を l_x レベルに、また、各制御変数 u_i^j ($j=1, 2, \dots, m$)を l_u レベルに量子化する。そして、状態ベクトルの n 次元格子の各点において量子化した制御を適用し、(9)式を用

いて格子点での $J_k(\mathbf{x}_k)$ と最適制御 $\mathbf{u}_k^*$ を求める。もし、この計算で、次段の状態 $\mathbf{g}_k(\mathbf{x}_k, \mathbf{u}_k)$ が格子点上に無い場合は、格子点での $J_{k+1}(\mathbf{x}_{k+1})$ を用い、内挿によって $J_{k+1}(\mathbf{g}_k(\mathbf{x}_k, \mathbf{u}_k))$ を計算する。状態変数の次元数 n が大きいとき、この内挿計算は相当煩わしい。(9)式の計算は $k=N-1$ から始め、後向きに、 $k=0$ まで再帰的に行う。

従来型動的計画法を適用する上で最も難しいことは、状態変数 $\mathbf{x}_k$ の次元数の増加に伴ってメモリーサイズ(データの格納領域)と計算数が飛躍的に増大することである。これは、状態ベクトルを量子化することに起因する。たとえば、問題の定義域の次元数が $n=5$ で、段数が $N=10$ の問題に対し、 $\mathbf{x}_k$ の各要素当り $l_x=20$ レベルに分割したとき、(9)式の右辺を評価するために計算機メモリーに格納する必要がある $J_{k+1}(\mathbf{x}_{k+1})$ の格子点の数は $l_x^n=20^5=3,200,000$ 個である。また、 $J_k(\mathbf{x}_k)$ と $\mathbf{u}_k^*$ の計算を必要とする格子点の総数(計算領域)は $l_x^n \cdot N=20^5 \times 10=32,000,000$ 個となる。量子化レベルをさらに増やしたとき、従来型動的計画法によるアプローチでは事実上、計算が実行不可能に陥る。

3. 基本形複合アルゴリズム

従来型動的計画法は、状態の遷移方向(通常は、時間の増加方向)に関し、後向きに定式化している。これは、「後向き」最適性の原理¹⁾を用いたことによる。一方、基本形複合アルゴリズムで用いる前向き動的計画法は、この原理を、時間に関して前後を逆にした「前向き」最適性の原理に基づき、初期点から前向きに定式化する。

いま、量子化した許容制御 $\mathbf{u}_k \in U(\mathbf{x}_k)$ を初期点 $\mathbf{x}_0$ から、各段 k で繰り返し適用してできる、初期点からの実行可能経路集合を考え、 $\{X_k^0\} = \bigcup_{i=0}^k X_i^0$ とおく。ただし、 X_k^0 は、再帰的に定義され、 $X_0^0 = \{\mathbf{x}_0\}$ 、 $X_{k+1}^0 = \{\mathbf{x}_{k+1} | \mathbf{x}_{k+1} = \mathbf{g}_k(\mathbf{x}_k, \mathbf{u}_k), \mathbf{x}_k \in X_k^0, \mathbf{u}_k \in U(\mathbf{x}_k)\} (k=0, \dots, N-1)$ とする。また、 $\mathbf{g}_k$ を状態遷移関数とする。この経路群は初期点からの枝構造をもつ。このような、状態点の決め方、すなわち、 k 段の状態点 $\mathbf{x}_k \in X_k^0$ を押し出すことによって、 $k+1$ 段の状態点 $\mathbf{x}_{k+1} \in X_{k+1}^0$ を確定させる計算を、「押し出し計算」と呼ぶことにする。ここで、計算上の工夫として、前向き動的計画法でのすべての計算点を、押し出し計算によって、この実行可能経路 $\{X_k^0\}$ 上にとり、かつ、コスト関数の計算と比較を、正確にこの経路上のコスト値を用いて行うことを考える。このような処理では、コスト関数の内挿計算を含まない。しかし、このままでは、段数 N が大きいとき、実行可能経路数の巨大化を招く。この巨大化を避けるため、次節では、量子化について考察する。

3.1 量子化手続きと前向き動的計画法

基本形複合アルゴリズムでは、前向き動的計画法を適用するのに、許容状態集合 X_k を、 $X_k = \bigcup_{i=1}^{m_k} X_{ki}$ 、 $X_{ki} \cap X_{kj} = \emptyset (i \neq j)$ となるように、適当な部分集合 $X_{k1}, X_{k2}, \dots, X_{km_k}$ に量子化する。以後、この部分集合 X_{ki} を、「ブロック」と呼ぶ。各ブロック X_{ki} に対し、初期点 $\mathbf{x}_0$ からの経路が存在する場合のみ、一個ずつ代表点 $\mathbf{x}_{ki} (\in X_{ki})$ とそのコスト関数の値を以下のように定義する。すなわち、

$$f_k(\mathbf{x}_{ki}) = \min_{\mathbf{x}_k} \{T(\mathbf{x}_k) | \mathbf{x}_k \in X_{ki}\} \quad (i=1, \dots, m_k, k=0, \dots, N) \quad (10)$$

とする。ここで、 $T(\mathbf{x}_k)$ は、前段の代表点 $\mathbf{x}_{k-1i} (\mathbf{x}_{k-1i} \in X_{k-1i})$ に、量子化した各 $\mathbf{u}_{k-1} \in U(\mathbf{x}_{k-1i})$ を適用し、式、

$$\begin{aligned} T(\mathbf{x}_0) &= 0 \\ T(\mathbf{x}_k) &= f_{k-1}(\mathbf{x}_{k-1i}) + L_{k-1}(\mathbf{x}_{k-1i}, \mathbf{u}_{k-1}) \\ \mathbf{x}_k &= \mathbf{g}_{k-1}(\mathbf{x}_{k-1i}, \mathbf{u}_{k-1}) \end{aligned} \quad (i=1, \dots, m_k, k=1, \dots, N) \quad (11)$$

を用い、押し出し計算によって計算する。ここで、 L_k を1段当りのコスト関数とする。ただし、便利さのため、最終段 N でのコスト $\Phi_N(\mathbf{x}_N)$ を、 $N-1$ 段でのコスト L_{N-1} に含めて考える。また、 $m_k (m_k \leq n_k)$ を、 k 段の代表点の数とする。この代表点 $\mathbf{x}_{ki}$ が、初期点 $\mathbf{x}_0 (\mathbf{x}_0 = \mathbf{x}_{01})$ から、 k 段の各ブロック上での到達点までの経路の中で、対応するコスト関数の値が(各ブロックの中で)最小となる点であり、初期点から再帰的に計算できる。このような代表点におけるコスト関数の最小値を最小コスト関数と呼び、代表点 $\mathbf{x}_k$ の関数として $f_k(\mathbf{x}_k)$ とおく。このとき、有限個の代表点 $X_k^0 = \{\mathbf{x}_{k1}, \mathbf{x}_{k2}, \dots, \mathbf{x}_{km_k}\} (k=0, \dots, N)$ 上で考慮した前向き動的計画法が成立する。すなわち、

$$\begin{aligned} f_0(\mathbf{x}_{01}) &= 0 \\ f_{k+1}(\mathbf{x}_{k+1i}) &= \min_{\mathbf{x}_{ki}, \mathbf{u}_k} \{f_k(\mathbf{x}_{ki}) + L_k(\mathbf{x}_{ki}, \mathbf{u}_k) | \\ &\quad \mathbf{x}_{ki} \in X_k^0, \mathbf{u}_k \in U(\mathbf{x}_{ki})\} \quad (i=1, \dots, m_k) \\ \mathbf{x}_{k+1i} &= \mathbf{g}_k(\mathbf{x}_{ki}, \mathbf{u}_k) \quad (k=0, \dots, N-1) \end{aligned} \quad (12)$$

とする。

一方、代表点以外の各ブロック内の状態点の最小コスト関数の値は、代表点の値で近似し、

$$\begin{aligned} \text{もし } \mathbf{x}_k \in X_{ki} \text{ ならば, } f_k(\mathbf{x}_k) &= f_k(\mathbf{x}_{ki}) \\ (i=1, 2, \dots, m_k, k=0, 1, \dots, N-1) \end{aligned}$$

とする。

3.2 優先順位計算

基本形複合アルゴリズムの計算は、初期点から前向きに進行する。初期点および各段の代表点では、量子化した制御を適用し、枝構造をもった最適経路候補群を繰り返し生成させる。この際、(11)式によって計算されるコスト関数値 $T(\mathbf{x}_k)$ の小さい順に、各経路がブロックに到

着するように処理する。以後、この処理を優先順位計算と呼ぶことにする。このようにすると、代表点の定義により、ブロックに最初に到着した経路がそのブロックの代表点となる。それ以後に到着するいかなる経路も代表点となることはできないから、それらを削除できる。このアルゴリズムは、初期点からの経路のいずれかが、終端条件を満たす状態集合 Ω_F に最初に到着したとき終了する。このとき、この先着経路が最適経路となる。この先着経路に対応する最適コストを

$$f_{0,N}^* = \min_{x_N} \{f_N(x_N) | x_N \in \Omega_F\} \quad (13)$$

と定義することにする。

3.3 限定操作とクリアランス

基本形複合アルゴリズムでは、前向き動的計画法の計算数を削減するために、分枝限定法の限定操作を使う。

いま、(9)式の最小コスト関数 $J_k(x_k)$ の下界値を $M_k(x_k)$ とし、また、原問題(3)～(8)式下での最終最適解 $f_{0,N}^*$ (13) 式の上界値を I (暫定解) とする。これらの下界値と上界値が満たすべき条件は、それぞれ

$$\begin{aligned} M_k(x_k) &\leq J_k(x_k), x_k \in X_k \\ &= \min_{u_0, u_{k+1}, \dots, u_{N-1}} \left\{ \sum_{i=k}^{N-1} L_i(x_i, u_i) + \phi_N(x_N) \right\} \\ (k=0, \dots, N-1) \end{aligned} \quad (14)$$

$$I \geq f_{0,N}^* \quad (15)$$

で与えられる。よく知られているように、もし、

$$f_k(x_k) + M_k(x_k) > I \quad (k=0, \dots, N) \quad (16)$$

ならば、状態 x_k を通る経路について、

$$f_k(x_k) + J_k(x_k) \geq f_k(x_k) + M_k(x_k) > I \geq f_{0,N}^*$$

が成り立つ。よって、代表点 x_k は最適経路の一部にならないから削除してよい。ただし、 $f_0(x_0)$ は、(11)式より $f_0(x_0) = T(x_0) = 0$ である。また、ここでは、最終段 N のコスト $\phi_N(x_N)$ を、 $N-1$ 段でのコスト L_{N-1} に含めて考えたから、形式的に、 $J_N(x_N) = 0$ とする。したがって、 $M_N(x_N) = 0$ とおく。削除できる代表点 x_k の数を増やすには、(16)式から明かなように、上界値 I をより小さな値へ、一方、下界値 $M_k(x_k)$ をより大きな値へと強化すればよい。ここで、上界値と下界値の強さを表わす量として、それぞれ、 Δa および Δb_k を導入し、

$$\Delta a = I - f_{0,N}^*, \Delta b_k = J_k(x_k) - M_k(x_k) \quad (17)$$

とおく。(17)式の I と $M_k(x_k)$ を(16)式に代入すると

$$f_k(x_k) + J_k(x_k) > f_{0,N}^* + \Delta a + \Delta b_k \quad (18)$$

となる。(18)式より、基本形複合アルゴリズムの計算数は、 Δa や Δb_k の個別の値というよりも、それらの和 $\Delta \epsilon (\Delta \epsilon = \Delta a + \Delta b_k)$ に依存する。以後、この和 $\Delta \epsilon$ をクリアランスと呼ぶことにする。基本形複合アルゴリズムを用いて大域解を得るには、(14)、(15)式の制約を満たす

有効な下界値と上界値を計算できなくても、より緩和された、大域解を得るためのクリアランスの条件、すなわち、

$$\Delta \epsilon = \Delta a + \Delta b_k \geq 0 \quad (19)$$

を満足すればよい。ここで、この複合アルゴリズムの計算数が最小となるのは、 $\Delta \epsilon = \Delta a + \Delta b_k = 0$ のときである。

3.4 基本形複合アルゴリズム

基本形複合アルゴリズムは、つぎの10ステップに要約できる。

1. (境界条件設定): 初期条件 $x_0 = c_0$ と終端条件 $x_N \in \Omega_F$ を設定する。

2. (状態集合の量子化): 各段の状態集合 X_k を、 n_k 個のブロック $X_{k1}, X_{k2}, \dots, X_{kn_k}$ に分割する(ただし、 $X_k = \bigcup_{i=1}^{n_k} X_{ki}, X_{ki} \cap X_{kj} = \emptyset (i \neq j)$)。

3. (クリアランスの設定): 各 $k=0, \dots, N$ に対し、適当な $x_k \in X_k$ に対する下界値 $M_k(x_k)$ を計算する。また、一つの上界値 I を設定する。(ただし、有効な下界値を計算できないときは、 $M_k(x_k) = 0$ とし、 I を $\Delta \epsilon \geq 0$ とするように設定する)。

4. (初期化): $\Omega \leftarrow \{x_0\}$, $T(x_0) = 0$, $\Re \leftarrow \emptyset$ を行う ($\Re$ は、そこまでの最小コスト経路が確定したブロックの集合を、また、 $\leftarrow$ は代入操作を表わす)。

5. (停止): もし、 $\Omega = \emptyset$ なら停止せよ。

6. (前向き動的計画法): Ω の中から $T(x^*)$ の値が最小である x^* を選び、 $\Omega \leftarrow \Omega \setminus \{x^*\}$ とせよ。そして、 x^* が属する段の値を k にセットし、 $x_k^* \leftarrow x^*$ とせよ ($\setminus$ は差集合の演算子)。

7. (終端テスト): もし x_N^* が終端条件 $x_N \in \Omega_F$ を満たすなら、停止せよ(この段階で最適解が得られる)。

8. (代表点テスト): もし $[x_k^*] \in \Re$ ならばステップ5へ行け。それ以外は $\Re \leftarrow \Re \cup \{[x_k^*]\}$ とし、ステップ9へ行け ($[y]$ は状態 y を含むブロックを示す。この段階で、 x_k^* に到達させる最適制御 u_{k-1}^* が確定する。また、最小コスト関数の候補値 $T(x_k^*)$ は、真の最小コスト関数値 $f_k(x_k^*)$ となる。)

9. (分枝限定操作): 量子化した各 $u_k \in U(x_k^*)$ に対して、次段の状態 x_{k+1} および最小コスト関数の候補値 $T(x_{k+1})$ を、 $x_{k+1} = g_k(x_k^*, u_k)$, $T(x_{k+1}) = T(x_k^*) + L_k(x_k^*, u_k)$ とする。そして、もし各 x_{k+1} に対し、 $T(x_{k+1}) + M_{k+1}(x_{k+1}) \leq I$ ならば、 $\Omega \leftarrow \Omega \cup \{x_{k+1}\}$ とせよ。

10. ステップ5へ行け。

ステップ6で x^* を選択するとき、 x をそれらに対応する最小コスト関数の候補値のサイズの順序に整理しておく、探索の手間を削減できる。

4. 経路群の繰返し生成による、コスト上下界値の推定精度の向上

4.1節と4.2節で述べる複合アルゴリズムは、3章で述べた基本形複合アルゴリズムを改良したものである。4章以降での複合アルゴリズムは、この改良型の複合アルゴリズムである。

4.1 繰返し複合アルゴリズム

(16)式による限定操作を強化し、より多くの計算数を削減するため、下界値を精度良く推定する繰返し論理を考える。この目的のため、複合アルゴリズムの一連の計算を、より細かく量子化した状態集合と許容制御の下で繰返す。すなわち、 $i(i=2, 3, \dots)$ 回目の逐次計算では、状態集合 X_k を、前回 ($i-1$) のブロックよりも、より小さなブロックに分割する。たとえば、各状態変数に対し、量子化幅を前回の半分とし、したがって、量子化レベルを2倍に増やす。一方、許容制御 $u_k^{l,i}$ の範囲を

$$u_k^{l,i} \min \leq u_k^{l,i} \leq u_k^{l,i} \max \quad (20)$$

ここで、

$$u_k^{l,i} \min = u_k^{*,l,i-1} - \Delta u_k^l, u_k^{l,i} \max = u_k^{*,l,i-1} + \Delta u_k^l \\ (k=0, 1, \dots, N-1, l=1, \dots, m, i=1, 2, \dots)$$

によって計算する。ここで、 $u_k^{*,l,i-1}$ は $i-1$ 回目の繰返し計算で得た最適制御であり、 l を制御変数のベクトル成分の添え字とする。また、 Δu_k^l は、通常、 $\Delta u_k^l \leq \Delta u_k^{l-1}$ となるように設定し、制御入力の変換数を変えずに量子化幅を前回よりも細かくする。たとえば、大域解を得る保証を放棄し、 $\Delta u_k^l = 0.5 \Delta u_k^{l-1}$ とする。また、 i 回目の許容制御 $u_k^{l,i}$ をつくる際、 $u_k^{*,l,i-1}$ を含めるように量子化し、かつ、前回の最適経路上の代表点を i 回目の代表点に加えると、最悪でも、前回の最適コスト値を保証できる。

しかし、状態変数や許容制御のこのような量子化レベルの増加の下では、複合アルゴリズムの計算数を指数関数的に増大させる可能性がある。この増大を抑制するため、上界値 I^i と下界値 $M_k^i(x_k) (x_k \in X_k)$ を、それぞれ、前回 ($i-1$) の繰返し計算で得た、最終最適コスト $f^*_{0,N}$ と最適経路上のコスト値を用い、それぞれ、

$$I^i = f^*_{0,N} \quad (i=2, 3, \dots) \quad (21)$$

$$M_k^i(x_k) = J_k^{i-1}(x_k^{*,i-1}) \\ = \sum_{l=1}^{N-1} L_l(x_k^{*,i-1}, u_k^{*,l,i-1}) + \Phi_N(x_k^{*,i-1}) \\ (k=0, 1, \dots, N-1, i=2, 3, \dots) \quad (22)$$

によって強化する。ここで、 $x_k^{*,i-1}$ と $u_k^{*,l,i-1}$ は、それぞれ、 $i-1$ 回目の繰返し計算で得た最適経路上の状態量と制御量である。(21)式による I^i は良好な上界値を与える。また、(22)式による M_k^i は、前回の最適経路の近傍で、ぴったりとした下界値を与える。しかしながら、最適経

路の近傍以外では、(22)式は大域解を得るための下界の必要条件、すなわち、(14)式の $M_k^i(x_k) \leq J_k(x_k) (k=0, \dots, N-1)$ を常に満たすとは限らない。また、この判定には $J_k(x_k)$ の評価を必要とするが、正確な $J_k(x_k)$ の値を知ることが難しい。そこで、次節では、(19)式の、大域解を得るためのクリアランスの条件 $\Delta \epsilon \geq 0$ を用いて、大域解を得ることを考える。

一方、この繰返し複合アルゴリズムの現実的な停止条件を

$$|f^*_{0,N} - f^*_{0,N}| \leq \epsilon_1 \quad (23)$$

で与える。ただし、 $\epsilon_1 \leq 1$ とする。

一般に、状態量や制御量の量子化単位が粗い時点では、経路群が終端条件を満たせなくなって、計算が停止してしまう可能性がある。この現象に対処するため、終端条件が満たされないほど終端段のコスト値が大きくなるようなペナルティ関数 $P(x_N)$ を導入する。そして、元の評価関数にこのペナルティ関数項を加えた新しい評価関数

$$J = \sum_{i=0}^{N-1} L_i(x_i, u_i) + \mu P(x_N) \quad (24)$$

ここで、

$$P(x_N) = \Psi(x_N)^T \Psi(x_N) \quad (25)$$

の下で、終端条件のない最小化問題を解く。ここで、 $\Psi(x_N) = \mathbf{o}$ は原問題の終端条件であり、 μ は十分大きな正定数とする。このとき、量子化単位を細かくするほど、したがって、イテレーション回数の増加とともに、最適経路は終端条件を満たすように改善される。

このように、繰返し複合アルゴリズムでは、上界値と下界値を逐次改良する過程で、同時に、初期点からの最小コストを、より精密な値に逐次近似している。

4.2 並列複合アルゴリズム

大域解を得るためには、状態変数の定義域全体に対して、 $\Delta \epsilon \geq 0$ (クリアランスの条件(19)式)となる必要がある。そのために、もし、 $\Delta \epsilon < 0$ の領域を含む場合は、(22)式の下界値を下方修正し、

$$M_k^i(x_k) = J_k^{i-1}(x_k^{*,i-1}) - |\Delta \bar{\epsilon}_{\min}| \\ (k=0, 1, \dots, N-1, i=2, 3, \dots) \quad (26)$$

と置く。ただし、 $\Delta \epsilon$ の値が明示的でないため、 $\Delta \bar{\epsilon}_{\min}$ を $\Delta \epsilon (< 0)$ の最小値の推定値とする。もし、 $\Delta \epsilon \geq 0$ ならば、 $\Delta \bar{\epsilon}_{\min} = 0$ と置く。しかし、コスト関数が多峰性である場合には、(26)式のように、一つのパラメータ $\Delta \bar{\epsilon}_{\min}$ によって、全定義域に渡る下界値を設定することは、クリアランスの増加、したがって、計算数とメモリーサイズの増加を招く。

並列複合アルゴリズムは、このような場合に、大域解を求める実際的な算法である。最初に、並列処理のため

に、粗い量子化の下で経路候補群を生成する。これらの経路での値を、並列計算の初期値とする。これは原問題のコスト構造を調べることに相当する。すなわち、並列複合アルゴリズムでは、繰り返し複合アルゴリズムの初回 ($i=1$) の計算で得た、1 番目から $j_{\max}$ 番目までのコスト値をもつ最適経路候補群に対し、次回以降 ($i \geq 2$)、それぞれ個別に、繰り返し複合アルゴリズムを適用し、各々の逐次近似解を求める。したがって、並列複合アルゴリズムの i 回目 ($i \geq 2$) の繰り返し計算が終了したとき、新たな最適経路候補群が生成される。並列複合アルゴリズムに対しても、 i 回目のイテレーションでの状態集合と許容制御の量子化のしかた、および、第 j 番目の経路に関する上下界値の更新のしかたは、繰り返し複合アルゴリズムと同様である。すなわち、

$$I^{j,i} = f * \tilde{u}_{N-1}^{j,i-1} \quad (j=1, 2, \dots, j_{\max}, i=2, 3, \dots) \quad (27)$$

$$\begin{aligned} M_k^{j,i}(x_k) &= f_k^{j,i-1}(x_k^{*,j,i-1}) - |\Delta \tilde{\epsilon}_{\min}^j| \\ &= \sum_{l=k}^{N-1} L(x_l^{*,j,i-1}, u_l^{*,j,i-1}) + \Phi_N(x_N^{*,j,i-1}) - |\Delta \tilde{\epsilon}_{\min}^j| \\ &\quad (k=0, 1, \dots, N-1, j=1, 2, \dots, j_{\max}, i=2, 3, \dots) \end{aligned} \quad (28)$$

によって強化する。ここで、 $\Delta \tilde{\epsilon}_{\min}^j$ は、クリアランスの条件 $\Delta \epsilon \geq 0$ (19) 式を満たす範囲を、全定義域から第 j 番目の最適経路周りの回廊と呼ばれる領域 (Fig. 1 参照) に緩和 (縮小) したときの推定値とする。

さらに、この上界値 $I^{j,i}$ は、並列処理によってすでに計算された、より小さな上界値、すなわち、

$$\begin{aligned} \bar{I}^{j,i} &= \min_{j_1 \in \{j, j+1, \dots, j_{\max}\}, j_2 \in \{1, 2, \dots, j-1\}} \{f * \tilde{u}_{N-1}^{j_1, i-1}, f * \tilde{u}_{N-1}^{j_2, i}\} \\ &\quad (j=1, \dots, j_{\max}, i=2, 3, \dots) \end{aligned} \quad (29)$$

によって強化することが可能である。このとき、第 j 経路の繰り返し計算で、全ての経路が限定操作により削除され、したがって、第 j 局所解が消去される可能性を期待できる。また、 i 回目の繰り返し計算では、より小さな上界値 $I^{j,i}$ をもつ経路を先に処理すれば、早い時点で、より強

力な上界値を得る可能性が高くなる。

前述の並列計算では、固定した各 i ($i=2, 3, \dots$) に対して、 j を変化させた。一方、固定した各 j ($j=1, \dots, j_{\max}$) に対して、 i を変化させれば、もう 1 つの並列計算を構成できる。

並列複合アルゴリズムの実行は、前回 ($i=1, 2, \dots$) の量子化単位下で計算した大域解と代表的な局所解の周りに、最適経路候補群からなる計算領域の回廊を逐次生成することに対応している。並列複合アルゴリズム (Parallel Hybrid Dynamic Programming Algorithm, Parallel HDP と略記) の計算領域の回廊を Fig. 1 に示す。この回廊幅のサイズは、クリアランスのサイズ $\Delta \epsilon$ によって制御され、イテレーション回数 i の増加とともに、上下界値の逐次改良によって、徐々に絞られる。

ここで、並列複合アルゴリズムの例題として、システム方程式 $x_{k+1} = x_k + u_k$ ($k=0, \dots, 4$) の下で、評価関数 $J = \sum_{k=0}^4 \{|x_k| + |u_k + 1|\}$ を最小にする最適制御列 $\{u_k^*, k=0, \dots, 4\}$ を求める問題を考える。初期値、終端値、制御変数を、それぞれ $x_0=3, x_5=4, u_k \in [-0.5, 1.5]$ としたときの計算手順を以下に示す。

1. 基本形複合アルゴリズムによる計算を第 1 回目のイテレーションとみる。状態量 x_k と制御量 u_k の量子化単位を、それぞれ 1 とし、 $u_k \in \{0, 1\}$ で代表させる。また、上下界値 $I^1=24, M_k^1=0$ ($k=0, 1, \dots, 4$) で出発する。この結果は Fig. 2 のとおりで、つぎの 2 つの経路候補が得られる。

経路候補-1: $0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 1$ (制御方策)

経路候補-2: $0 \rightarrow 0 \rightarrow 0 \rightarrow 1 \rightarrow 0$

2. 第 1 回目のイテレーションの結果、経路候補-1 より $I^{1,2} = f * \tilde{u}_{N-1}^1 = 21$ (最小コスト) で $M_0^{1,2}=21, M_1^{1,2}=$

Fig. 1 Corridor of computation regions in Parallel HDP

Fig. 2 Sketch of Parallel HDP at iteration $i=1$ for a simple problem

17, $M_2^{1,2}=13$, $M_3^{1,2}=9$, $M_4^{1,2}=5$ が, 経路候補-2 より $I^{2,2}=f * \frac{1}{2} \Delta u = 22$ (第2番目コスト) で $M_0^{2,2}=22$, $M_1^{2,2}=18$, $M_2^{2,2}=14$, $M_3^{2,2}=10$, $M_4^{2,2}=5$ が得られる. 第2回目のイテレーションでは, 状態量 x_k と制御量 u_k の量子化単位を, それぞれ 0.5 とする. すなわち, 制御操作 0 については, $u_k \in \{-0.5, +0.5\}$ で代表させ, 一方, 制御操作 1 については, $u_k \in \{+0.5, +1.5\}$ で代表させる. 経路候補-1 について, $I^{1,2}=21$ として並列複合アルゴリズムを適用した結果は Fig. 3 のとおりで, 経路: $-0.5 \rightarrow -0.5 \rightarrow -0.5 \rightarrow +0.5 \rightarrow +1.5$ が得られる. ただし, 終端条件のミスマッチによるペナルティー項のコスト $2.5 (\mu P = 10 \times 0.5^2)$ を加え, $I^{1,3}=f * \frac{1}{2} \Delta u = 19$ (最小コスト) となる. 一方, 経路候補-2 について, $\tilde{I}^{2,2}=19$ ((29)式を使って $I^{2,2}=22$ を強化) としたときの結果は Fig. 4 の通りで, ただちに限定操作が行われ, 新たな経路は生成されない.

3. 同様に, 第3回目のイテレーションを示すことは容易.

ただし, この例では $\Delta \epsilon = 0$, $\mu = 10$ として示した. 図中, 1 段当たりのコスト値 L_k を矢線中央に, 最小コスト関数の候補値 T_k を矢線頭部に, また, 最小コスト関数値 f_k を丸で囲った数字で示す. ただし, ペナルティーコスト

Fig. 3 Sketch of Parallel HDP at iteration $i=2$ for the candidate of pass-1

Fig. 4 Sketch of Parallel HDP at iteration $i=2$ for the candidate of pass-2

を加えた f_N を長方形で囲った数字で併記した. 一方, 優先順位計算 (Dominance Order Calculation, DOC と略記) によって確定した最適制御を太い黒矢印で, (分枝) 限定操作 (Branch and Bound, B & B と略記) によって削除された経路を X 印で示す.

5. 計算例

並列複合アルゴリズムによる解の精度とその収束性を Differential Dynamic Programming¹⁰⁾ (DDP と略記) と比較するため, 以下の, 状態変数制約最適制御問題

$$\text{Minimize } J = \int_0^1 ((x_1)^2 + (x_2)^2 + 0.005 u^2) dt \quad (30)$$

$$\text{Subject to } \dot{x}_1 = x_2 \quad x_1(0) = 0 \quad (31)$$

$$\dot{x}_2 = -x_2 + u \quad x_2(0) = -1 \quad (32)$$

$$x_2(t) \leq 8(t-0.5)^2 - 0.5 \quad (33)$$

を取り上げる. 全てのイテレーションを通じ, 独立変数 t の定義域を 20 分割し, 段 k と値 t_k を割り付ける. 初期点からの実行可能経路を正確に計算するため, 状態遷移の計算とコスト関数の計算には, 4 次ルンゲクッタ法を単精度で用い, ステップサイズを $\Delta t = 0.01$ とした. また, 各区間 $t_k \leq t \leq t_{k+1}$ での制御入力を $u_k(t) = u(u_k^*, u_{k+1}, t)$ ($u_{k+1} \in U$) とし, この区間を直線近似した. ただし, u_k^* を, 代表点での最適制御の値とする. 並列複合アルゴリズムでの初期値となる初回 ($i=1$) の計算では, 初期解が劣悪の場合 (最適解の近傍にない場合) を調べるために量子化を粗く採り, 状態変数の定義域を 8 分割とした. 一方, 制御変数の定義域を 17 分割した. 以後, イテレーションのたびに, 量子化レベルを増加させ, 最終のイテレーション ($i=12$) では, 状態変数について 320 分割した. また, $i=2$ 回目以降の制御変数のレンジを $\Delta u = 2.0$ ($i=2$) から 0.03125 ($i=12$) に徐々に狭めた. クリアランス $\Delta \epsilon$ は, 全イテレーションを通じ, 上界値の 1% とした. コストの収束の様子を Fig. 5 に示す. コスト関数値は, 数回のイテレーションで収束値付近に達し, 0.2127 ($i=1$) から 0.1707 ($i=12$) へ収束した. $i=12$ でのコス

Fig. 5 Cost versus iterations

Fig. 6 Convergence of x_2 trajectory

トの変化は 1×10^{-6} 以下となった。一方、経路 x_2 と制御量 u の収束の様子を、それぞれ、Fig. 6 と Fig. 7 に示す。計算時間は 168 秒 (15~20 MIPS の計算機) であった。

Ohno¹⁰⁾ は、この問題にニュートン法を使った differential dynamic programming (DDP と略記) を適用し、10 回のイテレーションで、コスト 0.1748 を得ている。

つぎに、並列複合アルゴリズムによる解の精度を推定する。制約条件付きのこの問題の解析解は得られないが、(33) 式の制約条件を取り除いた場合は、解析解が得られ、 $J=0.06936$ となる。一方、並列複合アルゴリズムでは、 $J=0.06945$ ($i=7$) の実行可能解を得た。この値は解析解からの誤差が僅か 0.13% である。制約条件付きの場合も、これと同程度の誤差を仮定できるとすると、並列複合アルゴリズムによる数値解 $J=0.1707$ は、精度が十分高いと考えてよく、また、DDP での解よりもコストが下がっている。

並列複合アルゴリズムによるこれらの結果は、(アルゴリズム自身が作り出す) 初期解が、たとえ劣悪であっても、最適解に収束することを示している。この主要因として、以下の 3 つが上げられる。第 1 に、本法の第 i 回目の繰り返し計算では、前回 ($i-1$) の最適経路の上下界値のみを参照し、経路そのものを直接参照していないこと。換言すれば、劣悪な初期解は参照経路としては使えないが、大域解を得るための計算回廊の大きさ (クリアランス) を決めることに利用できる。第 2 に、並列処理によって、局所解へ陥ることを防いでいること。第 3 に、本法でのコスト関数の計算と比較は、正確に、実行可能経路上に沿って行われるから、粗い量子化の下での解も、最適解の候補経路と成り得ることである。このような、本法の収束の特性は、本法をほかの 20 余例の最適制御問題に対して適用した結果からも確認されている。

他方、DDP 解は局所解であり、また、解軌道付近の初

Fig. 7 Convergence of control

期推定軌道を必要とすること、さらに、DDP の微係数計算における計算負荷を考慮すると、並列複合アルゴリズムは DDP との比較において、状態変数制約最適制御問題の解法に対し、かなり有力な算法だと考えられる。

6. 最適低推力軌道生成への応用

実際的な問題への複合アルゴリズムの有効性を調べるため、本法を最適な低推力軌道の生成問題に応用した。適用例として、太陽接近ミッションを考える。宇宙機は、地球の周回軌道を出発し、一定レベルの低推力によって加速を受けながら、舵角 $\theta(t)$ によって制御されている。ただし、途中、金星の重力場を利用した軌道変更 (スウィングバイ) を考慮に入れるものとし、このスウィングバイ過程での制御変数を、近金点距離 λ および、スウィングバイを行う際の惑星の側 s (太陽照射面/裏面) とする。また、金星とのスウィングバイ位置 p ($p=1, 2, \dots$) を最適化するものとする。ただし、 p は、宇宙機が金星の公転軌道上で p 回目の交差点をもった時点でスウィングバイを行うことを示す。このとき、問題は、あらかじめ指定した飛行時間 t_f 以内で、近日点距離 P_e を最小にする軌道を生成するように、最適制御列 $\{\theta(t), \lambda, s\}$ と最適スウィングバイ位置 p を決定することである。

この問題に対し、宇宙機の運動を平面に制約して考察する。質量 m の宇宙機の位置を極座標 (r, ϕ) で表わし、 r を引力中心からの距離、 ϕ を基準方向からの中心角、 u を宇宙機の半径方向速度、また、 v を円周方向速度とする。推力 T の方向を、この円周方向からの角度 θ (舵角) とする。これら状態量の定義図を Fig. 8 に示す。このとき、宇宙機の運動方程式を

$$\dot{r} = u \quad (34)$$

$$\dot{u} = \frac{v^2}{r} - \frac{\mu}{r^2} + \frac{T \sin \theta}{m_0 - |\dot{m}|(t - t_0)} \quad (35)$$

$$\dot{v} = -\frac{uv}{r} + \frac{T \cos \theta}{m_0 - |\dot{m}|(t - t_0)} \quad (36)$$

Fig. 8 Nomenclature used in the sun encounter mission problem

$$\dot{\phi} = \frac{v}{r} \quad (37)$$

と記述できる。ただし、 μ を重力定数、 m_0 と $-\dot{m}$ を、それぞれ、宇宙機の初期質量および推進剤の質量流量とする。また、 t_0 と t_f ($t_f \leq t_{\max}$) は所与であるとする。このとき、最小化すべき評価関数は

$$J = P_E(t_f) \quad (38)$$

となる。簡単化のため、スウィングバイ回数を1回以下とし、スウィングバイを行う際の近点距離 λ の範囲を、 $1.02 \leq \lambda \leq 1.22$ とする。ただし、 λ は金星半径で無次元化してある。ここでは、制約条件を、 $0 \leq r \leq 1.5$ AU (天文単位)とした場合を考察する。飛行時間を2年とする。低推力のレベルを 2×10^{-5} gm/sec² (ケース A) および 1×10^{-5} gm/sec² (ケース B) の2ケースとする。また、推力方向角 θ の範囲を、 $0 \leq \theta \leq 2\pi$ とし、初期速度増分と比推力を、それぞれ、5 km/sec と 5,000 sec とする。

6.1 計算結果

この数値例では、暦による宇宙機の具体的な出発日時を含めた計算を行っていない。また、コースティングがない場合を考察する。この問題に対し複合アルゴリズムを適用するため、許容状態集合をブロックに分割する。初回の計算 ($i=1$) では、状態変数 r, u, v の定義域を各16分割した。一方、制御変数 θ, λ, s の定義域を、それぞれ、8, 5 および 2 レベルに分割した。すべてのイテレーションを通じ、独立変数 t を40分割し、段 k と値 t_k を割り付ける。初回の計算での上界値 I^1 は、粗い量子化レベル下での数回の試行計算によって推定した。一方、下界値 M_k^1 を零と置いた。以後のイテレーションでは、状態変数の量子化レベルを各120分割 ($i=8$) まで増加させ、また、制御変数の可変域を徐々に減少させた。スウィングバイ過程以外の状態の遷移およびコスト関数の計算には、4次ルンゲクッタ法を単精度で使い、ステップサイ

Table 1 Minimum Perihelion distance by various methods

UNIT: (AU)					
CASE	SWINGBY POSITION	PARALLEL HDP	LATUS RECTUM	CIRCUMFERENTIAL	TANGENTIAL
A	1	0.0974	0.100	0.123	0.140
	2	0.106	0.107	0.125	0.143
	3	0.0988	0.113	0.119	0.137
	4	0.103	0.118	0.152	0.212
	NOTHING	0.139	0.148	0.197	0.219
B	1	0.203	0.211	0.207	0.219
	2	0.204	0.220	0.208	0.218
	3	0.204	0.226	0.206	0.215
	4	0.207	0.234	0.210	0.221
	NOTHING	0.307	0.331	0.332	0.347

THRUST LEVEL A: 2×10^{-5} gm/s², B: 1×10^{-5} gm/s²

Fig. 9 Optimal trajectory for $0 \leq r \leq 1.5$ (AU)

ズを $\pi/50$ とした。また、各区間 $t_k \leq t \leq t_{k+1}$ での推力方向角を、 $\theta_k(t) = \theta(\theta_k^*, \theta_{k+1}, t)$ ($\theta_{k+1} \in U, t_k \leq t \leq t_{k+1}$) とし、この区間を直線近似した。ただし、 θ_k^* を、代表点での最適制御とする。

並列複合アルゴリズム (Parallel HDP) で計算した最適解、すなわち、最小近日点距離 P_E を、Table 1 に示す。比較のため、latus lectum (楕円の通路方向、したがって、短径方向)、tangential (宇宙機の速度方向) および、circumferential (円周方向) を、それぞれ、推力方向とした固定推進戦略による数値解を併記した。この表より、繰り返し複合アルゴリズムでの解は、ほかのいずれの固定推

進戦略よりも、より小さな近日点距離を得ている。また、スウィングバイを行わない場合(NOTHING)と比べ、スウィングバイの効果がかなり大きいことがわかる。ケース A の最適解は、 $p=1, \lambda=1.02$ の位置で、スウィングバイを 1 回行った場合に得られ、最小近日点距離として、0.0974 AU を得た。この場合の最適軌道は、地球の公転軌道の内側に向かう。この様子を Fig. 9 に示す。計算時間は、約 300 秒(15~20 MIPS 計算機)であった。従来型動的計画法に対する計算数の割合は、約 $1/300$ ($i=1$) から $1/140,000$ ($i=8$) 程度であり、一方、計算領域は約 $1/100$ ($i=1$) から $1/40,000$ ($i=8$) に減少した。

これらの結果より、並列複合アルゴリズムは、低推力推進系と惑星スウィングバイを併用したような、実際的な複雑な軌道最適化に対して、十分、適用可能であると思われる。特に、低推力推進系を用いる場合には、低加速度ゆえに、解析にインパルス近似を用いることができないから、本法は、有効な手段を提供すると考える。

7. ま と め

今回の複合アルゴリズムでは、動的計画法と分枝限定法を併用した従来の複合アルゴリズムに対し、計算上の工夫として、コストの下界値を精度よく推定する繰り返し論理と、局所解を避ける並列処理を組み入れた算法を提案した。それらによって、数値計算上の負荷(計算数とサイズ)を大幅に削減し、局所解を回避できることが判明した。数値例による検討結果では、この拡張した並列複合アルゴリズムの解の精度は、0.13%程度であり、Differential Dynamic Programming のそれに匹敵する性能を示した。また、並列処理の組み合わせによって、全体のコスト構造を代表する、良好な近似解を得ることができた。これらの結果は、繰り返し論理を用いると、下界値を精度よく改良できることを示している。

また、金星とのスウィングバイを考慮した、低推力太陽探査軌道の最適化問題に本法を適用した結果、従来型動的計画法との比較で、計算数を大幅(約 $1/300$ ($i=1$) から $1/140,000$ ($i=8$))に低減させることができた。この結果は、低推力軌道最適化を含む、複雑な最適制御問題に対し、並列動的計画法複合アルゴリズムを適用できるこ

とを示している。

最後に、有益な助言を与えてくださった東大工学部航空宇宙工学科の田辺徹教授に謝意を表します。

参 考 文 献

- 1) R. E. Bellman: Dynamic Programming, Princeton University Press, Princeton, N. J. (1957)
- 2) S. Yakowitz: Dynamic Programming Applications in Water Resources, Water Resources Research, **18-4**, 673/696(1982)
- 3) O. G. Alekseev and I. F. Volodos: Combined Use of Dynamic Programming and Branch-and-Bound Methods in Discrete-Programming Problems Automation and Remote Control, **37**, 557/565(1976)
- 4) T. L. Morin and R. E. Marsten: Branch-and-Bound Strategies for Dynamic Programming. Opns. Res., **24**, 611/627(1976)
- 5) R. E. Larson and A. J. Korsak: A Dynamic Programming Successive Approximations Technique with Convergence Proofs, Automatica, **6**, 245/252(1970)
- 6) D. Jacobson and D. Mayne: Differential Dynamic Programming, American Elsevier, New York(1970)
- 7) H. Woźniakowski: Average Case Complexity of Multivariate Integration, Bulletin of the American Mathematical Society, **24-1**, 185/194(1991)
- 8) T. Hanaoka and T. Tanabe: A New Dynamic Programming Algorithm and its Application to Optimal Reentry Problems, Proc. of 13th Int'l Symp. Space Tech. Sci., 1031/1036(1982)
- 9) T. Hanaoka: A Unifying Approach by the Combined Algorithm for Optimal Control Problems of Flight Systems, Proc. of 4th RAMP Symposium, 67/82(1992)
- 10) K. Ohno: A New Approach to Differential Dynamic Programming for Discrete Time Systems, IEEE Trans., **AC-23**, 37/47(1978)

[著 者 紹 介]

花 岡 照 明 (正会員)

1970 年、日本大学理工学研究科修士課程修了。同年松下電器産業(株)勤務を経て、71 年東京大学工学部航空学科助手現在に至る。92 年、筑波大学経営政策科学研究科修士課程修了。動的計画法の応用の研究に従事。日本航空宇宙学会、日本 OR 学会会員。