効用の最大化と計算の複雑さ

論　文

効用の最大化と計算の複雑さ

谷　田　則　幸*

概　要

経済学の多くの場面で効用最大化問題などの最適化問題が表れる。最適化問題は、オペレーションズ・リサーチやゲーム理論にとどまらず、多くの科学分野で扱われている。

本稿では、最適化問題、とくに効用最大化問題について、計算量理論の側面から考察する。さらに、最適化問題を効率よく計算するアルゴリズムおよび方法について論じる。

キーワード：効用最大化、最適化問題、計算の複雑さ、数理計画問題、近似アルゴリズム、並列コンピューティング
経済学文献季報分類番号：02-21

1　はじめに

計算の複雑さの理論（Theory of Computational Complexity）は、理論計算機科学のもっともホットで、もっともアトラクティブな研究分野である。計算の複雑さの理論における研究の対象は、（問題を解くための）アルゴリズムの効率性である。一般に、ある現象を説明（解釈）するために、数学という道具を用いることが多い。それは、そのような現象の中に潜む性質や規則を発見するという意味での「真理の探究」と考えることができる。一方で、数学には問題を解くための方法（計算法）を発見するという「術の探求」という側面があることを忘れてはならない。例えば、二つの整数の最大公約数を求める、ユークリッドの互除法などがその例であり、古くはギリシャ時代にまでさかのぼることができる。この２つは、お互いに独立して存在するものではなく、互いに影響を与えながら成立し、発展するものである。このように、計算法、すなわちアルゴリズムを見つけることが数学の一分野として存在したわけであるが、その効率性についての研究が始まったのはそれほど古い話ではない。アルゴリズムの効率を考えるようになったきっかけはコンピュータの出現にある。それ以前の計算は、人間が行なっていたため、アルゴリズムの効率を計量するのが困難であった。しかし、コンピュータの場合は、アルゴリズムの効率は、実行時間という明確な量で捉えるこ

＊ taniad@kansai-u.ac.jp

とができるため、解こうとしている問題の複雑さをいやでも意識することになった。現在のコンピュータの起源は、ABCマシン（1942年）やENIAC（1946年）、あるいはプログラム内蔵方式を考案したvon NeumannによるEDVAC（1950年）といわれている。これ以降、人間は多くの場面で、コンピュータを用い、業務の効率化や利便性の向上、ひいては科学の発展に利用してきたといえる。人間が行なうには、複雑で難しい計算や簡単であっても退屈な繰り返し計算などは、コンピュータの得意とするところであり、計算に関してはコンピュータは万能であると考えられていた。しかしながら、この神話は、早くも1970年代には崩壊することとなった。すなわち、コンピュータにも解けない問題が存在すること、仮に解けたとしても非常に時間がかかる（例えば、数億年）ような問題が存在すること、が発見されたのである。言い換えれば、アルゴリズムを見つけることにより、その問題が解けることは示せても、その所要時間まで保証するものではないということである。

本稿では、コンピュータでさえも天文学的な時間をかけないと解けない問題について概観する。また、最適化問題としての効用最大化問題の計算の複雑さを評価し、それをとくためのいくつかの方法について考察する。

2　計算の複雑さ

ここでは、理論的（数学的）に解けることが分かっていたとしても、コンピュータが処理する（答えを出す）ために莫大な時間を必要とするような問題が存在することを明らかにするとともに、どのような例があるのかを見ることとする。

2.1　計算モデル

話を簡単にするために、計算モデル（理論上のコンピュータ）を決めておく。記憶領域（メモリ）が無限であることを除けば、現在目にするコンピュータと同じと考えても差し支えないであろう。もう少し、詳しく言えば、記憶域との固定長のやり取り（アクセス）が1ステップ、基本命令（四則演算、シフト、判断など）も1ステップで実現できるものとする。演算の対象が長い場合や浮動小数点を扱う場合には、そのような基本命令を何回か繰り返すものと考える。メモリを無限にしたのは、いかなる量のデータも扱えるようにするためである。このような計算モデルは、1936年にA. M. Turingにより考案されたTuring機械(Turing Machine, [6])と呼ばれる。また、この計算モデルで用いるプログラミング言語を表1のようにする。これは、FORTRANやCといった特定のプログラミング言語ではなく、アルゴリズムを分かりやすく記述するためのものである。

表1　記述できる実行文（[14] より引用加筆）

文の種類	形　　式	意　　味	ステップ数
代入文	$x \leftarrow a$	x の値を定数 a にする	$O(1)$ステップ
	$x \leftarrow y+z$, $x \leftarrow y-z$, $x \leftarrow y*z$, $x \leftarrow y$ **div** z, $x \leftarrow y$ **mod** z	y と x に四則演算を施し，その結果を x に格納する	$O(1)$ステップ
while文	**while** c **do** s **end**	条件式 c が真のあいだ s を繰り返し実行	$r\,(T(c)+T(s))$ ステップ，ただし繰り返しの回数を r とする
if文	**if** c **then** s_1 **else** s_2 **endif** **if** c **then** s_1 **endif**	条件式 c が真なら s_1 を，偽なら s_2 を実行する	c が真なら $T(c)+T(s_1)$ ステップ，偽なら $T(c)+T(s_2)$ ステップ
入力文	**input** x	コンマなど適当な区切り記号で区切られた1個のデータを読み，変数 x にその値を代入する	1ステップ
yes文	**yes**	入力に対し，yes と答えて停止する	1ステップ
no文	**no**	入力に対し，no と答えて停止する	1ステップ

2.2　計算可能性と決定可能性

ある問題に対し、その問題を解くアルゴリズムが存在しないとき、その問題を計算不能（uncomputable）という。例えば、プログラム停止問題と呼ばれる次のような問題は、計算不能であることが証明できる。プログラム停止問題は、yes か no かを問う判定問題であるが、計算不能の場合には、yes とも no とも決定できないので、決定不能（undecidable）であるともいう。

例1（プログラム停止問題）

入力：プログラム P と入力データ x

決定すること：入力 x のもとで P は停止するか？

また、問題を解くアルゴリズムが存在するとき、その問題を計算可能（computable）または決定可能（decidable）という。ここまでのところでは、計算可能かどうか、決定可能かどうかを議論しているだけであり、それに必要な時間（計算量）には言及していないことに注意されたい。

2.3　実際的計算可能性

まず、分割問題といわれる次のようにシンプルな問題を考えることにする。

例2（分割問題）

入力：自然数の列 $a_1, \ldots, a_n$

決定すること：$\{a_1, \ldots, a_n\}$ を2個の集合に分割する。このときそれぞれの集合に属する自然数の和が等しくなる、すなわち、$\exists A \subseteq \{a_1, \ldots, a_n\}$ s.t. $\Sigma_{i \in A} a_i = \Sigma_{i \notin A} a_i$、のようにできるか？

この問題を解くためには、入力として与えられた n 個の自然数を A とその補集合 A^c に分け、それぞれに含まれる要素を足し合わせ、得られた二つの数が等しいとき、答えはyesである、というアルゴリズムを考えればよい。集合の二分割は、2^n 通りあり、最悪の場合すべての分割を考えなければならない。したがって、答えを出すまでに、$O(c^n)$ $(c \geq 2)$ ステップを必要とする。ここで、O はオーダーを表し、カッコ内の整数倍のステップ数を意味する。また、n は入力のサイズである。この n が大きいときには、大変なステップ数（すなわち、時間）が必要であることがわかる。例えば、n ＝100の場合には、2^{100}＝1267650600228229401496703205376＞10^{30}であり、1ステップが1ナノ秒（$1ns = 10^{-9}sec$）とすると、10^{21}秒となり、30兆年以上の年月を要することがわかる。この計算は、生真面目にすべてのケースを考えるアルゴリズムを想定した場合であり、n が小さい場合やただ一つの解を求めればよい場合などはこれほどまでの時間は要しない。また、きわめて低い確率ではあるが1ステップで答えを見つけ出す場合もある。

このように、問題の入力サイズを n とするとき、ある定数 k が存在し、$O(2^{n^k})$ ステップで解くアルゴリズムが存在する問題のクラスを指数時間（exponential time）といい、$\mathcal{EXPTIME}$ であらわす。

2.3.1 効率の良いアルゴリズムが存在する問題のクラス—— $\mathcal{P}$

$O(n^k)$ ステップで解くアルゴリズムが存在する問題のクラスを決定性多項式時間（あるいは、単に多項式時間）（[deterministic] polynomial time）といい、$\mathcal{P}$ であらわす。たとえば、「n 個の自然数が与えられ、n 個の自然数の部分和が奇数になるものがあるか」は $\mathcal{P}$ に属する。このとき、あきらかに $\mathcal{P} \subseteq \mathcal{EXPTIME}$ であり、$\mathcal{P} \neq \mathcal{EXPTIME}$ であることも知られている。

一般に、アルゴリズムが効率がよいといわれるのは、多項式時間で解けるものをいう。問題の入力サイズ n は問題自身の複雑さをあらわすので、組み合わせを考えた n^k 程度の大きさは受け入れなければならないであろう。もちろん、k が100のような場合には非常に大きくなるので、実際的には役に立たない。まとめれば、次のようになる。

- 指数時間アルゴリズムは、効率が悪く、実質的に計算ができないほど時間がかかる

procedure Disservance,
$sum1 \leftarrow \sum_i a_i$
while 入力データがまだある **do**
  input a
  either $sum1 \leftarrow sum1 - a \wedge sum2 \leftarrow sum2 + a$ **or**
end
if $sum1 = sum2$ **then yes eles no**

図1　分割問題を解く非決定性アルゴリズム

- 多項式時間アルゴリズムは、効率がよく、問題の入力サイズに見合った時間で計算できる。とくに、kが比較的小さな場合には、きわめて短い時間で解くことができる

2.3.2　効率の良いアルゴリズムが存在しそうにない問題のクラス— $\mathcal{NP}$

先に見たように、分割問題を指数時間で解くアルゴリズムは存在する。しかし、分割問題を多項式時間で解くアルゴリズムが存在するかどうかはまだわかっていない。また、分割問題を多項式時間では解けないこともわかっていない。すなわち、分割問題が$\mathcal{P}$なのか$\mathcal{P}$でないのかが未解決な状況にある。プログラムの命令文として以下のものを用いた場合を考える。

either s_1 **or** s_2,

ここで、s_1は1個以上の文、s_2は0個以上の文とする。

この命令文は、実行時にs_1またはs_2のどちらかを選択し、最終的な結果としてyesが最も速く得られるように実行される。つまり、s_1またはs_2が非決定的な推測のもとで実行される。この命令文 **either** を用いて分割問題のアルゴリズムを図1に示す。

このアルゴリズムは、多項式時間（実際には、もっと効率が良いとされる、線形時間$O(n)$）で解けることが示せる。このように、非決定性計算モデル（アルゴリズム）により、多項式ステップで解くことの出来る問題のクラスを非決定性多項式時間といい、$\mathcal{NP}$であらわす。ここまでの議論で分かったことをまとめると次のようになる。

1．$\mathcal{P} \subseteq \mathcal{EXPTIME}$

2．$\mathcal{P} \neq \mathcal{EXPTIME}$

3．$\mathcal{P} \subseteq \mathcal{NP}$

4．$\mathcal{NP} \subseteq \mathcal{EXPTIME}$[1]

[1] 非決定性の計算モデルによりTステップで解ける問題は、決定性モデルでシミュレートすることにより、高々$O(2^T)$ステップで解ける。

5．$\mathcal{P}=\mathcal{NP}$か$\mathcal{P}\neq\mathcal{NP}$は未解決

6．$\mathcal{NP}=\mathcal{EXPTIME}$か$\mathcal{NP}\neq\mathcal{EXPTIME}$は未解決

2.3.3 $\mathcal{NP}$完全

まず、変換機（transeducer）を導入する。変換機とは、与えられた入力記号列を出力記号列に変換する 計算モデルを意味する。

この変換機が、関数fに対し、すべての入力語xに対し、多項式ステップで$f(x)$を出力する。このとき、fは多項式時間計算可能関数という。

A、Bをある問題とする。このとき、問題Aが問題Bに多項式時間還元可能（polynomial time reducible）であるとは、すべての語xに対し、Aへの入力xの解が存在するための必要十分条件が、Bへの入力$f(x)$の解が存在する場合にいう。

AがBに多項式時間還元可能で、Bが$\mathcal{P}$に属するならば、Aが$\mathcal{P}$に属する（多項式時間で解ける）ことは、トリビアルである。

さらに、問題Aが$\mathcal{NP}$に属し、$\mathcal{NP}$に属するすべての問題がAに多項式時間還元であるとき、Aは$\mathcal{NP}$完全（$\mathcal{NP}$ complete）という。これまでに例として用いてきた、分割問題は$\mathcal{NP}$完全である。$\mathcal{NP}$完全な問題は、（決定性モデルより強力な）非決定性計算モデルを用いても、多項式ステップを必要とし、現在われわれが使用するコンピュータでは（多項式ステップでは解けないという意味で）効率よく解くことは出来ないことを意味している。先に述べたように、$\mathcal{NP}$=$\mathcal{P}$？についての解答は見つかっていない。

2.3.4 $\mathcal{NP}$完全な問題

$\mathcal{NP}$完全な問題の数は、1000とも2000ともいわれている。ここでは、その中から代表的な問題を挙げておく。

例3（3彩色問題（3-colorability problem））

入力：無向グラフ$G=(V,E)$ ここで、Vは頂点（*vertex*）の集合、Eは辺（*edge*）の集合とする。

決定すること：Gは3彩色可能か（隣接する頂点が同じ色にならないように、3色で塗り分けられるか）、すなわち、$\forall(u,v)\in E$に対し、$f(u)\neq f(v)$を満たすような関数$f:\mathrm{V}\rightarrow\{1,2,3\}$が存在するか？

ハミルトン閉路とは、グラフのすべての頂点を一度ずつ通る閉路のことを言う。言い換えれば、グラフ上で、一筆書きが出来れば、その道筋がハミルトン閉路になっている。

例 4（ハミルトン閉路問題 (Hamiltonian cycle problem)）

入力：有向（無向）グラフ $G=(V,E)$ ここで、V は頂点の集合、E は辺の集合とする。

決定すること：G にハミルトン閉路が存在するか？

例 5（0-1 整数計画法問題 (0-1 integer programming problem)）

入力：整数行列 C と整数列ベクトル d

決定すること：$Cx=d$ を満たす0-1列ベクトル x が存在するか？

論理式の値が 1（すなわち、真）になるような真理値割り当てが存在するとき、その論理式は充足可能 (satisfiable) であるという。また、充足可能かどうかを判定するような問題を充足可能性問題という。リテラル（命題変数あるいは命題変数の否定）を和記号∨ (or) で結んだ式を節 (clause) といい、この節を積記号∧ (and) で結合した論理式を和積標準形 (conjunctive normal form: CNF) という。与えられた論理式が CNF で、それぞれの節が 3 個のリテラルからなるような論理式に対し、その充足可能性を判定する問題のことを3- 充足可能性問題といわれる。この問題が、最初に $\mathcal{NP}$ 完全であることを証明された問題である [3]。

例 6（3- 充足問題 (Satisfiability problem with 3 literals per clause: 3-SAT)）

入力：3 個のリテラルからなる、乗法標準形の論理式 P

決定すること：P は充足可能か？

例 7（ナップサック問題 (Knapsack problem)）

入力：自然数 C（ナップサックの容量）、$x_1,\ldots,x_n$（物の大きさ）

決定すること：大きさの和が C になるためには、どれを選べばよいか？

すなわち、$\sum x_{i\in A}x_i = C$ となるような集合 $A\subseteq\{x_1,\ldots,x_n\}$ が存在するか？

例 8（最適化ナップサック問題 (Knapsack problem with Optimization)）

入力：自然数 C（ナップサックの容量）、$x_1,\ldots,x_n$（物の大きさ）

決定すること：大きさの和が C 以下で、かつ最大となる値を求めよ。

すなわち、$\max\{\Sigma x_{i\in A}\, x_i \leq C \mid A\subseteq\{x_1,\ldots,x_n\}\}$

例9（拡張された最適化ナップサック問題（Extended Knapsack problem with Optimization））

入力：自然数C（ナップサックの容量）、$x_1, \ldots, x_n$（物の大きさ）と$p_1, \ldots, p_n$（価値）

決定すること：大きさの和がC以下で、価値の和を最大にするには、どれを選べばよいか？すなわち、$\Sigma_{i=1}^{n} x_i s_i \leq C$の制限のもと、$\Sigma_{i=1}^{n} x_i p_i$を最大にするような0-1ベクトル$(x_1, \ldots, x_n)$が存在するか？

例7、例8、例9が、この順で難しい問題であることは明らかであろう。

これらの問題は、効率を度外視にすれば、可能な組合せをすべて枚挙し、その中から条件に見合ったものを選ぶというシンプルな方法で解くことが出来る。いわゆる、「しらみつぶし法」により解けるわけだが、逆にそれ以外の方法で解けるとは思えない。ナップサック問題の考えうる組合せの数は、2^nであるし、ハミルトン経路の数は$\frac{(n-1)!}{2}$となる。nが小さな場合は、大した数にはならないが、nが大きくなれば、とんでもない数字になることがわかる。これらは、日常生活で、よく直面するような問題であり、多くの場合われわれは難なく解決している場合が多い。しかしながら、対象となるものが大きくなると、とたんに破綻してしまうことには気づいていないかもしれない。

分枝限定法（バックトラック法）やダイナミックプログラミング（DP）法などを用いて、組合せの数を極力少なくする（枝がりをする）ことにより、無駄を省くことは出来るが、これらの方法を用いたとしても、最悪の場合（あるいは、平均的には）指数オーダーになることがわかっている。

3　経済学における計算の複雑さに関する認識

3.1　経済学における最適化問題

最適化問題とは、目標となる事柄が最も良く達成できるようにするための解、すなわち最適解、を得ようとする問題である。最適化問題は、オペレーションズ・リサーチやゲーム理論にとどまらず、多くの科学分野で扱われている。最適化するための方法は、まず数学モデルを考え、数学的方法によって数学モデルで記述された問題に対する最適解を求めるというものである。たとえば、回帰分析における最小自乗法でのパラメータ推定も最適解を求めている、という意味で最適化手法である。

もう少し、一般的な最適化問題に数理計画問題がある。数理計画問題とは、ある制約条件のもとで、ある目的関数を最大化（あるいは最小化）するような最適化問題のことである。数理計画問題を解くための最適化手法は、数理計画法と呼ばれ、線形計画法、二次計画法や非線形計画法などが含まれる。数理計画法については、[17, 22, 23] などに詳しい記述が

ある。一般的な数理計画問題は、以下のようなものである。

定義1（一般的な数理計画問題）

m 個の不等式（制約）：

$$g_i(x_1, \ldots, x_n) \leq 0 (i = 1, \ldots, m)$$

および、l 個の等式（制約）：

$$h_k(x_1, \ldots, x_n) = 0 (k = 1, \ldots, l)$$

を満たす n 決定変数 $(x_1, \ldots, x_n)$ の中から、目的関数

$$f(x_1, \ldots, x_n)$$

を最小にするような変数の組が存在するとき、その値を求めよ。

このような数理計画問題において、f, g_i, h_k がすべて線形関数のとき線形計画問題、f が二次関数で g_i, h_k が線形関数のとき二次計画問題、それ以外は非線形計画法と呼ばれる。また、扱う変数の値が離散値（整数）に限られるとき、とくに、組合せ最適化問題といわれる。

最小自乗法は、無制約最適化問題の例であり、最適化したい目的関数を決定変数で偏微分した式を 0 とおいたときの解が最適解となる。

不等式制約を持たない最適化問題は、一般に Lagrange の未定乗数法により導出可能である。

一般的な数理計画問題は、Karush-Kuhn-Tucker 条件を満たすような決定変数の値が最適解となる。この問題を解析的に求めることは困難であるので、Lindo や NUOPT などの数理計画ソフトウェアを用いて、数値解法によりとくことが一般的である。

3.2 新古典派経済学—主流派経済学

現在の経済学における主流は、新古典派経済学だと思われる。イギリスの古典派の理論をマーシャルが発展させたという意味で、ケインズを除くケンブリッジ学派を新古典派と呼ぶ。広義な解釈では、1870年代の限界革命以降の、均衡分析を主流とした近代経済学をさしている [19]。新古典派経済学の起源は、限界効用論にあり、それ以前の古典派政治経済学と比べると、数学というツールが縦横無尽に使われていることがわかる。

新古典派経済学の核となるアイデアは、最適化原理（最大化原理、最小化原理）と均衡理論であろう。消費者は効用を最大化する行動をとり、生産者は利潤を最大化するような行動をとる。ある意味で、このように合理的な行動のみをとる主体の登場は、数学の弊害によるものと考えられなくもないが、ある状況下ではこのような主体の存在は十分考えうるものである。このように、定義された各主体が個別に行動をとり、その結果、経済全体としての状況変化がある。この状況を説明するために、均衡理論が必要となる。自然科学では、一般的

には平衡（equiliblrium）とよばれる均衡は、何らかの意味で釣り合いが生じ、変化しない状態を表している。新古典派経済学のいう均衡も、上記のような個別の行動が市場レベルで集められたとき、それぞれの行動を変える必要がないような状況を表している。

均衡理論において、経済全体について考えているのが、一般均衡理論である。経済には多くの要素が存在し、それらが互いに関連しあって、ある状態を形成しているものと考えられる。ただ、存在しうるすべての要素を考慮に入れることは難しいので、人工や社会構造などが変化しないということを仮定する。その条件の下で、ある一時点を考えると、それぞれの経済主体は自身の最適化原理に基づいた行動による競争により、相互に影響を及ぼしあうすべての変数の値が変化しなくなり、均衡状態に入る。すなわち、市場の調整機構により、一瞬にしてすべての需給が一致する、ということである。このようなメカニズムにより均衡が到達できるためには、各主体は一様に合理的かつ迅速な判断（最適行動をとること）をしなければならないし、さらにはその集積として得られた状態が均衡状態であることを、きわめて速く、正確に、コストをかけずに判断できる存在が必要となる。

3.3 効用最大化問題の計算量

次のような最適化問題を考えてみよう。

「消費者の効用関数を $u(x_1, \ldots, x_n)$、予算を C とする。ここで、$x_1, \ldots, x_n$ は購入する財・サービスの量とする。いま、消費者が任意の価格体系 $p_1, \ldots, p_n$ とするとき、予算の範囲内、すなわち、

$$x_1 \cdot p_1 + \ldots + x_n \cdot p_n \leq C$$

なる制約条件のもとで効用が最大となる値を求めよ。」

これは、効用最大化問題といわれるものである。これを別の表現で、次のように書き改めることが出来る。

入力：自然数 C（予算）、$x_1, \ldots, x_n$（購入する財・サービスの量）と $p_1, \ldots, p_n$（価格体系）

決定すること：大きさの和が C 以下で、価値の和を最大にするには、どれを選べばよいか？すなわち、$\Sigma_{i=1}^{n} x_i s_i \leq C$ の制限のもと、$\Sigma_{i=1}^{n} x_i p_i$ を最大にするような0-1ベクトル $(x_1, \ldots, x_n)$ が存在するか？

これは、組合せ最適化問題であり、拡張された最適化ナップサック問題に帰着されることがわかる［17, 18］。したがって、この効用最大化問題は $\mathcal{NP}$ 完全、すなわち、この問題を多項式時間で解くような効率の良いアルゴリズムはありそうにないことがわかる。

4 少しでも効率よく解くために

4.1 近似アルゴリズム

効用最大化問題は、$\mathcal{NP}$完全であることがわかったので、正攻法ではまともな時間では解けそうにない。そこで、次のような方針で、解を求めるようなアルゴリズムを考える。

1．求めた解は、最適なものでなくても良い。

2．ただし、出来るだけ最適解には近いものであってほしい。

3．絶対に、多項式時間、出来れば線形時間で解を求めたい。

そもそも、経済主体たる人間が行うべき計算を、コンピュータ上でシミュレーションしているわけだから、この程度の解でも問題はないものと思われる。

いま、Qを最適化問題、Iをその個別問題、Iに対する最適解を$Opt(I)$とする。さらに、Qに対する近似アルゴリズムを$\mathcal{A}$とすると、$\mathcal{A}$はIに対する（おそらく、最適ではない）解$V_{\mathcal{A}}(I)$を出力する。

いま、最大化問題を考えると、次のような関係が成立する必要があることに注意されたい。

$$v_{\mathcal{A}}(I) \leq Opt(I)$$

このとき、次のような近似アルゴリズムを考えることが出来る。

定義2（絶対近似アルゴリズム） 問題Qに対する近似アルゴリズム$\mathcal{A}$が絶対近似アルゴリズムであるとは、次の条件を満たす場合であり、かつ、その場合に限る：

$$\forall I \in Q, \exists k \quad s.t. \quad |Opt(I) - v_{\mathcal{A}}(I)| \leq k.$$

定義3（$f(n)$-近似アルゴリズム） 問題Qに対する近似アルゴリズム$\mathcal{A}$が$f(n)$-近似アルゴリズムであるとは、次の条件を満たす場合であり、かつ、その場合に限る：

$$\forall I \in Q \quad s.t. \quad size(I) = n \wedge \frac{|\, Opt(I) - v_{\mathcal{A}}(I) \,|}{Opt(I)} \leq f(n),$$

ここで、$size(I)$はIのサイズをあらわす。また、$Opt(I) > 0$とする。

定義4（ε-近似アルゴリズム） 問題Qに対する近似アルゴリズム$\mathcal{A}$がε-近似アルゴリズムであるとは、次の条件を満たす場合であり、かつ、その場合に限る：

$$\forall I \in Q, \exists \varepsilon \ s.t. size(I) = n \wedge \frac{|\, Opt(I) - v_{\mathcal{A}}(I) \,|}{Opt(I)} \leq \varepsilon .$$

定義から、絶対近似アルゴリズムが、もっとも望ましいことは容易にわかる。しかしながら、$\mathcal{NP}$完全な最適化問題のほとんどに対して、多項式時間近似アルゴリズムを期待できそうにないことも次のように容易に証明できる。

（証明）仮にそのような近似アルゴリズムが存在したと仮定すると、最適解そのものも多項式時間で求められることになってしまう。したがって、$\mathcal{P}=\mathcal{NP}$となってしまうが、これは$\mathcal{P}\neq\mathcal{NP}$という、基本予想に矛盾する。（*Q. E. D.*）

$f(n)$-近似アルゴリズムやε-近似アルゴリズムも三角不等式の条件をはずした場合には、ハミルトン閉路問題、巡回セールスマン問題、グラフの彩色問題と同様の結果が導かれる。個々の問題に対して、近似アルゴリズムの存在と問題の複雑さ$\mathcal{NP}$完全であるかどうか）は必ずしも一致しない。たとえば、「最適化ナップサック問題」や「箱詰め問題」と呼ばれる問題は、$\mathcal{NP}$完全であることがわかっているが、ε-近似アルゴリズムが存在し、かつかなり効率よく問題が解けることがわかっている［13］。

4.2 並列コンピューティング

現存のコンピュータでは、$\mathcal{NP}$困難な問題を効率よく解くことは難しいことは、すでに述べた。ナップサック問題に限っても、最適解に近い解（許容解）を求めてくれるような、効率の良い近似アルゴリズムはなさそうである。

他の可能性として考えられるのは、並列コンピューティングである。

量子コンピューティング［11］、分子コンピューティング［20］、分散コンピューティングなどがあげられる。量子コンピューティングの歴史は古く、1959年にRichard Feynmanがカリフォルニア工科大（Caltech）で行った講演での、「量子力学的な振る舞い」を計算に利用する、という言葉を起源としている［4］。もう少し具体的には、Paul Beniofの1980年からの一連の論文で、エネルギーを消費せずに計算が行えることを示し、量子コンピューティングの概念が誕生した［2］。量子コンピューティングの能力は高く、Shorの因子分解法アルゴリズムは、暗号技術に関して現在主流である、R. Rivest、A. Shamir、L. AdlemanによるRSA公開鍵暗号系さえも、たちまちにして破ってしまう可能性を示した［8］。従来は、スーパーコンピュータでさえも何億年かけても解けないと思われていたので、いわゆる、キラーアプリケーションといえよう。ただ、この量子コンピュータを実際に構築することが非常に困難であるため、なかなか実験の域を出ないことも事実である。

分子コンピューティングとは、DNAなどの高分子の性質を利用して計算を実現させるための研究分野であり、それを実現する可能性のひとつがDNAコンピュータである。分子コンピューティングの名を轟かせることとなったのは、南カリフォルニア大学のLeonard Adlemanによるデモンストレーション実験である。Adlemanは、前述の公開鍵暗号系の提案者の一人であるが、$\mathcal{NP}$完全な問題として知られる、有向ハミルトン経路問題を取り上げ、DNA分子上に問題を符号化し、分子生物学での高度な実験手法を駆使して、この問題

を解いた。そのときの実験は、頂点数が7、辺数が14のものであり、所要時間も2週間を要した。しかしながら、DNAを使って問題を解くという、その解決手法のユニークさには、筆者も少なからず感動を覚えた［1］。1995年には、同様の手法が、Liptonにより3-SAT問題に適用された［7］。この3-SAT問題に対し、論理変数を100とすると、解の候補は$2^{100}=1.3\times10^{30}$となる。1変数を15塩基対のDNA分子で、その解を1分子であらわすことを考えると、少なくとも100万トンのDNA分子（60億人分の総染色体DNAの量よりもはるかに多い）が必要になる。すなわち、計算量が空間量に置き換えられたことにより、計算効率が向したことがわかる。しかし、萩谷ら［20］により、ダイナミックプログラミングを利用することで、ミリグラムの単位のDNA分子により3-SAT問題をとくアルゴリズムも考案され、大きな可能性を見せている。とはいえ、量子コンピュータにしてもDNAコンピュータにしても、それ自身を作成する（すなわち、実験環境を整える）だけでも困難で、多くの時間を要する。ましてや、主体たる人間が持ち歩けるような、たとえばPalmサイズの小さなコンピュータではまだまだ実現しそうにない。

もうひとつの分散コンピューティングは他の二つと比べると、環境を整えやすいという意味では、敷居が低いように思う。分散コンピューティングとは、物理的に異なる（離れた）複数のコンピュータを何らかの方法で結合し、あたかも1台のスーパーコンピュータであるかのようにあつかえる技術である。たとえば、このようなストーリーである。パーソナルコンピュータはほとんど一家に一台以上の普及を見せ、その処理能力も向上の一途である。また、ほとんどのコンピュータは遊休状態（CPUを使っていない）であることが多く、言い換えれば「眠ったCPUパワー」が相当量あると思われる。さらに、最近のインターネットの急速な普及、ブロードバンド化、低価格化により、そういった余裕のあるコンピュータが高速ネットワークでつながれている。よって、インターネットを介し、巨大な仮想スーパーコンピュータが実現でき、多くの計算を必要とする、エイズ新薬の開発やがん新薬の開発といった遺伝子解析、デリバティブなどの金融工学の研究、などに大きく貢献するというものであり、もうすでに実際に行われている。この例では、パーソナルコンピュータとしたが、クライアントはワークステーションでも、スーパーコンピュータでも、PCクラスタ（多くのパーソナルコンピュータをローカルなネットワークで結合したもの。これ自体もひとつの分散コンピューティングの実現と考えることも出来る。）でもかまわない。実際このような取り組みは、日本でも国レベルで実施されている。スーパーSINETは、2001年3月にＩＴ戦略本部が発表した「e-Japan重点計画」に取り上げられ、2001年8月には「スーパーSINET推進協議会」を発足させて準備を進めていた、NII（文部科学省国立情報学研究所）が2002年1月4日から運用を開始した、約750の研究機関を結ぶ「SINET」の中核部分に導入され、

大学等の研究拠点を10Gbpsの光波長多重通信と光クロスコネクトを用いて接続した、世界で最初に光クロスコネクトを使用した総通信容量約300Gbpsの研究用のインターネットのことをさす。スーパーSINETは、高エネルギー・核融合科学研究、遺伝子情報解析研究、宇宙・天文科学研究、ナノテクノロジー研究、スーパーコンピュータ間接続研究等のバックボーン・ネットワークとして用いられる。[15]などでも、並列分子限定法によるアルゴリズムをＰＣクラスタ上で実現し、巡回セールスマン問題にアプローチしている。しかし、これを人間が瞬時に最適化問題を解くためのハンディツールとしては不十分である。

5　まとめ

新古典派経済学で用いられる効用最大化は、コンピュータサイエンスの視点からは、とても難しい問題であることがわかる。もちろん、本稿での主張は最悪のケースを想定し、その状況ではいかに困難なことになるか、ということをいっているに過ぎない。したがって、このことだけを捉えて、効用最大化を理論的よりどころとすることが間違いであり、それに替わる理論が早急に望まれるなどとは思わない。

おそらく、人間は自分が持ちうる情報の範囲内で、出来るだけ得になるような行動を自分なりの能力に合わせてとっていると思う。その意味で、このアイデア自身に問題はないと思う。もし、問題があるとすれば、個人差、個体差が、考えられていないことではないだろうか。これは、数学自身の存在意義にかかわることであるし、数学によるモデル化の宿命である。多くの不要な（不要だと思う）ものを取り除き、出来るだけシンプルにし、かつ、もとの存在をある程度説明しているように、表現する。それが、諸科学が数学に求められているものである。多くの場合、そのようなモデル化の過程で、本来備っていたであろう個人差、個体差という情報を捨て去られてしまったのである。

本稿では、人間が何気なくやっている計算が意外と複雑で、計算はコンピュータに任そうという考えは、通用しない場合もあるということがわかった。このことは、決定的手続きにより動作するNeuman型のコンピュータ（理論的には、Turing MachineやRandom Access Machine）では無理そうだという意味である。ただ、現在活発な研究がなされている並列コンピューティングにより、いま困難だと思っている問題がいとも簡単に解けてしまう可能性は十分あることは言っておきたい。将来、DNAがぎっしり詰まったパームトップコンピュータが登場するかもしれないのだから。

本稿ではふれないが、もう一つのアプローチとしては、マルチエージェントシミュレーションを用いたボトムアップ的にモデルを考察するものがある。これについては、[5]、[10]などを参照されたい。

謝辞

本稿は、平成13年度関西大学在外研究および平成15年度関西大学学部共同研究員（財政・金融政策の有効性に関する理論的・実証的研究）の成果報告の一部である。

参考文献

[1] L. M. Adleman, Molecular Computation of Solutions to Combinatorial Problems, Science, Vol.266, No.11, pp.1021-1024, 1994.

[2] P. Benioff, Quantum Mechanical Models of Turing Machines That Dissipate No Energy, Phys. Rev. Lett. 48, pp.1581-1585, 1982.

[3] S. Cook, The Complexity of theorem proving procedures, Proceeding of Third ACM Symposium on Theory of Computing, pp.151-158, 1971.

[4] R. P. Feynman, There's Plenty of Room at the Bottom , 29th annual meeting of the American Physical Society at Caltech, 1959.

[5] N. Gilbert and K. G. Troitzsch, Simulation for Social Scientist, Open University Press, 1999.

[6] J. E. Hopcroft and J. D. Ullman, Introduction to Automata Theory, Language, and Computation, Addison-Wesley, 1979.

[7] R. Lipton, DNA solutions of hard computational problems, Science, Vol.268, pp.542-545, 1995.

[8] P. W. Shor, Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, Nov. pp.20-22, 1994.

[9] J. E. Stiglitz, ECONOMICS, W. W. Norton & Company Inc., 1997（邦訳：藪下史郎 他 訳, 入門経済学、ミクロ経済学, 東洋経済新報社, 1999).

[10] N. Tanida and M. Murakami, A Study of the Problem of Poverty and Inequality using the Artiticial Society Model Multi Agent Based Simulation, 2003(in submition).

[11] C. P. Williams and S.H. Clearwater, Explorations in Quantum Computing, Springer Verlag, 1997（邦訳：西野哲朗 他 訳, 量子コンピューティング—量子コンピュータの実現へ向けて, シュプリンガー・フェアラーク東京, 2001).

[12] 井庭崇・福原義久, 複雑系入門, NTT出版, 1998.

[13] 伊理正夫・野崎昭弘・野下浩平, 計算の効率化とその限界, 日本評論社, 1980.

[14] 岩田茂樹, NP完全問題入門, 共立出版, 1995.

[15] 大西克実・榎原博之・中野秀男, ＰＣクラスタ環境における並列分枝限定法, 関西大学工業技術研究所情報科学研究会講演会［平成14年度第１回（第52回）].

[16] 小島政和・水野真治 他, 内点法, 朝倉書店, 2001.

[17] 塩沢由典, 市場の秩序学, 筑摩書房, 1990.

[18] 塩沢由典, 複雑系経済学入門, 生産性出版, 1997.

[19] 西村和雄, ミクロ経済学, 岩波書店, 1996.

[20] 萩谷昌己・横森貴, DNAコンピュータ, 培風館, 2001.

[21] 枇々木規雄, 金融工学と最適化, 朝倉書店, 2001.

[22] 柳浦睦憲・茨木俊秀, 組合せ最適化, 朝倉書店, 2001.

[23] 山川栄樹・福島雅夫, 数理計画における並列計算, 朝倉書店, 2001.