組合せの工学問題解決のための NECO/NODE システムの開発 (第1報)*

[JSPE-58-07
'92-07-1232]

——対象問題の性質と NODE 言語処理系の開発——

伊藤 俊** 高田 昌之*** 川野 邊 衛† 曾布川 拓司††

Development of NECO/NODE System for Combinatorial Engineering Problem Solving
(1st Report)
Characteristics of Problems and NODE Language Development

Masatoshi ITO, Masayuki TAKATA, Mamoru KAWANOBE and Hiroshi SOBUKAWA

There are many design problems that require the search-based problem solving method. Especially in the case that the design problem is to determine the object's structure, the trial and correction method is valid because the theoretically best algorithm is not discovered. In this paper the authors point out several characteristics of the problem class and propose the Hierarchical Backtrack Algorithm for the problems. To aid building problem solvers using the Hierarchical Backtrack Algorithm, the authors extended the PASCAL language and have developed a language called NODE which enables the automatic backtracking even when the program control moves among several procedures with different abstractions. In the latter part of this paper the design and implementation of NODE is described, and some examples are considered.

Key words: computer-aided design, problem solving method, backtrack algorithm, hierarchical backtrack algorithm, combinatorial optimization problem

1. 結 論

近年の知識情報処理技術の進展に呼応して、CADシステムの開発においても知識処理の重要性が認識され、知的な設計支援環境の実現を知識情報処理の立場から行なおうとする研究が現れている¹⁾。設計過程に知識が深く関与することに疑いないが、一方で知識処理だけでは解決されずに試行錯誤的に解を求めざるを得ない設計問題も数多い。現在の知識処理技術の中核であるルールベースシステムでは、一般に探索に基づく試行錯誤的処理は極力排除されている。それは人工知能研究初期に提案された探索に基づく問題解決手法²⁾ (現在では弱解法と呼ばれることがある) の多くの欠

点に対するアンチテーゼとして提唱されたためであると考えられる。設計過程全体が試行錯誤の過程であるという指摘は以前から存在するが³⁾、むしろ設計過程全体からみれば下請的な部分過程の問題において試行錯誤的な探索の必要性が高いものと考えられる。このような部分問題を設計の下部問題と呼ぶことにすれば、それは必ずしも高度に知的とは限らないが設計の質を左右する重要な問題となる。設計の下部問題と考えられるものとして、トリムプロブレム⁴⁾あるいはフローショップスケジューリング⁵⁾などの研究が既に行なわれてきている。

これらのような資源割当て問題は、本質的には計算理論の研究分野でいう組合せ最適化問題⁶⁾に含まれる。しかし、組合せ最適化理論では理論展開のために問題設定に制限を加えることが多く、目的関数や制約条件などが陽的に定義できる一部の抽象的な問題クラスについて最適アルゴリズムの追求が行なわれているに過ぎない。個々の対象問題に固有な制約条件が捨象されてしまうため、理論的な成果を実際の問題に応用することが困難な場合が多い。例えば板取り問題を考えたとき、シェア切断を行なうとすれば単に部材を詰め込むだけでなく切断時に他の部材を切断しないように切断順序を決定することが要求される。この条件を組合

* 原稿受付 平成3年9月4日

** 正 会 員 東京工業大学 総合理工学研究科(横浜市緑区長津田町4259)

*** 正 会 員 日本アイ・ピー・エム(株) CIMPO技術センター(東京都中央区日本橋箱崎町19-21)

† 正 会 員 コスモ・エイティ(株) (東京都杉並区阿佐ヶ谷北2-13-2)

†† 正 会 員 荏原製作所(株) 藤沢工場(藤沢市藤沢4720)

せ最適化問題として陽的に定式化した例は存在しない。ガスやレーザなどの切断法を用いれば切断線の順序を考慮する必要がなくなり理論的な詰込み問題に近くなるが、熱の集中などの条件を考慮する必要があるれば純粋に理論的方法だけで解くことは不可能となる。さらに、使用設備などからより個別的な制約が付加されることがある。

実用を目的とするときは最適アルゴリズムの開発を待つよりも個々の対象問題に対する解そのものを得ることが先決である。数理的に最適なアルゴリズムの開発には、多分に偶然の要素があり、多種多様な個別問題に対応した最適アルゴリズムを開発してゆくことは容易でない。しかも、組合せ最適化理論という最適アルゴリズムの意味は、問題のサイズに対する計算量のオーダが最小ということであって、実際に必要となる範囲においても計算量（計算時間）が最小であるとは限らない。このことは工学的には重要な区別である。

従来の研究は、知識処理に重点を置くあまり探索能力が低いものか、アルゴリズム論的な最適性に重点を置き過ぎて工学問題を解くという観点を欠いたものかのどちらかであったといえる。両者の中間に実用上重要な問題が数多く存在し企業などでは様々なシステムが開発されてきている。しかし、トリムプロブレムなどを除くとほとんど研究されてきておらず、設計の下部問題を解くための工学的なアプローチが必要となる。

そのための方法の一つとしてヒューリスティック探索が考えられる。この方法の利点は、対象問題固有のヒューリスティクスを反映させることが可能となることに加えて、問題設定が自由であることである。ヒューリスティック探索は、以前から知られているが、それは主にヒューリスティック関数の導入による探索枝の枝刈りを意味していた。本研究ではヒューリスティクスを活用した探索⁷⁾というより広い意味で用いる。探索に基づく問題解決システムは、組合せ爆発を生ずる危険性を抱えており、その構築環境には少なくとも次の二つの条件を提供することが要求される。

- (1) ヒューリスティクスの効果的な組込みのための方法。
- (2) 探索可能上限の拡大のための高速な演算実行速度。

本研究では、これらに対応して、専用の計算機言語 NODE の設計とその処理系の開発、NODE によって記述された問題解決システムを高速に実行する専用ハードウェアシステム NECO の開発を目的としている。

本論文では、設計の下部問題の性質を明らかにし、この問題を解くにあたって階層バックトラックアルゴ

リズムとよぶ方法が有効であることを示す。次に階層バックトラックアルゴリズムを手続き的に記述するための言語 NODE の設計と処理系の開発について述べる。また、著者らが NODE 処理系を利用して別途開発している応用システムについても触れる。NECO システムの開発については次報に述べる。

2. 設計の下部問題

本研究では、一般的な組合せ問題のうち、設計過程において解を求めることが要求されている問題を設計の下部問題とよぶ。理論化やモデル化の都合などによって実用上必要な条件を捨象することをしない。したがって組合せ最適化理論研究が対象としているよりも広い範囲の問題を対象とすることになる。設計の下部問題は、以下の P1 から P4 の性質のうち幾つかを持つことによって特徴付けられる。

- (P1) 問題固有の性質から様々なヒューリスティクスが得られる。
- (P2) 問題解決器の操作対象は構造的にモデル化される必要がある。
- (P3) 目的関数は陽的に定義されない。
- (P4) 解は、それが厳密な最適解でなくてもよい。

ヒューリスティクスが全く存在しないような問題は、純粋の数値計画問題として解くか、機械的探索によって解かざるを得ず、最適アルゴリズムの発見か演算能力の強化を待つこと以外に工夫の余地はない。しかし、設計の下部問題では、経験的あるいは設計者の洞察にもとづくヒューリスティクスが存在するのが普通である。人間は、自分の利害に直接関係するような具体的問題については様々な方策を考え出すことが知られている⁸⁾。ただし、その有効性は千差万別である。

緒論に述べたように問題解決器の操作対象は設計対象モデルであることが多く、これらは一般に構造的にモデル化される。したがって、それらの「良さ」を単一の目的関数やヒューリスティック関数などによって適切に定式化することは困難である。解の「良さ」を評価することも一つの問題解決となることもあり、少なくとも対象モデルを手続き的に評価しながら探索を行なうことが必要となる。ただし、達成された状態が目標条件（必要条件）を満足するかどうかの判定は可能でなければならない。例えば、機械を分解することを目標としたとき、具体的にどのような状況を分解したとするかについては部品の位置に無限の自由度があるが、与えられた部品群がその状態で分解されているかどうかの判定と組立手段に対する難易のおおよその評

価は可能である。

また、ヒューリスティック関数を導入して目的関数の値を無理に定量化することが従来多かった。しかし、このような定量化は、手法を見掛け上整ってみせる効果はあっても問題解決能力の向上は望めない。ヒューリスティック関数の値が有効であるのは限られた局面に関してのみ有効であって、探索過程全体の制御に利用したり探索の終端条件に用いたりすることはできない。もし、幾つかの目的関数を区分的に結合することが可能であったとしても、局面に依存するという性質は不変であるから、いずれにせよ局面の評価が必要となる。

設計では、満足される解が許容される手間によって得られればよく (Satisficing⁹⁾)、数理計画的な意味での最適性が保証される必要はない。Satisficing のための決定版の方法は知られていないが、自己の挙動を監視する (例えば展開される節点数などを監視する) ような手続きによって疑似的に実現することができる。

以上の性質は本来設計の下部問題の定義となるべきものであるが、本研究では形式的な定義を行なうことに成功していない。それは、組合せ最適化理論研究も成功していないように、現在の技術では対象問題のクラス定義を実際条件の捨象を一切伴わずに行なうことが不可能であるからである。なお、しばしば問題解決システムの開発に利用されてきた探索空間における解達成についての単調性は仮定できず分枝限定法や動的計画法などは適用不能である。また、求解を目的とするのであるから現在状態と目標状態との差異を探索の制御に利用するという方策も適用できない。

3. 設計の下部問題の解法

3.1 従来の問題解決手法

状態空間の木探索には大きく縦型探索あるいはバックトラックアルゴリズムと呼ばれているものと横型探索の二つの方法が知られている。これらの原理的な方法だけでなく幾つかの改良されたものが提案されている。代表的なものは、横型探索の拡張としてコルーチン探索¹⁰⁾、縦型探索の拡張として知的あるいは依存性指向 (Dependency Directed) バックトラックアルゴリズム¹¹⁾ (以下、DDBAと記す) が知られている。

コルーチン探索は、オブジェクト指向プログラミングシステムにおいても類似の機能が実現されているが、人間と同様な振舞をするという意味での知的問題解決の重要な要素とされている。利点としては以下のものが挙げられている。

(1) 手続きの中断および再開を自由に行なうことがで

きる。そのため、最も確からしい探索枝のみについて重点的に探索を進めることができ、それが失敗に終わったときも中断していた別の探索枝を再開するというような知的な探索を行なうことができる。

(2) 手続き間の交信機能がある。これによって、ある探索節点での失敗の情報を他の手続きに伝達することなどが可能となる。

(3) 中断されている探索節点については、探索木の重複を実行中に検出することが可能である。

しかし、以上の利点は、あらかじめ静的に設定される複数のプロセスが本質的に並行動作する必要がある場合か、問題がかなり小型であるためにヒューリスティクスが相当に強力になる場合にのみ発揮される。設計の下部問題では、探索木が大きいく、探索木の形があらかじめ判明している訳ではないので探索節点对応したプロセスの生成は動的に行なわれることになることなどから、上記の利点は必ずしも有効とはならない。(3) の探索部分木の重複発見は有用な機能ではあるが、その効果は、同時に展開される探索節点が比較的少数に限定できるときに限られる。もし、展開される探索節点数が増大するか、あるいは失敗となった探索枝もすべて保存しておく場合には、横型探索と同様に所要記憶容量が増大するとともに重複検査の計算コストが増大する。(1)、(2) の手続きの中断・再開および並行する手続き間の交信機能についても、多数の探索節点が展開されてしまった状況において、多数の中断節点の中から適切な再開節点を選択することは容易でない。もし、それを可能とする知識が存在する場合には、最初から候補を絞り込んでおくことが可能であることがほとんどである。また、不特定 (多数) の探索節点の間で節点ごとの局所的情報を交信するだけでは、探索全体の効率を向上させることは不可能である。オブジェクト指向システムにおいても観察されるように現状ではこれらの機能を実現するための計算コストが大きく、複雑なデータ構造を操作してゆく場合には問題が多い。

それに対して、DDBAは、再帰的に記述すれば制御構造は簡単で見通しがよい、探索木が大きいくとも使用記憶領域はそれほど増大しないなどといった通常の縦型探索の利点を生かしつつ、逐次的に後戻りするのではなく失敗の原因となる探索節点に直接遡って探索を再開する方法である。失敗の因果関係を記述するために、Dependency Networkと呼ばれる対象問題固有の依存関係情報を探索の進行に伴って生成することが行なわれる。どのような依存関係に着目するかが重要な関

題となるが、ヒューリスティクスの組込みが容易で展開される節点数を抑制できることから設計の下部問題の解法に適していると考えられる。

3.2 対象問題の階層分割

複雑な問題を解決する場合一般に、対象問題をその性質に基づいて階層分割するという方針が採用される。例えば、著者らの機械の分解経路を問題解決として求める問題^{1,2)}では、機械の分解という問題を図1に示すような階層構造に分割している。通常の探索木は二次元的であるが、問題を階層分割すると、図2に示すように階層に基づく立体的構造となる。このような探索木をここでは階層探索木と呼ぶ。

問題の階層分割は、それ自身が問題解決を目的とする一種のヒューリスティクスと考えることができるが、あらかじめ定められた静的な対象問題の整理である。人間は限定された局面においては様々な方策を考え出すことが可能であって、対象問題の階層ごとにそれらを組み込むことができ、同時に経験的知識の有効性が局面の限定によって保証されることになる。

問題の階層構造を利用した問題解決の研究としては、ロボットの行動計画を立案する階層計画システム^{1,3)}が有名である。このシステムでは、階層の上位からそれぞれ探索を完了させ、上位の探索結果を下位の探索のための計画とするようになっている。下位の探索は上位の探索節点を詳細化することに対応し、そのために計画と呼ばれる。上位の計画の変更を必要とせずに詳細化が可能な限りは、無用な上位の探索節点に対応した詳細レベルの探索を回避することができ探索効率を非常に向上させることができる。本論文では、このように階層の上位から階層ごとに探索を進めてゆく方法を同一階層優先探索と呼ぶことにする。

しかし、設計の下部問題を含む一般の問題では、階層計画システムが対象とした問題と異なり上位の計画

の実行が常に可能とは限らないことから、同一階層優先探索を行なうことは困難である。すなわち、ある階層において節点の選択を確定するためには、その節点の実行可能性を下位の階層すべてにわたって調べるとともに、場合によっては副問題の解決がさらに必要となる。例えば図1の問題では、移動部品の選択のみについてその順序の計画を立案しても、干渉部品の存在などによってそれらの計画は実行可能とはならず、干渉する障害部品を除去するという副作用を伴う副問題の解決が必要になる。

3.3 階層バックトラックアルゴリズム

階層計画システムのような同一階層優先探索が不可能である場合には、図2の点線で示すような探索を行ない、すべての階層にわたって完全な詳細化を逐一行なってゆく必要がある。本研究ではこのような探索法を、階層探索木を階層にまたがって縦型探索するという意味で階層バックトラックアルゴリズムと呼ぶ。

階層バックトラックアルゴリズムは原理的には同一階層優先探索と双対の関係にあるにもかかわらず、それを明示的に利用した例が存在しない。それは主に、同一階層優先探索が作用復元に関しては同一階層の直前の作用を逐次的に復元しながら後戻りすればよいのに対して、階層にまたがった複数の作用復元に大きな困難があるためと考えられる。DDBAの場合も、プログラムのどの箇所まで現在の状態を待避するか、失敗時にはどの箇所まで戻りどのように状態を復元するかなどといったことを逐一記述しておく必要がある。そのため、操作対象が比較的小型の問題に適用されてきたようである。

3.4 自動作用復元の方法

作用復元時のプログラミングの負担が問題解決システム構築の障害になっているような問題の場合、それをなんらかの方法によって解消する必要がある。とこ

Fig.1 Example of Hierarchical Problem Reduction^{1,2)}

Fig.2 Three-dimensional structure of Hierarchical Search Tree

ろで、図1の場合には移動方向を決めてから移動距離を決めるというように、上位の決定に基づいて下位の決定が行なわれる。すなわち、問題の階層構造は決定の依存関係を規定していると考えることができる。したがって、階層バックトラックアルゴリズムはDDBAの一種である。ただし、階層バックトラックアルゴリズムの場合は、一般的なDDBAと異なり階層分割が静的である。対象問題はあらかじめ階層的に整理され、その静的な関係に基づいて作用復元を行なえばよいことになる。したがって、そのような階層構造を認識することによって作用復元を自動的に行なうことが可能となるから、専用の言語処理系を設計すれば、作用復元は言語処理系の側で自動的に行なうことが可能となる。階層バックトラックアルゴリズムは、DDBAの利点を継承しつつ、階層分割によってヒューリスティクスの組み込み容易性と自動的な作用復元の可能性を獲得しているということになる。

4. 階層バックトラックアルゴリズム記述用言語NODE

4.1 NODEの開発方針

スカラー属性の探索について新しい言語などが開発されつつあるが、パターン情報などを記述する構造的データをも対象とした専用探索言語は開発されていない。階層バックトラックアルゴリズムによって設計の下部問題を解くシステムを記述するための専用言語は、パターン情報などを記述するために多様なデータ構造が取り扱えること、それを操作しかつ様々な形のヒューリスティクスを活用するために自由な制御構造を記述可能であることが必要である。従って、手続き的汎用計算機言語レベルの専用言語を用意する方が適していると考えられ、データ構造化に優れた手続き型言語PASCALを拡張するという方針を採用した。

バックトラック時の作用復元は構造化されたデータについて一括して行なわれる必要があるが、拡張はそのための機能とプログラムの制御構造を中心とし、データ構造化の方法すなわち型宣言などは大きく変更していない。著者らはこの言語を探索節にちなんでNODEと名付けている。

PASCALなどのALGOL系の言語は、実行の制御や変数のスコープを規定するために再帰的なブロック構造を採用している。NODEでは、これらに加えて自動的な作用復元をベース言語PASCALのブロック構造に基づいて行なうようにしている。一つの探索節点に対応するプログラムコードすなわち選択枝の数上げや最尤枝の

選択と試行などを一つのブロックとして記述すれば、バックトラックはブロックごとに生じる。作用の管理をプログラムのブロック構造に基づいて行なうようにすれば、階層バックトラックアルゴリズムのような階層にまたがった複雑なバックトラックにおいても作用復元を容易に自動化できることになる。

4.2 NODEのセマンティクス

Floyd^{1,4)}は、通常の手続き型言語でのバックトラックアルゴリズムの記述を容易にするために、DoステートとUndoステートと呼ぶ二つの実行モードの原理を考えた。Undoステートは、バックトラック時に対応するDoステートの(通常の)プログラムステートメントをそれぞれの作用を復元しながら逆行するものである。Undoステートにおける実行は、(人間が記述する)Doステートのプログラムステートメントの逆操作であるから機械的に自動生成することが可能である。そのため複雑なデータ操作の復元も確実に行なわれるようになる。Floydの研究ではこれをマクロ生成によって行なうようになっていた。

NODEプログラムの実行原理も、Floydの研究に従う。しかし、ステートメントを完全に逐次逆行する必要はなく、対象モデルや世界モデルを操作する単位ごとに復元を行なえばよい。その操作とプログラムブロックとが対応するように問題解決システムを記述すれば、Undoステートの逆行はブロック単位に行なえばよいことになる。

NODEでは、予約語BEGINとENDに囲まれた標準PASCALのブロックに加えて、予約語CHOICEとCHOICE_ENDに囲まれた非決定ブロックと呼ぶブロック構造を導入した。これに伴って標準PASCALのBEGIN/ENDブロックを決定ブロックと呼び、明示化するために予約語BLOCKとBLOCK_ENDによって囲むように変更した(予約語BEGINとENDは複合文を構成するためだけに用いる)。従って、NODEのPROCEDUREは局所変数宣言と決定ブロックおよび非決定ブロックによって構成され、ブロックは互いに入れ子をなすことができる。ここで、非決定(Non-deterministic)という用語は、Floydに依っており、CHOICEブロックはFloydのChoice関数の手続き的な拡張とみることができる。なお、バックトラック時に復元されるのは、大域変数のうち予約語MODELによって静的に宣言されたものだけである。設計対象モデルあるいは世界モデルは、MODEL宣言された大域変数に記述されるのが典型的である。

CHOICEブロックにおいて、実行ステートを切り替える実行文としては、SUCCESS(Do)、FAIL(Undo)およびCHOICE_FAIL(Undo)が用意されている。SUCCESSの実行

```

PROCEDURE try;
BEGIN initialize selection of candidates;
  REPEAT select next;
    IF acceptable THEN
      BEGIN record it;
        IF solution incomplete THEN
          BEGIN try next step;
            IF successful THEN cancel recording
          END
        END
      UNTIL successful ∪ no more candidates
    END
  END

```

Fig.3(a) Procedure skeleton of Backtrack Algorithm in PASCAL¹⁵⁾

は CHOICE ブロックの成功を表わし、Do ステートで当該ブロックの外部へ抜け出すことを指示する。FAIL の実行は何らかの失敗を表わし、最も内側の CHOICE ブロックの入口までバックトラックすることを指示する。CHOICE では、Do ステートに切り換わり実行が再開される。CHOICE_FAIL は通常、候補となる探索枝がすべて尽きてしまったとき実行され、対応する CHOICE ブロックから Undo ステートで抜け出ることを指示する。実行は最も内側のブロックの入口までバックトラックすることになる。

図3に N. Wirth の教科書¹⁵⁾にある PASCAL によるバックトラックアルゴリズムの記述例と、それに対応させた NODE の記述例を示す。

4.3 NODE による階層バックトラックアルゴリズムの記述

NODE のプログラムテキスト上で、通常のバックトラックアルゴリズムを階層バックトラックアルゴリズムに拡張することは非常に簡単である。図3(b)の例では、“acceptable”であるかどうかを判定するだけであるが、もし候補が acceptable となるようにすることを副問題とし、それに成功したとき探索を進めるようにすることが考えられる。副問題は、主問題に依存し下位階層に属するから、このような拡張は階層バックトラックアルゴリズムの最も簡単な例となっている。下位階層の問題を解くあるいは副問題を解くことは、別の手続きに制御を渡すことであり、NODE プログラムでは、PROCEDURE 呼出ししか別のブロックへの移動に対応する。NODE では、ブロックの任意の部分に他のブロックや他の PROCEDURE 呼出しを記述することが可能となっており、図4のようにより複雑な階層バックトラックアルゴリズムを記述することができる。

4.4 処理系の開発

本研究では NODE で記述した問題解決システムを専用ハードウェアによって高速に実行することを目的としており、そのコンパイラが必要となる。PASCAL には、

```

PROCEDURE try;
BLOCK initialize selection of candidates;
  CHOICE
    IF no more candidate THEN CHOICE_FAIL;
    select next;
    IF acceptable THEN FAIL;
    IF solution incomplete
      THEN try next step
    ELSE SUCCESS;
  CHOICE_END;
BLOCK_END

```

Fig.3(b) Procedure skeleton of Backtrack Algorithm in NODE

Fig.4 Typical NODE program

P-codeと呼ばれる仮想的な機械語を出力するコンパイラ(P-Compiler)が公にされており、それ自身PASCALで記述されている¹⁶⁾。P-code はスタックマシンを仮定しているが、専用ハードウェアもスタックアーキテクチャを持つことが望ましいため、専用ハードウェアの機械語はP-codeに準拠させることとした。これに自動的な作用復元のための拡張を行なってNECODEと名付けた。従ってNODEコンパイラの開発は、P-Compilerを改造して拡張を行ない、NECODEを出力するものとなる。NECODEの詳細と作用復元の方法については次報において述べる予定であるが、NECODEのインタプリタを別途開発し、通常の計算機上でもNODEを利用できるようにしている。その様子を図5に示す。なお、改造したP-compilerは P2と呼ばれる版であり、NODE コンパイラとしては PASCALで約4200行、NECODE インタプリタがFORTRANで約1000行程である。

5. 検 討

現在までに、アセンブリの分解経路生成問題¹²⁾、シ