プログラミング－言語・基礎・実践－ 20－4
（1995．1．27）

局所伝播法と方程式解法に基づく制約充足法

高橋 栄治　徳田 雄洋

東京工業大学 工学部 情報工学科

代表的な制約充足法の一つである局所伝播法に基づくＢｌｕｅ法は、すべての制約を充足できない場合に制約の優先順位を考慮し、要求に見合う解を出すが、各制約が局所的な情報のみを扱うために、連立した制約を解くことができなかった。そこで局所伝播法の枠組に線形方程式解法を導入した。優先順位の指標に従いながら、連立した制約がある場合でも解を求めることができる。また制約系内の一部の変数と必ず充足される必要がある制約を用いて他のすべての変数の値を決定できる場合があることから、制約変数の部分集合の本質性を考慮し、計算の効率化を図った。更に、この方法を図形定義へ応用して評価を行ない、その有効性を示した。

A Constraint Solving Algorithm Based on Local Propagation and Equation Solving

Eiji TAKAHASHI　Takehiro TOKUDA

Department of Computer Science, Tokyo Institute of Technology

We present a new constraint solving algorithm based on both local propagation and equation solving. Constraint solving algorithm, such as Blue algorithm, based on local propagation cannot solve simultaneous constraints. This is because local propagation methods require partial orderings of determination of values of variables. Equation solving methods allow us to determine values of variables in most simultaneous constraints. Our algorithm has a feature that equation solving techniques coexist within the framework of Blue algorithm without losing the efficiency of local propagation methods. Our algorithm was used for the generation of graphical user interface systems.

1 はじめに

制約とは特定のオブジェクトに要求される特性や、複数のオブジェクト間に要求される関係である。制約に基づく言語やシステムはGUI(Graphical User Interface)の構築、物理シミュレーション、アルゴリズムアニメーションなど幅広く応用されている。制約を利用することでプログラマやユーザは維持すべき関係を宣言的に指定するだけであり、その関係を維持するための手続きを記述する必要がないので、記述・変更・保守が容易であるという利点がある。

記述された制約を充足する制約充足システムで用いられる制約充足法の代表的なものとして局所伝播法[1]がある。局所伝播法は単純で実現が容易であり、高速である。ThingLabII で用いられている Blue アルゴリズム[2]は、局所伝播法を基とし、制約に優先順位を設ける制約階層[3]の理念を導入し、すべての制約を充足できない場合に制約の優先順位を考慮し、要求に見合う解を求めることができる。

しかし Blue アルゴリズムは各制約が局所的な情報のみを扱い、個々の制約の充足を組み合わせることによって制約系全体の充足を行なうために連立した制約を解くことができなかった。また制約系内の一部の変数の値と必ず充足される必要がある制約を用いて他のすべての変数の値を決定できる場合があるにも関わらず、Blue アルゴリズムでは常に制約系内のすべての値を決定してしまう。

これらの問題を解決するために本論文では局所伝播法への方程式解法の導入と制約変数の部分集合の本質性を提案する。そしてこの二点を考慮し、Blue アルゴリズムを拡張した。優先順位の指標に従いながら、連立した制約がある場合でも比較的少ない時間で解を求めることができる。

さらに、この方法を図形定義へ応用して評価を行ない、その有効性を示した。

2 関連研究

2.1 局所伝播法

最も単純で実現が容易な制約充足法として**局所伝播法 (local propagation)** がある。これは局所的な制約充足の組合せによって制約系全体の充足を行なう方法である。この方法では制約は図2．1のようなグラフで表現できる。制約変数は円で表され、制約は矩形のラベルをもつ超辺で表される。この例では"a+b=c"と"c*d=e" という制約が表現されている。制約グラフにおいて各制約がどの変数を決定するかを有向グラフで表現したものが解グラフである。図2．2の解グラフでは、a と b の値を与えると c の値が計算され、次に d の値を与えると e の値が計算されることを示している。

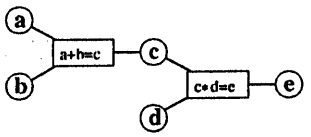

図2．1 制約グラフ

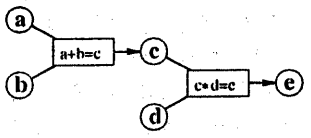

図2．2 解グラフ

局所伝播法では制約充足の過程を次の二つのフェーズに分けている。

1. どの制約をどのような順序で充足するかというプランを立てるプランニングフェーズ
2. プランに従って実際に変数の値を計算する評価フェーズ

この方法では、値のみの変更が行なわれた場合には、プランの再実行を行なうだけですむので効率が良い。

2.2 制約階層

制約階層 Hは制約の集合 $H_0,\ldots,H_N$から構成されるリストである。ここで H_n 中の制約は H_{n+1}中の制約より優先して充足される。H に含まれる制約は強度をもち、一つの階層 H_n に含まれる制約の強度は n である。強度は 0, 1,...., Nの他に required, strong, weak などの記号で表されることもある。H_0中の制約は**required** 制約と呼ばれ、必ず充足される必要がある。それに対し、H_1から H_nに含まれる制約は優先度に従って可能な限り充足すべき制約で **prefered** 制約と呼ばれる。

制約階層 Hの解は各制約変数の値の割当を決定する代入式で表される。required 制約をすべて充足する解を許容解といい、さらに許容解の中で prefered 制約を考慮し、それ以上最良な解が存在しない解を最良解という。許容解の集合 S_0と最良解の集合 Sはコンパレータ better を用いて次のように定義される。

S_0 = $\{x \mid \forall c \in H_0,\ x$ は c を充足する $\}$

S = $\{x \mid x \in S_0 \wedge \forall y \in S_0\ \neg\mathrm{better}(y, x, H)\}$

ここで better(x, y, H) は H の解として y より x の方が良いという意味である。

代表的なコンパレータである **locally-predicate-better** は次のように定義される。

locally-predicate-better$(x, y, H) \equiv$
$\exists k \in \{1, \ldots, n\}$ に対して
$\forall i \in \{1, \ldots, k\},\ \forall p \in H_i$
(yがp を充足するならばx はp を充足する)
$\wedge\ \exists q \in H_k$
(x はqを充足する $\wedge\ \neg$(yはqを充足する)).

2.3 Blue アルゴリズム

Blue アルゴリズムは、局所伝播法を基とし、locally-predicate-better な解を求めるアルゴリズムである。

各制約はメソドの集合をもっている。メソドとは制約に属するいくつかの変数を入力として出力変数の値を決定するプロシージャで、その結果このメソドをもつ制約を充足する。

入力変数の代わりに、外部からの入力値を使用して出力を決定するメソドをもつ制約を **input** 制約という。また制約に属する変数の現在の値を変えずに維持する制約を **stay** 制約という。**weakest** はシステム全体で最も弱い強度であり、強度 weakest の stay 制約はデフォルトで各変数に付けられている。

Blue アルゴリズムは、H_0 から H_n の順に、その各々の制約について以下の操作を繰り返し行なって、locally-predicate-better な解を求める。

1. 値が既に決定されている変数の集合 D に入力変数がすべて含まれており、かつ出力変数が一つも D に含まれていないメソドを一つ選ぶ。
2. そのメソドを実行して出力変数の値を決定する。
3. その出力変数を D に入れる。

なお最初 D は空集合なので、メソドが入力変数をもたない stay 制約や input 制約からアルゴリズムを開始することになる。すべての変数が D に含まれたら終了する。

3 連立する制約の解法

3.1 局所伝播法の問題点

局所伝播法では各制約が局所的な情報のみを扱うために連立した制約を解くことができなかった。つまり各制約が表している変数間の関係式を陽関数として考え、その組合せによって変数への値の割当を行なうからである。例えば図3．1 は a と b の平均が c であることを表す制約グラフであるが、a と b の値を与えても制約 "a+d=c" は変数 a,c,d の情報のみをもち、制約 "c+d=b" は変数 b,c,d の情報のみをもつために局所伝播法では変数 c,d の値を求めることはできない。

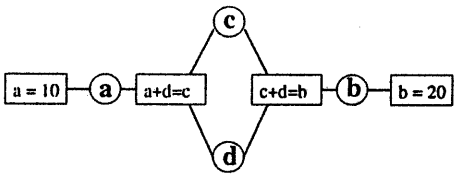

図3．1 局所伝播で失敗する例

3.2 メソドの拡張

上記の問題点を克服するために局所伝播法に方程式解法を導入するための準備としてメソドを拡張する。まず準備として変数 C_s,V_s, V_{s_in}, V_{s_out}, k を次のように定める。

C_s : 制約の集合,

$V_s \equiv \bigcup_{c \in C_s}$($c$ に属するすべての制約変数から成る集合),

V_{s_out} : 空集合でない V_sの部分集合,

$V_{s\_in} \equiv V_s - V_{s\_out}$.

k : C_sの要素数（自然数）.

メソドとは V_{s_in} に含まれる変数を入力として、V_{s_out} に含まれる変数の値を決定するプロシージャであり、V_{s_out} 中のすべての制約変数の値を決定することによって、C_s 中のすべての制約を充足する。

$k = 1$ のときのメソドは Blue アルゴリズムにおけるメソドと同様であり、個々の制約に附属する。

$k > 1$ のときのメソドは制約集合 C_s を連立方程式と見なして V_{s_out} について解き、得られた解によって V_{s_out}に含まれる変数の値を決定するプロシージャがメソドである。$k = 1$ の場合と異なり、制約充足アルゴリズムの実行中に生成される。また方程式の解が得られない場合にはそのようなメソドは存在しないこと

になる。数学的に連立方程式を解く場合とは少し異なり、入力変数は定数として扱われ、出力変数について方程式が解かれる。

$k=1$ と $k=3$ の場合のメソドの例をそれぞれ図3．2、図3．3に示す。図3．2はb, c, d, eを入力変数として出力変数aの値を決定するメソドである。このメソドを実行することによって制約"a+b*c=d-e"が充足する。図3．3はd, e, fを入力変数として出力変数a, b, cの値を決定するメソドであり、このメソドを実行することによって制約"a-b=d" "a+b+c=10" "b-c=f-e"が充足する。

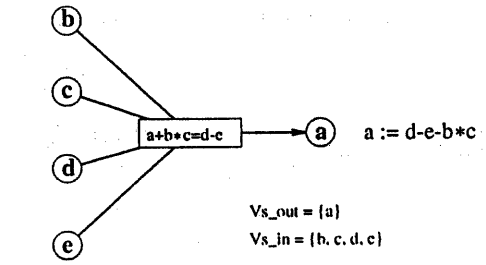

図3．2 $k=1$ の例

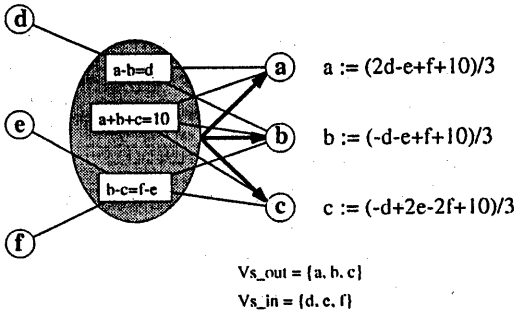

図3．3 $k=3$ の例

3.3 局所伝播法への方程式解法の導入

局所伝播法において解を求める際、変数の値を決定するためのメソドを探索するが、そのときメソドとして3.2節で述べたように、個々の制約がもつメソドだけでなく、複数の制約を組み合わせて得られるメソドも考えれば局所伝播法の枠組に方程式解法を導入できる。選ばれたメソドを実行することによって制約が充足され、既知状態が入力変数から出力変数へと伝播していく。

しかし複数の制約の組合せと入力変数、出力変数の組合せによってメソドの数は爆発的に増大してしまう。そこでメソドを探索する際の順序が非常に重要になる。また連立方程式を解くときの時間的コストも考慮しなければならない。局所伝播法の効率を保ち、連立方程式を解く回数を少なくするためには、組み合わせる制約の数が少ない方を優先する方法が一般的である。しかし強度をもつ制約を扱う場合には「制約の強度」と「組み合わせる制約の数」のどちらをどのように優先するかという問題が生じる。

4 制約変数の部分集合の本質性

4.1 本質的な制約変数集合

一般に、インクリメンタルではない制約充足アルゴリズムは制約変数すべての値を、またインクリメンタルなアルゴリズムは変更の影響を受ける制約変数すべての値を決定しようとする。しかし、制約階層においてrequired制約は必ず充足されなければならない制約であることを考慮すると、制約系内の一部の変数の値が決定すれば、その変数の値とrequired制約を充足することにより残りのすべての変数の値を決定することができる場合がある。

そこで制約変数の部分集合の本質性という概念を導入する。次の二つの性質をもつ集合を**本質的な制約変数集合**という。

1. 制約系内のすべての変数から成る集合の部分集合である
2. その集合に含まれる変数すべての値が決定すれば、後はrequired制約を充足することによって他のすべての変数の値を決定することができる

本質的な制約変数集合に含まれる変数を**本質的な変数**、またそれに含まれない変数を**非本質的な変数**とよぶ。

4.2 本質的な制約変数集合の決定法

本質的な変数の決定法を示す。

1. 対象とする制約階層 H に含まれるすべての変数、すべてのrequired制約、デフォルトで各変数に付く強度weakestのstay制約から、新たな制約階層 H' を構築する
2. 利用する制約充足アルゴリズムによって、H' を解く

3. 得られた解において、強度 weakest の stay 制約によって決定される変数すべてから成る集合を、本質的な制約変数集合とする

この決定法は利用する制約充足アルゴリズムに依存する。required 制約とデフォルトの stay 制約を組み合わせて変数の値を決定するアルゴリズムでは本質的な制約変数集合を決定することができない。

Blue アルゴリズムを利用する場合の本質的な制約変数集合の例を図4．1に示す。この例では本質的制約変数集合 E は {b, c} となる。非本質的な変数は E と required 制約を用いて値を決定することができる E としてはこれ以外に {b, d}, {f, c}, {f, d}, {c, d}, {c, e} が考えられる。

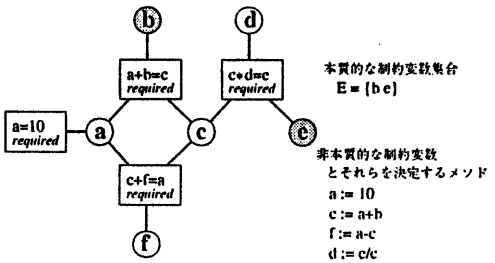

図4．1 本質的な制約変数集合の例

4.3 本質的な制約変数の効用と特徴

従来、制約を解くときには制約変数すべての値を決定していたが、本質的な制約変数集合の本質性を考慮し、本質的な制約変数集合の要素すべての値を決定するようにアルゴリズムを改良すれば計算を効率化することができる。

また本質的な制約変数集合は以下のような特徴をもつ。

- 一般に本質的な制約変数集合の組合せは複数通り存在する。
- required 制約同士で矛盾するものが存在する場合、決定することができない。
- 変数の追加・削除、required 制約の追加・削除が行なわれなければ本質的な制約変数集合は変化しない。
- 実際に制約充足アルゴリズムを実行するときに、変数の値が決定する順序は必ずしも

  本質的な変数 ⟶ 非本質的な変数

  とは限らない。

5 制約充足アルゴリズム

3.2 節の局所伝播法への方程式解法の導入と、4.1 節の制約変数の部分集合の本質性を考慮し、Blue アルゴリズムを拡張した制約充足アルゴリズムを提案する。

5.1 アルゴリズムの概略

このアルゴリズムは Blue アルゴリズムを基にしており、次の特徴が共通している。

- 強さをもった制約を扱うことができる。
- プランニングフェーズと評価フェーズがある。
- まずすべての変数を未決定とし、先に実行されるメソドから順に決定され、既知状態が伝播していく。
- 循環・衝突がある解は導出されない。

Blue アルゴリズムでは変数の値を決定するために個々の制約がもつメソドだけを利用するのに対し、このアルゴリズムでは複数の制約を連立方程式として解いた解もメソドとして利用する。

locally-predicate-better な解を求めるために制約階層 H において H_{n+1} より H_n が優先される。しかし連立方程式として組み合わされる制約が、すべて一つの階層に含まれるとは限らない。二つ以上の階層に跨った制約が組み合わされる場合は、それらの中で最も強度の弱い制約の強度を優先度の比較の対象にする。さらに最弱の強度が同じ場合には、計算効率向上のために組み合わされる制約の数が小さい方を優先する。

さらに制約変数の部分集合の本質性を考慮し、入力となる制約から本質的な変数の値を求めるまでを計算するようにメソドの順序列を再構成する。

5.2 プランニングフェーズ

プランニングフェーズは手続き **Planning** に相当する。手続き Planning では、手続き Planning1, Planning2, Planning3 を逐次に実行する。得られる解であるメソドの順序列は大域変数 *SolutionOrder* にストアされる。

5.2.1 初期化

手続き Planning1 では、まず次の二つの条件を満たす required 制約 c とそれに属する非本質的な変数 v の組合せすべてを制約系から取り除く。

1. v は c とデフォルトの stay 制約以外の制約には属さない。

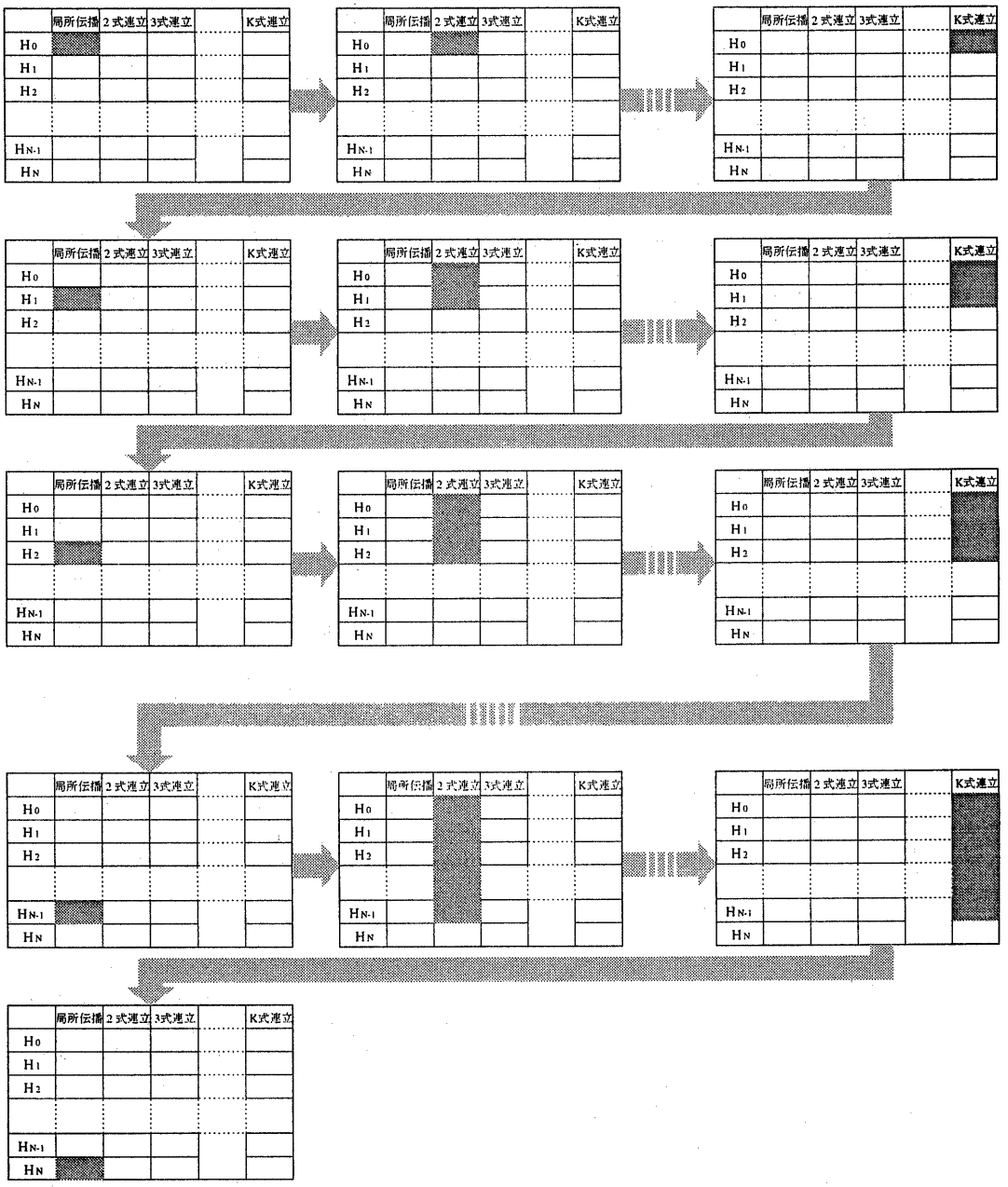

図５．１メソド探索の優先順位

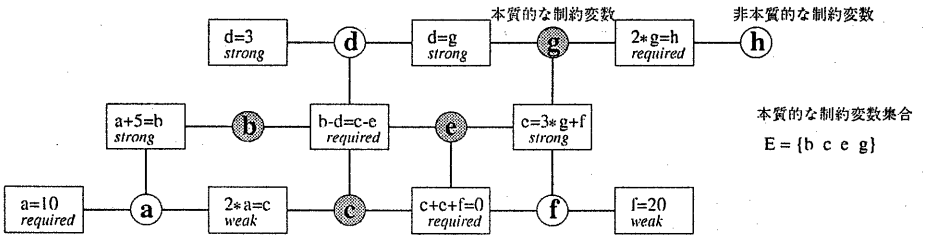

図5．2 実行例 (制約グラフ)

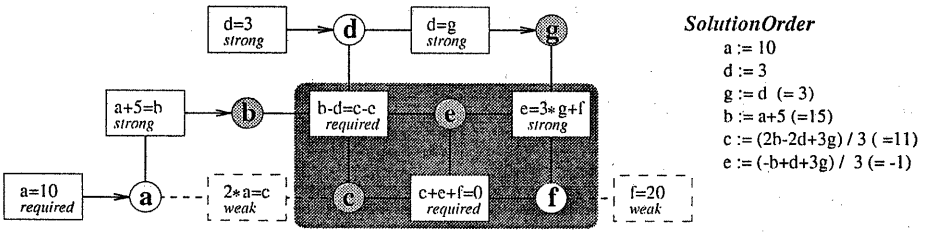

図5．3 実行例 (解グラフ)

2. cに属する非本質的な変数はvのみである。

次にすべての変数を未決定とし、大域変数 *SolutionOrder* を空列とする。

5.2.2 メソドの探索

手続き Planning2 では図5．1に示す順序で先に実行されるメソドから順に探索していく。図中の表の縦軸は個々の制約階層 H_n $(0 \leq n \leq N)$、横軸は組み合わされる制約の数 k を表す。表中の網かけの部分 H' から k 個の制約を選んで組み合わせて実行可能なメソドを探索する。実行可能なメソドとは k の値によって次のように定義できる

- $k = 1$(局所伝播) のとき、H'の元である c がメソド m をもち、m の出力変数が未決定、m の入力となるすべての変数の値が決定している。このとき m が実行可能なメソドである。
- $k > 1$(k式連立) のとき、まず変数 C_s, V_{s_out}, V_{s_in} を次のように定める。

  C_s : H'の元を要素とする制約の集合,

  $V_{s\_out} \equiv \bigcup_{c \in C_s}$($c$ に属するすべての未決定の制約変数から成る集合),

  $V_{s\_in} \equiv \bigcup_{c \in C_s}$($c$ に属するすべての決定の制約変数から成る集合).

  C_s の要素数が k、V_{s_out} の要素数が 1 以上 k 以下、C_s を連立方程式として V_{s_out} について解き、V_{s_out} を一意に決定する解が得られたとき、その解が実行可能なメソドである。V_{s_out} を一意に決定する解が存在するかどうかが V_{s_in} に含まれる変数の値に依存するとき、一意に決定する解が存在しない場合と同様に扱う。

一度実行可能なメソドが見つかると、そのメソドを *SolutionOrder* の先頭に入れ、出力となる変数すべての値を決定とする。次に図5．1の表中の上方、つまりより制約の強度の強い階層にその影響を伝播させる。また計算の効率化のために kが増加する際に表中の k' $(1 \leq k' < k)$ の部分を探索したときの情報を利用する。すべての本質的な変数の値が決定となったらPlanning2 は終了する。

5.2.3 メソドの再選択

手続き Planning3 では Planning2 で得られた解であるメソドの順序列 *SolutionOrder* に対し、メソドの再選択を行なう。*SolutionOrder* の最後尾、つまり最後に実行されるメソドから順に探索し、本質的な変数を決定するメソド、および本質的な変数を決定するために用いられる変数を決定するメソドを選択して新たな

SolutionOrder を構成し、これを手続き Planning によって得られる解とする。

5.3 評価フェーズ

評価フェーズは手続き Exec_Plan に相当する。手続き Exec_Plan では *SolutionOrder* の先頭のメソドから順に実行し、変数に値を代入していく。

5.4 実行例

制約充足アルゴリズムによって制約階層の解が得られる例を示す。

まず図5．2に示す制約グラフについて考える。制約変数と required 制約から本質的な制約変数集合 E は {b c e g} に決まる。

手続き Planning1 では変数 h と制約"2*g=h" が制約グラフから取り除かれる。これは非本質的な変数 h が本質的な変数 g によって値が決定されることが明らかであるからである。またすべての制約変数と制約の初期化が行なわれる。

手続き Planning2 では図5．1の優先順位に従い、メソドの順序列を決定する。"b−d=c−e"、"c+e+f=0"、"e=3*g+f" の三つの制約は連立方程式として変数 c, e, f について解かれる。また制約の優先順位を考慮し、"2*a=c"、"f=20" の二つの制約は充足されない。

手続き Planning3 では Planning2 で得られたメソドの順序列から変数 f を決定するメソド "f:=(−b+d−6*g)/3" が取り除かれる。このメソドは本質的な制約変数集合 E={b, c, e, g} の要素を決定するためには必要ないからである。

このようにして図5．2の制約グラフを制約充足アルゴリズムによって解いた解グラフが図5．3である。

6 図形定義への応用

5節で提案した制約充足アルゴリズムの図形定義への応用を示す。

6.1 基本図形

ここで用いる図形定義システムでは line、rectangle などの基本図形がライブラリとして用意されていると仮定する。これらは図6．1のように定義されているクラスのインスタンスである。ここでの属性とはその図形の中点の座標、高さ、幅などを表す。ここでのクラス定義中の制約とは属性間の関係を表すものと属性のデフォルト値を指定するものがある。

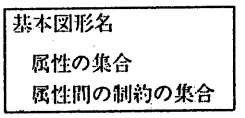

図6．1 基本図形のクラス定義

```
class : rectangle
attributes : nw.x, nw.y, width, height            ; essential
             ne.x, ne.y, sw.x, sw.y, se.x, se.y   ; inessential
             center.x, center.y, diagonal         ; inessential
constraints : width² +height² = diagonal²
              nw.x + width = ne.x
              nw.y = ne.y
              nw.x = sw.x
              nw.y + height= sw.y
              nw.x + width = se.x
              ne.y + height = se.y
              ne.x + width / 2 = center.x
              ne.y + height / 2 = center.y
```

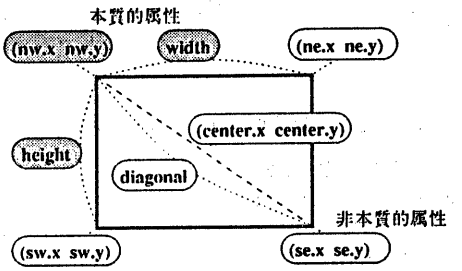

図6．2 rectangle のクラス定義の例

図6．2に基本図形 rectangle のクラス定義の例を示す (デフォルト値指定の制約は省略)。

6.2 ユーザによる記述

図形定義システムを利用するユーザは、表示したい基本図形名を列挙し、次に制約を記述する。これらの制約は次のいずれかである。

- 基本図形インスタンスの属性値の指定 (input 制約、stay 制約)
- 単一の基本図形インスタンス内の属性間の関係
- 複数の基本図形インスタンスがそれぞれもつ属性間の関係

また制約にそれぞれ強度を付けることができる。

6.3 図形と制約系との対応

ユーザが列挙した基本図形インスタンス内のすべての属性を制約系の変数とする。そしてユーザが列挙した基本図形インスタンスのクラス定義中の制約と、ユーザが記述した制約を制約系の制約とする。制約階層 H は図6．3のように構成する。そして5節で述べたアルゴリズムによって制約充足を行なう。

H_0	クラス定義中の制約（デフォルト値指定以外）
H_1	ユーザ定義の制約
H_2	
⋮	
H_{N-1}	
H_N	クラス定義中のデフォルト値指定制約

図6．3 制約階層の構築

図形における本質的な属性集合は、制約系における本質的な制約変数集合に対応する。本質的な属性集合の要素である属性すべての値が決定すれば、後はデフォルト値指定以外のクラス定義中の制約、つまり required 制約を充足することによって他のすべての属性の値を決定することができるという性質をもつ。つまり本質的な属性のみを用いて図形を表示することができる。

またユーザは required 制約を記述することができず、かつ異なる基本図形インスタンスのもつ属性間には required 制約は存在しないので、クラス定義の段階で本質的な属性集合を決定しておくことができる。したがって制約の追加・削除が行なわれる度に本質的な属性集合を決定する計算を行なう必要はない。

6.4 本質的な属性集合による効用

ユーザが制約を記述する際、本質的な属性を特に意識する必要はなく、ただユーザが関心をもっている属性についての制約を記述すればよい。

属性集合の本質性を考慮しない場合、ユーザによる記述を容易化するために各基本図形がもつ属性の数を増加させると、対応する制約系の変数の数も増加してしまい、それに伴って変数の総数に対して図形表示に直接使用される変数の数の割合が減少し、計算の効率が悪化する。逆に計算の効率の向上のために各基本図形がもつ属性の数を減少させると、ユーザが関心をもつ属性が存在しなかった場合、制約の記述が容易でなくなる。属性集合の本質性を考慮することによってこれらの問題を解決できる。

7 評価

5節で述べた制約充足アルゴリズムを Scheme の処理系である Elk を用いて Sun SPARCstation 10 上で実現した。連立方程式の解法としてはガウス消去法を用い、解法可能な連立方程式のクラスは slightly-nonlinear-equations とした。本節では Blue アルゴリズムと比較しながら評価を行なう。

7.1 Blue アルゴリズムとの解の相違

本研究で提案するアルゴリズムと Blue アルゴリズムに同じ制約を与えて、それぞれの解を比較する。

6 節で述べた図形定義システムを基に、基本図形 rectangle の二つのインスタンス r1 と r2 について図7．1のように制約を記述した。この記述例では r1 の左上の頂点の座標が (10, 20)、r2 の右下の頂点の座標が (40, 40)、r1 の右下の頂点と r2 の左上の頂点が接しており、r2 の幅は r1 の幅の2倍、r2 の高さは r1 の高さの3倍であるという制約を表している。また図中では省略したが記述した制約はすべて強度 strong として計算する。

```
r1 : rectangle
r2 : rectangle
  r1.nw.x = 10                r1.nw.y = 20
  r2.se.x = 40                r2.se.y = 40
  r2.width = 2 * r1.width     r1.se.x = r2.nw.x
  r2.height = 3 * r1.height   r1.se.y = r2.nw.y
```

図7．1 制約の記述例

図7．1の制約を Blue アルゴリズム、本研究の制約充足アルゴリズムを用いて解き、得られた解を用いて表示した図形をそれぞれ図7．2、図7．3に示す。Blue アルゴリズムでは制約"r2.width = 2 * r1.width" と "r2.height = 3 * r1.height" は充足されず、r1 の width と height はデフォルト値が用いられている。それに対して本研究のアルゴリズムでは"r1.ne.x + r1.width = r1.se.x", "r2.ne.x + r2.width = r2.se.x", "r1.se.x = r2.nw.x", "r2.width = 2 * r1.width" の四つの制約を一つの連立方程式として、また"r1.ne.y + r.height = r1.se.y", "r2.ne.y + r2.height = r2.se.y","r1.se.y = r2.nw.y", "r2.height = 3 * r1.height" の四つの制約を一つの連立方程式として解くことによって図7．1に記述された制約をすべて充足することができる。

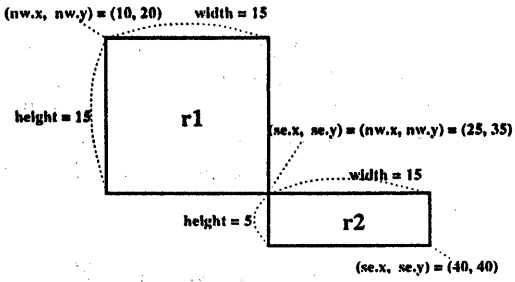

図7．2 Blue アルゴリズムによる解

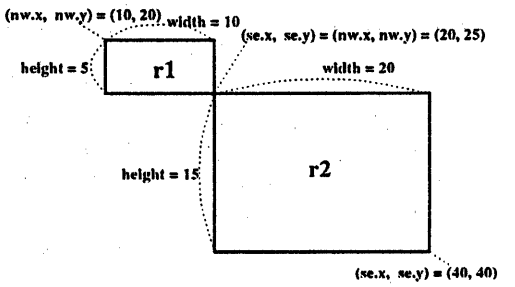

図7．3 本研究のアルゴリズムによる解

またBlue アルゴリズムでは基本図形 rectangle のもつすべての属性の値を計算するが、本研究の制約充足アルゴリズムでは本質的な属性の値を計算するので、属性 ne.x, ne.y, sw.x, sw.y, center.x, center.y, diagonal およびこれを含む required 制約は制約系から除外してから計算が行なわれる。

このように連立する制約がある場合には、本研究の制約充足アルゴリズムは Blue アルゴリズムと比較して、より制約階層の優先度に適合した解を導出する。さらに制約変数の部分集合の本質性を導入することによって計算の効率化を図っているが、複数の制約を組み合わせるためにメソドの探索が複雑になるので、計算時間は Blue アルゴリズムより遅いことが予想される。

8 おわりに

本論文では、強さをもった制約を扱うことができる Blue アルゴリズムを拡張し、複数の制約を組み合わせる方程式解法を局所伝播法の枠組に導入し、さらに制約変数の部分集合の本質性を考慮した制約充足アルゴリズムを提案した。そしてそれを応用した図形定義システムを示し、さらに Blue アルゴリズムと比較し、評価を行なった。

問題点としては連立方程式を解く際に同じ連立方程式を何度も解いてしまう場合があることである。これを解決するためには方程式の解を記憶しておく機構が必要である。また方程式を解いてメソドを得る際に、解が一意に存在するかどうかが入力変数の値に依存することがある。本論文ではそのような場合は方程式が一意に存在しない場合と同様に扱っているが、プランニング時に制約変数の値に応じて条件分岐を行なえばこの問題は解決できると思われる。

locally-predicate-better であることの証明、インクリメンタルなアルゴリズムへの拡張、さらに複数の制約が同時に追加・削除されたときに効率的な計算を行なうアルゴリズムへの拡張などが今後の課題である。

参考文献

[1] Steele,G.L. and Sussman,G.J.Constraints. In *APL Conference Proceedings part 1*, pp. 208-225. APL Quote Quad, June 1979.

[2] John Harold Maloney. Using Constraints for User Interface Construction. Technical report, University of Washington, Department of Computer Science and Engineering, August 1991.

[3] Alan Borning, Robert Duisberg, Bjorn Freeman-Benson, Axel Kramer and Michael Woolf. Constraint Hierarchies. In *OOPSLA '87 Conference Proceedings*. pp. 48-60. ACM, October 1987. In-Proceedings

[4] Wm Leler. *Constraint Programming Languages :Theier Specification and Generation.* Addison-Wesley, 1988.

[5] 柿沼 伸夫: 局所伝播法を用いた制約充足アルゴリズムの効率化に関する研究. 修士論文. 東京工業大学. 1993