人 工 知 能 81－5
（1992. 3. 5）

依存関係管理による制約充足問題解決

古渡 聡† 長尾 和彦‡ 石井 直宏†

†名古屋工業大学 ‡弓削商船高等専門学校

あらまし

制約ネットワークによる制約充足問題(CSP)を解決する手法の1つに局所制約伝播法がある。この手法は個々の制約ごとに充足することで効率的な探索を実現するが、問題全体を把握しづらいということや、制約の再起動を避けられないといった問題がある。本稿では、これらの問題点が伝播されるべき情報をシステムが知らないことが原因であるとし、制約ネットワーク内の値の間に存在する依存関係を管理することで効率的に局所伝播を行なう方法を紹介する。また、その依存関係を用いて大域解を形成する手順も紹介する。

Constraint Satisfaction by Managing Dependency

Satoshi Kowatari† Kazuhiko Nagao‡ Naohiro Ishii†

†Nagoya Institute of Technology ‡Yuge Mercantile Marine College

Abstract

Local constraint propagation is one of the methods solving Constraint Satisfaction Problem (CSP) in the constraint network. This method realizes an efficient search by satisfying primitive constraints. But, there are some defects; the system using this method can not comprehend the problem as a whole and it activates the same constraints repeatedly. In this paper, we regard that these problems are caused by the property of local constraint propagation, that is, the system does not have the imformation to propagate in the network. Here, we present a method to carry out an local constraint propagatoin efficiently by recording dependency relations in the constraint network. We also present a way of optaining global solution by using the dependency.

1 はじめに

制約とは、物体間に存在する依存関係を宣言的に記述したものであり、特に、単純な制約を組み合わせることで対象とする問題を表現したものを制約ネットワークと呼ぶ。制約解消系の一つである局所制約伝播法(以下、局所伝播と記す)は部分的な問題(つまり、個々の制約)を中心に考えることで効率的な探索を実現している。しかし、制約の実行による変化をその制約に隣接する制約を実行することで周囲に伝えていく方法をとるために、1つの制約を何度も起動したり、扱う情報が変数のとる値の領域のみであることから、他の値との関係を把握しづらいという欠点がある。

本稿では、制約ネットワーク内の値の間に存在する依存関係を、制約を満たす値の組合せを用いてATMS[1][2]におけるラベルのように明示的に記録し、その明示化された依存関係を管理することによって、局所伝播を実現する方法について述べる。また、ラベルを大域解形成時の情報として用いる方法についても述べる。

2 制約ネットワークでの制約充足

本稿で対象とする制約充足問題は、有限領域(値の個数が有限個)[1]から値をとる変数に対して、全ての制約を満たす各変数への値の割当を求める問題であり、次のように定式化されるものである。

制約充足問題は、変数の有限集合 $\mathcal{X}=\{X_1,\cdots,X_n\}$ および制約の集合 C からなり、変数はそれぞれ値を領域 $D_1,\cdots,D_n$ からとる(ただし、初期的に領域が決定されていない変数もある)。そして、$k(\leq n)$ 変数間の制約(k 項制約)$C_{i_1,\cdots,i_k}(\in C)$ は、変数集合 $\{X_{i_1},\cdots,X_{i_k}\}$ と、直積 $D_{i_1}\times\cdots\times D_{i_k}$ の部分集合(互いに整合のとれた変数の値組を表わす)または変数間の関係式の組みで表現される。

[1]ただし、全ての変数が初期的に領域をもっている必要はない。例えば、'X+Y=Z' のような制約は、3変数のうち2つまで領域がわかっていれば、制約の実行によって残りの1つの領域を明らかにできる。

2.1 制約ネットワーク

実際の問題を制約で表現する場合、ある1つの制約で全ての関係を記述することは可能であるが、そのように記述された制約は非常にわかりづらいものとなってしまう。したがって、1つの制約は対象とする問題のある部分的な局面のみを記述するようにして、制約の集合をもって、問題全体の関係を表記すべきである。ただし、個々の制約をただ無意味におくだけでは問題全体を表現することはできない。何らかのつながりが必要である。この制約間の相互結合は制約が関係している変数の参照によってなされる。このようにして得られた制約の相互結合全体を制約ネットワークと呼ぶ。つまり、制約ネットワークとは変数を共有するような制約によりネットワークを構成することで問題全体を表現する制約表記法である。

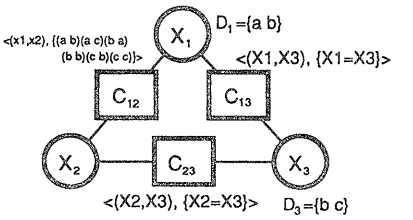

図 1: 制約ネットワーク

$Con(X_i)$ を変数 X_i につながっている制約の集合とし、また、$Var(C)$ を制約 C が関連する変数の集合としたとき、制約ネットワークにおける無矛盾性は次のように定義される[3]。

(局所無矛盾性)
n 変数の制約ネットワークが局所無矛盾であるとは、変数とその値の組み合わせを v_i=X_i:d(ただし、$d\in D_i$)とし、その集合を V_i としたとき、

$$\forall i \in \{1,\cdots,n\}\ \forall C \in Con(X_i)\ \forall v \in V_i$$
$$\exists v_1 \in V_1,\cdots,v_{i-1} \in V_{i-1},$$
$$v_{i+1} \in V_{i+1},\cdots,v_n \in V_n :$$
$$(v_1,\cdots,v_{i-1},v,v_{i+1},\cdots,v_n)_{|Var(C)} \in_S C$$

を満たすときである。□

(大域解)
n 変数制約ネットワークにおいて、値割当 $(v_1,\cdots,v_n)$ が大域的に無矛盾である (大域解) とは、

$$\forall C \in \mathcal{C} :\ (v_1,\cdots,v_n)_{|Var(C)} \in_S C$$

となることである。□

ただし、$\in_S$ は制約を充足するという意味である。また $V_{|X}$ は、$V=(v_1,\cdots,v_n), X=\{X_{i_1},\cdots,X_{i_k}\}$ $(k \leq n)$ としたとき $V_{|X}=(v_{i_1},\cdots,v_{i_k})$ である。

次節では、局所無矛盾性を実現するための戦略の一つである局所伝播について述べる。

2.2 局所伝播

制約ネットワークを制約解消するための制御戦略に局所伝播という方法がある。局所伝播とは、ある制約 C の実行によりその制約につながる変数の値の領域に変化が生じたとき、その変化を制約 C に隣接している制約 (制約 C と変数を共有するような制約) 全てに伝えるような伝播方法である。アルゴリズムは次のようになる。

```
LC(CL)
begin
1. while CL is not empty do:
2.     select C from CL.
3.     XO ← REDUCE(C).
4.     if XO is not empty then
5.       foreach X ∈ XO do:
6.           CL ← CL ∪ Con(X).
7.       end.
8      fi
9.     delete C from CL.
10. end.
end
```

引数 CL は、初期的に実行可能な制約の集合である。また、手続き REDUCE(C) は、制約 C を評価して変数の領域を減少させる。そして、領域が減少した変数の集合を返す。

局所伝播において、値は制約やノードの局所的な集まりから演繹的に推論され局所的に管理されることから、生成－検査法などにみられるようなバックトラック的組合せ爆発とは無縁である。しかし、隣接制約の起動によって変化を周囲の変数に伝えるため、同じ制約を何度も起動してしまう (もちろん、同じ状態で起動することはない) ことを防ぐことができない。また、システムの管理する情報が変数の値の領域のみであることから、領域内に保持されている各値が隣接変数がどのような状態のときかを知ることができず、問題全体の解 (大域解) を形成する際、制約評価を行なう必要がある、などの問題が存在する。

このような局所伝播の問題は全て、伝播させるべき情報をシステムが知らないことが原因であると考えられる。次に続く 3 節で、制約ネットワーク内の変数の値間に存在する依存関係と、その表現と管理の仕方について述べる。そして 4 節において、3 節で局所的に無矛盾に保たれたネットワークから大域解を導出する手順について述べる。

3 依存関係の管理による制約伝播

本稿で紹介する制約伝播は、図 2のように、問題解決部とラベル管理部という 2 つの動作部から構成される。

問題解決部 制約ネットワークの形状 $(\mathcal{X},\mathcal{C})$ と変数の値領域を情報としてもち、制約の実行 (評価) を行なう。

ラベル管理部 問題解決部での制約実行結果を (局所的に) 無矛盾に保つ。制約伝播を管理する。

本節では、まず、制約ネットワークにおける値間の依存関係について述べ、その後、その依存関係

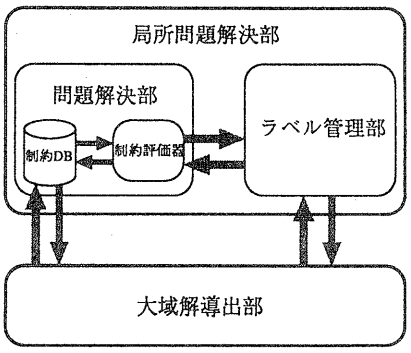

図 2: システム図

を用いてどのように制約伝播を行うかについて述べる。

3.1 依存関係の明示化

制約ネットワークにおいて、変数の間には順序関係が存在せず、各変数の値が制約を通して相互に影響し合うという関係が存在する。例えば、次のような例を考える。

制約C_{23}が'$X_2 = X_3'$のとき
起動前： $D_2 = \{a, b, c\}$, $D_3 = \{b\}$
起動後： $D_2 = \{b\}$, $D_3 = \{b\}$

上の例の場合、変数 X_2=b ならば変数 X_3=b、または、変数 X_3=b ならば変数 X_2=b ということではなく、値組 (X_2:b, X_3:b) が制約 C_{23} で真であるから X_2 が値 b を、そして X_3 が値 b を領域にもつことができるといえる。

このような変数の値間に存在する依存関係を表現するために制約実行後の局所的に無矛盾な値の組み合わせを用いる。つまり、$C_{i_1,\cdots,i_k}$ が真となるような値組 $(v_{i_1}, \cdots, v_{i_k})$(ただし、$v_{i_1} \in V_{i_1}, \cdots, v_{i_k} \in V_{i_k}$) を値間の依存関係を表現するための最小単位とするわけである。この値の組み合わせを**仮説**と呼び、仮説を構成しているデータに付加する事によって、後にデータに付加された仮説を参照することでそのデータがどの値に影響を及ぼすか（及ぼされるか）を知ることができる。したがって、k 項制約 $C_{i_1,\cdots,i_k}$ に属す仮説 $A^{C_{i_1,\cdots,i_k}}$ は次のように与えられる。

$$A^{C_{i_1,\cdots,i_k}} = \{(v_{i_1}, \cdots, v_{i_k}) | \\ (v_{i_1}, \cdots, v_{i_k}) \in_s C_{i_1,\cdots,i_k}\}$$

このことから、データ v に付加される、制約 C に属する仮説の集合 A_v^C は、次のようになる。

$$A_v^C = \{\alpha \in A^C | \ v \ appears \ in \ \alpha\}$$

前述の例においては、$A^{C_{23}} = \{(X_2{:}b, X_3{:}b)\}$ であることから、変数 X_2 の値 b と変数 X_3 の値 b には仮説 (X_2:b, X_3:b) が付加されることになる。

制約ネットワークにおいて、大抵の場合、変数は複数の制約関係によって制約を受けている。したがって、それぞれの制約関係における仮説を組み合わせることでその変数がその値をとるための（制約を通して）隣接している変数の値の組み合わせを表すことができる。この仮説の組み合わせを**環境**と呼ぶ。そして、環境の集合をラベルといい、このラベルをデータに付加することで、値間の依存関係を表現する。データ v のラベル (環境の集合)L_v は、次のように与えられる。

$$L_v = \prod_{C \in Con(v)} A_v^C$$

例えば、図 1のネットワークにおける変数 X_2 に属するデータ $v(\in V_2)$ のラベルは、$A_v^{C_{12}} \times A_v^{C_{23}}$ となる (図 3)。

また、仮説の定義より、ラベルとデータの間には次のような性質が存在することがわかる。

[性質 1]

a. 値が矛盾となればそのラベルを形成している仮説全てが矛盾となる

b. ラベル内の全環境（または仮説）が矛盾となったときその値は矛盾となる

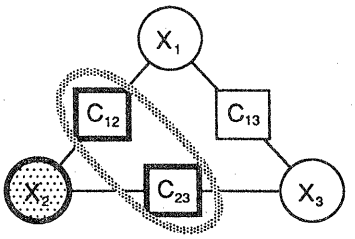

図 3: データと環境

2.1 で述べた局所無矛盾性の定義は仮説とラベルを用いて次のようにいうことができる。

(局所無矛盾性)'
n 変数の制約ネットワークが局所無矛盾であるとは、

$$\forall i \in \{1,\cdots,n\}\ \forall C \in Con(X_i)\ \forall v \in V_i$$
$$\forall \alpha \in A_v^C\ \forall v' \in \alpha :\ \alpha \in A_{v'}^C$$

であることである。□

次に [性質 1] を用いることによって制約の再起動無しに制約伝播を実現する方法を述べる。

3.2 依存関係の管理

問題解決部は、制約伝播をラベル管理部にまかせることから、制約の実行のみを行なえばよく、それも全ての制約を 1 度だけ評価すればよい。したがって、2.2 で述べた手続き LC は次のように変更される。

```
LC'(CL)
begin
1.  CN ← C.
2.  while CL is not empty do:
3.      select C from CL.
4.      delete C from CN.
5.      XO ← LABEL(EVAL(C),Var(C)).
6.      if XO is not empty then
7.        foreach X ∈ XO do:
8.          CL ← CL ∪ {C ∈ Con(X)|
                                C ∉ CN}.
9.        end.
10.     fi
10.     delete C from CL.
11. end.
end
```

手続き EVAL(C) は LC における REDUCE(C) と同様な処理を行なう手続きだが、返値が制約 C を満たす値組と C を充足しないデータ (矛盾) のリストを返すという点で異なる。

Step5. の LABEL がラベル管理部の処理であり、問題解決部からデータ, 仮説, 矛盾を受けとり、次のような処理を行う。

1. データのラベルの変更
2. 問題解決部への伝播後の値領域通知

また、1. のデータのラベルの変更は、次の 2 つの処理に分けられる。

a. 仮説による環境の作成と変更

b. 矛盾の解消

これは次のようなアルゴリズムになる。

```
LABEL(AL,XO)
begin
1. NG ← get nogood from AL.
2. AS ← get assumption from AL.
3. if AS is not empty then
4.    UPDATE(AS,XO).                  —(a)
5. fi
5. if NG is not empty then
6.    NOGOOD(NG).                     —(b)
7. fi
end
```

(a)(b) はそれぞれ、ラベル変更の 2 つの処理 a,b に対応している。次に (a)(b) それぞれの処理について述べる。

3.2.1 環境の作成と変更

定義より、仮説は制約で許可される具体的な値の組み合わせであり、データを支持する環境を形成する仮説には必ずそのデータ自身が含まれている。したがって、ラベル管理部にあるデータ $v(\in V_i)$ が存在する場合は、そのデータのラベルを問題解決部から受けとった仮説の集合との直積に変更すればよい。アルゴリズムは次のようになる。

```
UPDATE(AS,XO)
begin
1. foreach X ∈ XO do:
2.    foreach N ∈ Node(X) do:
3.       AL ← {A ∈ AS| N.Datum appears in A}.
4.       N.Label ← {E ∪ {A}|
                        E ∈ N.Label and A ∈ AL}.
5.    end.
6. end.
end
```

上の N.Datum と N.Label はそれぞれ、データとそのラベルを表すアルゴリズム中のデータ構造である。

3.2.2 矛盾の解消

ラベル管理部において矛盾とされるものには、データと仮説の 2 種類がある。前者は問題解決部から送られてくるものであり、後者はこれから述べるラベル管理部における (局所的な) 矛盾解決手順によって発見され制約伝播をさらに進めるための情報として使用されるものである。それぞれ、次のようなルールに基づいて処理される。

矛盾なデータを支持する環境の削除

[性質 1a] より、データが矛盾となればそのラベルを形成している仮説全てが矛盾となることがわかる。システムは、そのようなデータのラベル内の仮説を全てを矛盾な仮説として記録し、次のルールを実行する。ただし、はじめて具体化されたデータはそのラベルをもたないことから、次のルールは実行されない。

矛盾な仮説を含む環境の削除

前のルールの実行によって矛盾ということが判明した仮説を含む環境は、環境が仮説の連言によって表されていることから、もはやその時点において無矛盾ではなくなる。 したがって、システムはそのような環境をそれが含まれるラベルから削除する。 このような矛盾な仮説を含む環境は、仮説を構成している各データのラベルに見つけられる。
また [性質 1b] から、矛盾となった仮説によって、その環境がすべて矛盾となってしまったようなデータは、支持される環境がなくなったことから矛盾となることがわかる。 そして、これら矛盾となったデータのラベルを構成する環境を用いて、このルールをさらに進める。

前の 2 つのルールは実際には次のようなアルゴリズムにより実現されている。

```
NOGOOD(NG)
begin
1.  while NG is not empty do:
2.      select NA from NG.
3.      foreach A ∈ ⋃_{E∈NA.Label} E do:
4.          foreach N in A do:
5.              NE ← {E ∈ N.Label | A ∉ E }.
6.              if NE is empty then
7                   NG ← NG ∪ {N}.
8               else N.Label ← NE.
9.              fi
10.         end.
11.     end.
12.     delete NA from NG.
13. end.
end
```

この手続きによって、次の " 初めて実行される制約 " の評価前に、ネットワークの局所無矛盾性を制約の再評価無しに実現している。

4 大域解の導出

大域解の形成は図 2で示されているように、大域解導出部が問題解決部とラベル管理部から情報を受けとることで行われる。また 3.1 の仮説の定義から、その環境に同じ仮説をもつ値は同じ制約を同じ状況で充足している値であるということがいえる。よって、大域解について次のような性質がいえる。

[性質 2]
大域解 G を構成する各データは次のような条件を満たす。

$$\forall v_i \in G\ \forall C \in Con(v_i)\ \forall v_j \in G_{|Var(C)}(i \neq j)$$
$$\exists \alpha \in A^C_{v_i}\ \exists \alpha' \in A^C_{v_j} :\ \alpha = \alpha' \quad \square$$

上の性質から、次のような条件を満たすデータ同士 (v_i, v_j) を組み合わせていくことで大域解を求めることができることがわかる。

1. $\exists E \in L_{v_i}\ \exists E' \in L_{v_j} :\ E \cap E' \neq \phi$
2. $\exists E'' \in \{E \cup E' |\ E\ and\ E'\ satisfy\ 1.\} :$
$$\#E'' = \#(Con(v_i) \cup Con(v_j))$$

ただし、$\#E''$ は集合 E の大きさを表す。

前述の条件を満たすようなデータ同士の組み合わせを**大域解の断片**と呼ぶ。そして、条件 2. を満たすような環境の集合を、大域解の断片が制約をどのように充足しているかを示すためにラベルとして付加する。(今後、大域解の断片とそのラベルは、それぞれ、データとそのラベルと同等の扱いをする。つまり上の条件において、v_i と v_j のどちらか一方が大域解の断片で他方が新たに加えられるデータを表していることになる。) 条件 1. は [性質 2] を表しており、条件 2. は次のような状況が起こることを防ぐ。

$$\exists E \in L_g\ \exists \alpha, \alpha' \in E$$
$$\exists C \in Con(g) :\ \alpha, \alpha' \in A^C$$

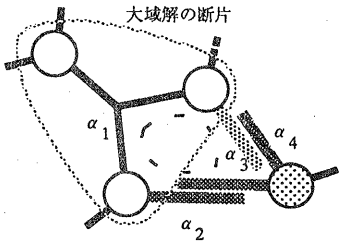

図 4: 条件 2. が必要となる状況

ただし、上の L_g は大域解の断片 $g = (v_{i_1}, \cdots, v_{i_k})$ $(k \leq n)$ のラベルである。これは、図 4のように新しいデータを加えるとことにより閉路が完成されるときに生じる (丸がデータで、直線が仮説)。

大域解の断片にどのデータを加えるかは、環境 (仮説) を参照することで容易に行なうことができる。また、3.1 で述べたように制約ネットワーク内の変数に割り付けられる値の間の依存関係には順序関係がないため、具体化を行なうときの起点となる変数はネットワーク内の変数のどれでも良い。しかし、それ以降の具体化の順序に関しては、大域解の断片を形成するにあたってデータの環境を用いる (正確には環境に含まれる仮説を用いる) ことから、大域解導出部へデータを供給した変数に隣接していて、かつ、まだ大域解導出部にデータを与えていない変数がデータを供給するという順序にする必要がある。

5 評価

本節では LC と LC' を比較する。ただし実現時には、メモリ上の効率から手続き UPDATE の Step3. においては直積ではなく集合和を用いている。つまり、データのラベル L_v を

$$L_v = \bigcup_{C \in Con(v)} \{A^C_v\}$$

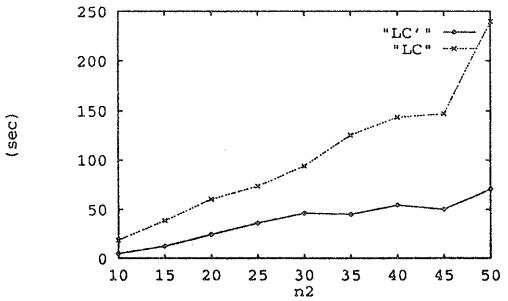

図 5: n_2 の変移における実行時間の推移 (k=5)

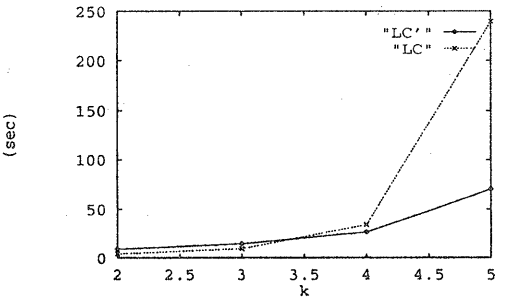

図 6: k の変移における実行時間の推移 (n_2=50)

と変更することを意味する。このことは、手続き LC' が局所無矛盾を達成するものであり、ラベル内の各環境の無矛盾性については考慮していないことから、元の手続きと同等の処理が行なわれることが保証される。(ただしそれにともなって、手続き NOGOOD を多少変更する必要がある。)

比較のための制約ネットワークは、5 つのパラメータを用いてランダムに生成されたものである: 変数の個数 n_1, 全制約数 n_2, 各変数に対する値の個数 n_3, 制約の最大項数 k, 制約が与えられた値組を許可する確率 p。n_2 と k を除く他の 3 つのパラメータは、それぞれ、n_1=25, n_3=4, p=0.6 に固定し、k を 2,3,4,5 の 4 通り、そして n_2 を [10,50] 間を 5 きざみ 9 通りで変動させ、(k, n_2) それぞれについて各 10 個 (計 360 個) のテストネットワークを生成し、LC と LC' における実行時間を測定した。測定は SUN SP/SLC 上の KCL で行なった。そして、グラフは 10 回の測定の平均値をもとにプロットしている (どちらのグラフも縦軸が実行時間で単位は sec。横軸は図 5は n_2 で図 6は k である)。

図 5のグラフにおいて、ネットワークが密になるほど LC' が LC よりも良い結果を出しているのを見てとれる。また図 6のグラフより、制約ネットワーク内の制約の最大項数が大きくなるほど良い結果を得ている。このことから、LC' が複雑な制約ネットワークにおいて有効であることがいえる。

6 まとめ

本稿では、制約ネットワーク内の値間に存在する依存関係を明示化し管理することにより局所伝播とデータ間のつながりを管理する方法を提案した。また実験により、本手法が複雑な制約ネットワークにおいて有効であることを示した。

参考文献

[1] de Kleer,J., "An Assumption-based TMS", Artificial Intelligence, vol.28,(1986)

[2] de Kleer,J., "Extending the ATMS", Artificial Intelligence, vol.28,(1986)

[3] H.W.Guessgen, "CONSAT: A System for Constraint Satisfaction", Research Notes in ArtificialIntelligence. (1989)

[4] K.Nagao, "JIGSAW PUZZLE Matching by Assumption based TMS", Proc. of the Pacific Rim International Conference on Artificial Intelligence'90 (1990)