並列制約充足アルゴリズムとそのKL1による実現

4K－7

杉本 勉
NTTデータ通信(株)

生駒憲治
(財)新世代コンピュータ技術開発機構

1 はじめに

設計問題や画像理解などの分野では「制約」に基づいて問題解決を行う研究がなされてきている[1, 2]。このような問題では、「制約」とはある要素間の関係を記述したものであり、「制約充足」とはこの要素間の関係を満たすような解を見つける問題解決である。

本稿では、変数の組とその変数間で考えられる関係をすべて列挙したもの（解候補）の対を「制約」と考える。この場合、解領域は有限で離散的であるので「制約充足」は組み合わせ問題を解くことに等しくなる。組み合わせ問題は一般に効率面で問題がある。

そこで本稿では、効率改善のための第一歩として並列制約充足アルゴリズムを提案し、それを並列言語KL1で実現する方法を述べる。

2 併合法による制約充足

制約集合の等価表現に制約ネットワークがある。これは一つの制約をノードに、各制約が持っている変数の共有関係をエッジとして考えたものである（図1）。図1では3つの制約$V1, V2, V3$のネットワーク表現を表している。

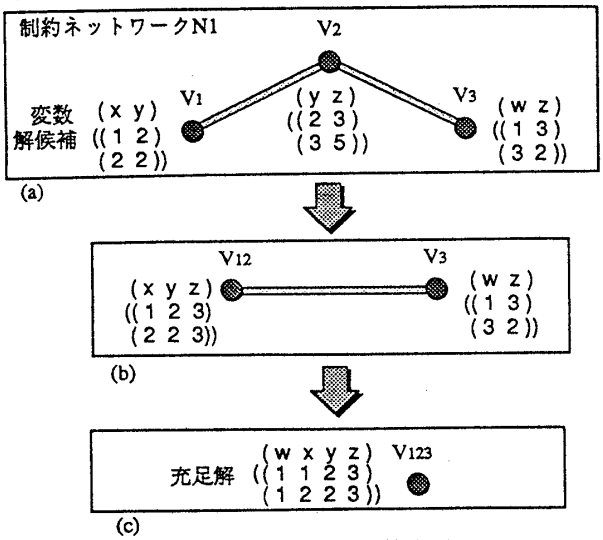

図1: 併合法による制約充足

本研究で対象とする制約充足法は「併合法」と呼ばれているものである[3]。これは制約ネットワークのノードの併合操作によって充足を行う。図1(a)の併合操作では、まず制約ネットワーク$N1$におけるノード$V1$と$V2$の併合を行う。この併合操作において矛盾する解候補を削除する。$N1$では$V1$と$V2$は変数Yを共有している。$V2$の変数Yの解候補である「3」は$V1$の変数Yの解候補にないため矛盾する。この矛盾する解候補が削除され、残りの解候補が$V1$と併合して中間解$V12$が生成される（図1(b)）。これを繰り返し、1つのノードに縮退したときにすべての制約を充足した解($V123$)を得る（図1(c)）。

併合法で制約充足を行う場合、必ずしもノードを併合する順序（併合系列）を考える必要はない。図1において、3つの制約をどのような併合系列で併合させても、すべてのノード（制約）が併合されるならば得られる解は同じである。このことは制約ネットワーク内で中間解の数をなるべく抑えながら、局所的にそして並列にノードの併合を行えばよいことを示す。このことから並列制約充足アルゴリズムが期待できる。

3 並列制約充足とKL1での実現

並列制約充足アルゴリズムを実現するKL1プログラム（一部分）を図2に示す。また、図2のプログラムの変数の内容を表1に示す。ここで提案するアルゴリズムは制約がインクリメンタルに与えられるものとしている。したがって、制約ネットワークを生成しながらノード（制約）の併合が行われる。

ここで提案する並列制約充足では、新しく入力された制約にとって併合するのに最もよい制約は、既に存在している制約の中から選ばれる。「併合するのに最もよい」ことを判断する基準は変数の共有関係の有無とその共有の度合いである。そのために以下のように制約ネットワークを作り、併合を行う。

制約ネットワークの形成は動的に行われ制約ノードは仮配置と削除を繰り返して適当な位置を探す。それを実現するのが図2において1,2,4,5,10,11節である。制約ネットワークに入力される制約はほかの情報と一緒に一つのメッセージとして流れていく（addメッセージと呼ぶ）。このメッセージは4つまたは5つの引数を持つ。4引数の場合は、$ID, Vars, Cands, S$（表1）を、5つの場合はメッセージ通信用変数Msgを加えたものである。この2つの違いは、最初入力されるときは4引数であり、一度でも制約がノードとして仮配置されると5引数となる。4,5節がaddメッセージに対するノードの処理である。ここで、変数間の関係を求め10,11節でそれぞれの関係に対する処理を行う。また、1,2節では仮配置に対する削除と決定を示している。位置が決まるとそのノードは併合のためにメッセージ（mergeメッセージ

Parallel Constraint Satisfaction Algorithm and its Implementation in KL1

Tsutomu SUGIMOTO, NTT Data Communications Systems Corp.
Kenji IKOMA, ICOT

と呼ぶ）を11-2節で求めた併合先へ送る。

図2において、ノードの併合に関わるのは2,6,7,8,9節である。先に述べたように、2節でmergeメッセージを出し、それを相手が受け取ったときに併合が行われる（6節）。このときに注意することは送り側の解候補と受け側のノードの中間解との併合が行われることである。これは、ノードが送り側となるのは一回しかないが受け側となるのは複数回考えられるため、重複して同じ制約が併合されるのを避けるためである。

制約の入力がすべて終わると、conquerメッセージが流れノードが持っている中間解をすべて併合し、充足解を得る。

先に示した方法では制約ネットワークは方向性を持ったツリー型ネットワークとなる。しかし、図2のプログラムでは、制約ネットワークはノード（制約）をKL1のプロセスとする方向性を持ったリング型ネットワークとして実現している。このように実現したのは以下の理由による。

1. 制約は動的に与えられるので、なるべくプログラムの構造を簡単にしたい。すなわち、1つのノードに複数のノードからの併合要求が存在し、それらをすべて共有変数による通信によって管理するのは、プログラムが複雑になる。

2. 新たに入力された制約が併合要求を出す相手の制約を見つけられない場合がある。この場合、動的に入力されるので論理的にはネットワークが分割され、都合が悪い。これをプログラムとして実現するのはリング型の方がよい。

4 まとめ

並列制約充足アルゴリズムと、それを並列言語KL1で実現する方法を述べた。本稿で提案したアルゴリズムは制約がインクリメンタルに与えられることを前提としたものであり、論理的にはツリー型の制約ネットワークをプログラムではリング型で実現している。またこれは、ICOT Symmetry上のPDSSおよびマルチPSI上で負荷分散も考慮して実現されており、プログラムサイズは約1,300行である。

今後はプリミティブな制約表現（たとえば、$=, \neq, >, <$など）や解領域を積極的に利用した方法を考えて行く予定である。

最後に、本研究を進めるにあたり貴重な意見を頂いた元ICOT研究員（現 東芝）永井氏に感謝致します。

参考文献

[1] 淵監修, 制約論理プログラミング, 知識情報処理シリーズ別巻2, 共立出版,(1989).

[2] Hentenryck,P.Van."Constraint Satisfaction in Logic Programming", The MIT Press,(1989).

[3] 西原, 松尾, 池田,"概整合ラベリング問題における併合法の最適化", 人工知能学会誌, Vol.3, No.2, (1988).

```
1) node(In,Out, _,Flg,_,_,_) :- Flg=remove | In=Out.
2) node(In,Out,ID,Flg,Vars,Cands,To) :- Flg=decide |
        Out=[merge(To,ID,Cands)|Out1],
        node(In,Out1,ID,merged,Vars,Cands,{[],[]}).

   alternatively.

3) node([],Out,_,_,_,_,_) :- true | Out=[].

4) node([add(ID,Vars,Cands,S)|In],     Out,ID,Flg,Vars1,Cands1,ImAns)
        :- true |
        relation_analysis(Vars,Vars1,Rel),
        gen_network(Rel,add(ID,Vars,Cands,S),
                        In,Out,ID,Flg,Vars1,Cands1,ImAns).
5) node([add(ID,Vars,Cands,S,Msg)|In],Out,ID,Flg,Vars1,Cands1,ImAns)
        :- true |
        relation_analysis(Vars,Vars1,Rel),
        gen_network(Rel,add(ID,Vars,Cands,S,Msg),
                        In,Out,ID,Flg,Vars1,Cands1,ImAns).

6) node([merge(To,ID1,Cands1)|In],Out,ID,Flg,Vars,Cands,ImAns)
        :- To=ID |
        merge_constr(ImAns,Cands1,NewImAns),
        node(In,Out, ID,Flg,Vars,Cands,NewImAns).
7) node([merge(To,ID1,Cands1)|In],Out,ID,Flg,Vars,Cands,ImAns)
        :- diff(To,ID) |
        Out=[merge(To,ID1,Cands1)|Out1],
        node(In,Out1,ID,Flg,Vars,Cands,ImAns).

8) node([conquer|In],          Out,ID,Flg,Vars,Cands,ImAns)
        :- wait(Flg) |
        ( Flg=merged ->
                Out=[conquer(ImAns)|Out1],
                node(In,Out1,ID,merged,Vars,Cands,ImAns) ;
          Flg=alone  ->
                merge_constr(ImAns,Cands,NewImAns),
                Out=[conquer(NewImAns)|Out1],
                node(In,Out1,ID,merged,Vars,Cands,NewImAns) ;
          otherwise ;
          true       ->
                node(In,Out,ID,Flg,Vars,Cands,ImAns) ).
9) node([conquer(ImAns1)|In],Out,ID,Flg,Vars,Cands,ImAns)
        :- wait(Flg) |
        ( Flg=merged ->
                merge_constr(ImAns1,ImAns,NewImAns),
                Out=[conquer(NewImAns)|Out1],
                node(In,Out1,ID,merged,Vars,Cands,NewImAns) ;
          Flg=alone  ->
                merge_constr(ImAns1,ImAns,Cands,NewImAns),
                Out=[conquer(NewImAns)|Out1],
                node(In,Out1,ID,merged,Vars,Cands,NewImAns) ;
          otherwise ;
          true       ->
                node(In,Out,ID,Flg,Vars,Cands,ImAns) ).


10) gen_network(none,X,In,Out,ID,Flg,Vars,Cands,ImAns) :- true |
        Out=[X|Out1],
        node(In,Out,ID,Flg,Vars,Cands).
11) gen_network(intersect(Vars1),
                AddMsg,In,Out,ID,Flg,Vars,Cands,ImAns) :- true |
        eval_function(Vars1,S1),
        vector_element(AddMsg,4,S,AddMsg1),
11-1)   ( S >= S1 ->
                Out=[AddMsg1|Out1],
                node(In,Out1,ID,Flg,Vars,Cands,ImAns) ;
11-2)     S <  S1 ->
                reply_msg(AddMsg,ID1,Msg1,Cands1,AddMsg1),
                Out=[AddMsg1|Out1],
                v_node(In, Ch, ID1,Msg1,Vars1,Cands1,ID),
                v_node(Ch,Out1,ID, Flg, Vars, Cands, ImAns) ).
```

図2: 並列制約充足のためのKL1プログラム

表1: 変数の説明

変数	内容
In, Out	ノードの入出力チャネル
ID	ノードの識別子
Flg	ノードの状態フラグ (remove/decide/merged/alone)
$Vars$	変数の組
$Cands$	解候補
$ImAns$	ノードが持つ中間解
Rel	ノード間の関係
S	制約ネットワーク内の位置を決めるための評価得点（初期値は0）
Msg	ノード間の通信用変数 (none/intersect)