4J－4

制約充足問題の木探索法による高速化手法の検討

野中　哲
NTTデータ通信（株）

1. はじめに

木探索法を用いた制約充足問題の高速化手法の1つとして、不要な探索枝をあらかじめ刈り込むフォワード・チェッキング（forward checking 以下、FCと略記する）による手法がある[1]。

FCは、論理型言語を用いた制約充足問題解決のための推論機構としてCHIP[2]やBeta-Prolog[3]等において採用されており、木探索法を用いて制約充足問題を解く際の、効率の良いアルゴリズムの1つであることが知られている。

本稿では、FCにおける制約式の評価をあらかじめ行い、変数の値域に関する評価結果を蓄積しておくことにより、同一制約の重複評価の回避と実行時の速度向上を可能とするFFC（full forward checking）法について提案し、簡単な例題(N-Queens問題)への適用結果について述べる。

2. 制約充足問題の定義とFCにおける制約の評価回数

制約充足問題は以下のように定義される。

変数の集合 $\{x_1,\ldots,x_n\}$ と、それぞれの変数に対する値域 $\{D_1,\ldots,D_n\}$ の集合および制約 $\{c_1,\ldots,c_m\}$ の集合が与えられた時に、すべての変数に対して、すべての制約を満たすような値（ex. $(a_1,\ldots,a_n)\in D_T=D_1\times\ldots\times D_n$）を求める問題である。

制約処理系においては、上記の制約 (c_k) が制約式（例えば不等号制約＜や等式の否定≠などをもちいて、$x_1<x_2+x_3$, $x_1\neq x_2$ のように表されている場合）で記述された制約を解くタイミングが重要となる。

前述の論理型制約言語等においては、制約式の中の値が確定していない変数が残り1つとなった時点で、残りの変数について値域を求める方法が用いられている。

このFCにおける値域を求める操作は、文献[4]における表記方法に従うと、変数 x_i に値が割り当てられた時に、2項制約 (c_k) の残りの変数 x_j の値域および値域の直積を変化させる動的制約(**DC**)により定義される。

$$\mathbf{DC}(a_i,x_j,D_1\times\ldots\times D_n)=D_1\times\ldots\times D'_j\times\ldots\times D_n$$
$$\text{with } x_i\neq x_j,\ a_i\in D_i \text{ and } |D'_j|\leq|D_j|$$

ここでD_jからD'_jを求めるにあたっては、$a_j\in D_j$ なるすべての a_j に対して、(a_i,a_j) が c_k を満たしているかどうかを検査（評価）し、満たしている場合に、$D'_j = D'_j \cap \{a_j\}$ とする（最初は、$D'_j=\varnothing$）必要がある。従って、制約式の中の値が確定していない変数が残り1つとなった時点で、残りの変数について値域を求める場合、**DC** 1回につき、$|D_j|(\geq 1)$ 回の制約 c_k の評価を行うことになる。

FCにおいては、**DC**の操作を繰り返し用いて、問題を解くことになるが、以下の2つの理由により、全く同一の制約評価を行う可能性が考えられる。

（1）**DC**において、バックトラックしたことにより、同一の a_i に対する評価を行う場合。
（2）制約式は同一であるが、制約式で用いられる変数が異なる場合。（例えば、制約式として $x_1\neq x_2$ と $x_1\neq x_3$ があるような場合）

このような同一の制約評価が多い場合や、$|D_j|$の値が大きく、**DC**を行っても$|D_j|$から$|D'_j|$への変化が少ないような場合に、制約評価の回数の増大につながる。

3. FFCについて

前節においてFCの制約評価回数が増加する要因について、同一制約式の重複評価を中心に言及した。

A Speed-up Technique of Forward Checking for Constraint Satisfaction Problem
Satoru Nonaka (nonaka@rd.nttdata.jp)
NTT Data Communications Systems. Co.
66-2 Horikawa-cho, Saiwai-ku, Kawasaki-shi
Kanagawa 210, JAPAN

そこで、このような制約の重複評価を回避する方法として、あらかじめ制約式を変数の取り得る値域を用いて評価し、bit-vector構造[5]で蓄積（D_{j,a_i}）した後、**DC**を行う際に

$D'_j = D_{j,a_i}$ AND D_j .

によりD'_jを求める方法（FFCと呼ぶ）が考えられる。ここで、ANDは、bit-vector間の論理積を表す。FFCアルゴリズムの概略を図1に示す。

```
1:DT = D1×...×Dn
2:for i=1...n
3:     ai∈Di
4:     for j=i+1...n
5:        Dj,ai を求め蓄積する
6:S = ∅
7:for i=1...n
8:     ai ∈ Di
9:     for j=i+1...n
10:       DT = DC(ai,xj,DT)
11:       when |DT| = 0, exit(S = ∅)
12:    S = S ∪ {ai}
13:    DT = Di+1×...×Dn
```

図1：FFCアルゴリズムの概略

FFCによる方法を用いた場合、制約の評価は、図1における2行目から5行目の間で行われる。制約の評価回数は、変数x_1とx_2の間の制約C_kに対しては、$|D_1|\times|D_2|$回必要である。一方、前節の（2）のような制約式が同じで、しかも変数の値域が同一の場合は、$D_{3,a_1} = D_{2,a_1}$のように、一度評価している値域を利用することにより、あらかじめ重複評価を行わないことが可能である。

なお、問題によってはFCでは行わない制約の評価をFFCで行うこともあり、FFCにおける評価回数がFCより多くなる場合もある。

4. FFCのN-Queens問題を用いた実験

図2にN-Queens問題の全解を求める場合に、Nを変化させた時のFCとFFCの制約の評価回数、FFCにおけるbit-vectorのAND演算を行った回数（=**DC**の回数）および実行時間を示す。変数は、$\{x_1,...,x_N\}$で、値域は$D_1=...D_N=\{1,2,...,N\}$である。制約はx_iとx_jの距離の大きさdを$|i-j|$とすると、no_attack_d（x_i,x_j）という形の$_nC_2$個の制約からなる。図2のFFCにおける（ ）内の数は、図1の2行目から5行目の間で行われる制約の評価回数と実行時間を示したものである。ここでは、距離の大きさが同じ場合は、値域が同一であるため、同一制約式の評価は行わないものとして測定してある。実験においては、HP9000/735上のC++言語を用いて実装し、実行時間を測定した。

N		制約の評価回数	bit-vectorのAND回数	実行時間(sec)
8	FC	13,024		0.27
	FFC	(448)	4,380	0.16 (0.00)
10	FC	242,174		5.13
	FFC	(900)	82,194	3.17 (0.00)
12	FC	5,958,644		134.27
	FFC	(1,584)	2,038,962	86.81 (0.11)

図2：N-Queens問題によるFCとFFCの比較

図2より、FCの制約の評価回数に対し、FFCにおいてはbit-vectorのAND演算の回数を約1/3程度にすることができた。一方、実行時間は1/1.6程度にとどまった。これは、制約式が単純であるの対し、bit-vectorをC++のクラスとして汎用的に作成したため、制約の1回の評価に要する時間が、bit-vectorのAND演算1回の時間より短かったためと思われる。

5. まとめ

本稿では、制約充足問題をFCで解く場合の同一制約の重複評価について言及し、この問題を回避するFFC法について述べた。また、比較的制約の評価回数が多く、FFCに適していると思われるN-Queens（N=8,10,12）問題の全解を求める例題において、FCに比べ1.6倍程度の速度向上を得ることができた。

参考文献

[1] 西原清一: 整合ラベリング問題と応用, 情報処理学会誌, Vol. 31, No. 4, pp.500-507 (1990)

[2] P. Van Hentenryck: Constraint Satisfaction in Logic Programming. Logic Programming Series, The MIT Press, Cambridge, MA (1989)

[3] Neng-Fa Zhou and Isao Nagasawa: An Efficient Finite-Domain Constraint Solver in Beta-Prolog, 人工知能学会誌, Vol. 9, No. 2, pp.275-282 (1994)

[4] Bernard Burg: Parallel Forward Checking First part, Technical Report TR-594, ICOT (1990)

[5] 奥乃博: ATMSの高速化技法とその応用, 人工知能学会誌, Vol. 6, No. 1, pp.24-37 (1991)