5H−10

演繹データベースにおける計画型問題の制約緩和解法

堤 富士雄　篠原 靖志

(財)電力中央研究所

1 はじめに

近年, 論理プログラミングに制約を組み込んだ制約論理プログラミングの研究が盛んになされている. 制約とは問題の構成要素間のさまざまな関係を宣言的に記述したもので, 線形 / 非線形方程式, 不等式, リストに関する関係式などがあり, 問題記述力の向上に役立つ. [4][1]

制約論理プログラミング言語としては, すでに PrologIII[5], CLP(R) [10], CHIP [6] [9] などが提案されているが, これらの言語では扱う制約を, 必ず満たさなければならない制約 (要求制約, required constraint) に特定している. しかし多くの応用 (GUI, 決定支援, エンジニアリングDBの検索など) では, 必ずしも満たさなくても良いができるだけ満たしていて欲しい制約 (選択制約, preference constraint) を記述できることが求められる.

制約論理プログラミングの特徴である宣言的な記述力をそこなわないためには, これらの選択制約をもそのまま記述できることが望ましい. そこで制約に強さをつけ, その強さによる順序構造として制約の階層を導入することで, 選択制約を記述可能にした言語として HCLP [3] が提案されている.

しかし HCLP は他の制約論理プログラミング言語と同様, 制約充足のために費やされる計算量が大きいため, 大規模で複雑な問題にそのまま適用することはできない. また大規模な問題を扱うためにはデータベースとの効率的な連係が重要になる.

本論文では, 制約階層を含む論理プログラムで記述された大規模な問題を, 効率的に解くことを目的とした手法 Fich を提案する. 本手法の特徴は次の2点である.

(1) 制約階層を計算に積極的に利用することで効率的に解を求める計算方法

(2) データベースからの論理的な検索を効率的に行なう演繹データベースへの制約階層処理の導入

2 制約階層

提案する手法 Fich では HCLP で定義された制約階層を, 知識表現の柔軟性と解探索ヒューリスティックスの記述力を上げるため拡張する.

HCLP ではルールを次のように定義している.

$$p(t) \ :- \ q_1(t),...,q_m(t), s_1c_1(t),...,s_nc_n(t).$$

ここで $p, q_1,...,q_m$ は述語シンボル, t は項のリスト, $c_1,...c_n$ は制約, s_i は c_i の強さを表す. Fich では次の3種類のルールを用いて制約階層を表現する.

定義 2.1

(1) $p(t) :- q_1(t),...,q_m(t), c_1(t),...,c_k(t), ch_1(t),...,ch_l(t).$

(2) $ch(t) :- s_1c_1(t);...;s_nc_n(t).$

(3) $\#invoke(p) \text{ -> } ch_1,...,ch_k.$

ここで $p, q_1,...,q_m, s_1c_1,...,s_nc_n$ は HCLP に同じ. $c_1,...,c_k$ は要求制約の制約述語シンボル, $ch_1,...,ch_l$ は制約階層を表す述語シンボル, $\#inovke$ は制約階層のグループ化のために導入した特殊な述語である. (1) を標準ルール, (2) を制約階層の定義ルールもしくは制約階層ルール, (3) を起動ルールと呼ぶ. □

起動ルールは制約階層のグループ化と制御を表す. 以下各タイプのルールに関して例を使ってその役割を説明する.

Example 1:

```
1.require-pass(X,Y,L) :- pass(X,Y,L),
          X = 大手町, Y = 京都, #length(L)
2.pass(X,Y) :- road(X,W,T),pass(W,Y), #type(T).
3.pass(X,Y) :- road(X,Y,T), #type(T).
4.#type(T):-(5)T= 高速;(4)T= 国道;(3)T= 県道.
5.#length(L):-(3)L<400;(2)L<500;(1)L<600.
6.#invoke(require-pass) -> #type,#length.
```

この例は, いくつかの要求を満たしながら, 大手町から京都までのパスを計算することを目的としている. 要求は `#type` と `#length` の2つの制約階層によって記述されている. なお説明のために例は引数を省略するなど簡略化している.

制約階層ルール (ルール4,5) のボディの制約の頭についた番号 (例えば T= 高速　の (5)) はその制約の強さを表す. 番号の小さいものほど強い制約を表す.

`#type` は道の種類に対する要求である. ルール4を翻訳すると, なるべく高速をさもなければ国道を選ぶようにという制約を表している.

`#length` はパスの道のりに対する要求である. ルール5を翻訳すると, なるべく400Kより短いパスを, もしだめなら500Kより短いパスを選べという制約を表している.

`#invoke` をヘッドに持つ起動ルール6は, `#type` と `#length` が同じグループに属しており, その2つの制約階層に含まれる制約は `require-pass` ファクトの計算状況に応じて取捨選択されることを示している. `require-pass` をこのルールの起動ファクトと呼ぶ.

制約の強さは, 同じグループに属する制約階層の中で意味を持つ. 例えば, `#type` の T= 県道は強さ4で, `#length` の L<400 は強さ3であるので, 後者が強い制約とみなされる. 違うグループの制約階層間では強さの強弱は意味を持たない.

□

HCLP の記法との比較を制約の強さを中心に説明する.

この例で制約の強さ (番号の少なさ) と, どの制約を優先的に選ぶかという優先度が逆転していることに気づかれると思う. この理由を Example 1 の制約階層を HCLP と同じ形式で整理した次の表を参考に説明する.

強さ	制約
5	T= 高速
4	T= 高速 ∨ 国道
3	T= 高速 ∨ 国道 ∨ 県道, L<400
...	

これでわかるように, 例えば `#type` で実際に満たさねばならない強さ3の制約は, 制約階層ルールに記述された強さ3以上の制約の OR 結合である. このように, Fich の制約階層ルールは意味的には OR 結合になるもの省略した記法になっている. この記法のもとでは, 同じ制約階層ルール内の弱い制約は必ず強い制約に包含される. したがってより弱い制約を優先的に満たそうとすることによって全体の制約充足度を上げることができるのである.

A Constraint Relaxation Method for Planning Problem in Deductive Databases

Fujio TSUTSUMI, Yasushi SHINOHARA

The Central Research Institute of the Electric Power Industry

3 制約緩和を導入した解の計算方法

CHIP では，解の探索空間を減らすために制約を積極的に利用している．これはフォワードチエッキングなどと呼ばれる代表的な探索技法を応用したものである．[1]

我々は制約階層に対してもこの技法を応用し，解の探索空間を減らすことに利用する．そして，それを実現するために Fich では，演繹データベースの代表的な計算方法であるセミナイブ法 [2] [11] にその機能を組み込み拡張する．

拡張した計算手法を SNCH (Semi-Naive method for Constraint Hierachy) と呼ぶ．SNCH の中で使用する概念である制約レベルと起動条件の定義を示す．

定義 3.1 (制約レベル)
次の定義ルールを持つ制約階層に対して制約レベルを定義する．

$$ch(t) :- s_1c_1(t);...;s_nc_n(t).$$

1. 制約レベル s は $s_1,...,s_n$ および 0 のいづれかである．
2. 制約レベル s の制約階層 $ch(t)$ ルールの論理的な意味は次の通りである

$$\forall x ch(t) \leftarrow c_{i1}(t) \vee ... \vee c_{im}(t)$$

x は t 中の自由変数． $s_{ik}(k = 1...m) \in \{s_1,...,s_n\}, s = max(s_{i1},...,s_{im})$．なお $min(s_1,...,s_n) \neq 0$ のルールの場合，制約レベル 0 の論理的意味は恒真とする．(強さ 0 の制約は要求制約を表す) □

定義 3.2 (起動条件)
ある起動ルールの起動条件は，そのルールが示す起動ファクトおよびそのファクトを生成するのに必要なファクトが，セミナイブ法の演算中に差分ファクト (incremental relation[11]) からなくなることである． □

SNCH のアルゴリズムを示す．セミナイブ法は [2] [11] を参照．

SNCH

入力: 定義 2.1 の 3 種類のルールで構成されたプログラム，計算の終了条件 (1)

出力: 終了条件が満たされるまで計算されたファクトの集合

step1. プログラム中の制約階層の制約レベルをすべて最弱とする．

step2. セミナイブ法を用いてプログラムを 1 ステップ評価する．

step3. 終了条件が満たされたなら終了．満たされておらず，いずれかの起動ルールの起動条件が満たされたなら，その起動ルールによってグループ化された制約階層に対し，次のように制約緩和を行なう．

(制約緩和)

3.1 制約緩和の対象となる複数の制約階層の内，その制約レベルが最も弱いものを，その制約階層の中で次に強い制約レベルに替える．

3.2 制約レベルの変化した制約階層をボディに持つルールをセミナイブ法で 1 ステップ評価する．起動ルールの起動条件が満たされなくなったら，すべての制約階層の制約レベルを初期状態に戻して，制約緩和を終了し，step2 に戻る．まだ起動条件が満たされているようであれば，step3.1 に戻りさらに制約緩和を行なう． □

SNCH は何も終了条件 (1) を与えないと，すべての制約階層の制約レベルを 0 にするまで不動点演算を行なうため，「解が少なくとも一つ生成されたら終了」などの終了条件を指定する必要がある．

Example1:(続き)
Example 1 のプログラムを SNCH アルゴリズムは次のように評価する．終了条件は require-pass が少なくとも一つ生成されることであるとする．

まず，T= 高速，L<400 という条件でパスの生成を試みる．差分ファクト中に require-pass，pass が存在しなくなったら，条件を T= 高速 ∨ 国道，L<400 に緩めパスの生成を試みる．pass が一つでも生成されたら，条件を T= 高速，L<400 に戻してパスの生成をさらに続ける．

このようにして，条件を緩めたり厳しくしたりを繰り返しながら終了条件が満たされるまで計算を続ける． □

4 関連する研究

選択制約という考え方は，HCLP 以外にも AI ベースの計画システムの中で多く使われており，問題記述に有効であることが示されている．代表的なシステムである ISIS[7] では制約を様々に分類して記述させ，その種類に適した処理を行なう．

制約階層型探索法 [8] の基本的な考え方は HCLP と同じで，解の探索の課程で少しづつ制約を増やしてゆくものである．この探索法では充足しやすい制約と充足の難しい制約を分類して，階層化することでヒューリスティクスを効果的に利用している．

Fich や HCLP の選択制約を仮説と考えることができれば，優先度を持つ仮説を扱う ATMS で扱える問題になる．しかし選択制約は矛盾するために取捨選択されるのではない．例えば Example1 ではパスの一部を高速道路，一部を県道として解を構成することがある．これは県道を解の部分に含んでいるので，T= 高速という仮説は棄却されたことになるが，この制約は選択制約として役割を果たしている．つまり選択制約は単純に優先度付きの仮説と考えることはできない．ただし ATMS の計算方法の工夫には Fich や HCLP と類似した部分が多くある．

5 今後の研究

本論文では演繹データベースに制約階層を組み込み，制約階層を積極的に計算に利用することにより効率的に解を計算する手法 Fich を提案した．

現在，本手法を導入した DDB システムを構築中である．今後は形式的意味論を明確にし，連接水系の発電計画に本手法を適用することを計画している．

参考文献

[1] 相場亮, 古川康一: 制約プログラミングについて – 制約ロジックプログラミングを中心として –, 人工知能学会誌, Vol.6, No.1, (1991), pp 47–59.

[2] Bancilhon,F.:*Naive Evaluation of Recursively Defined Relations, On Knowledge Base Management Systems - Integrating Database and AI Systems*, Brodie and Mylopoulus, Eds., Springer - Verlag.

[3] Borning, A., et al. : Constraint Hierarchies and Logic Programming, *Proc. of ICLP'89*,(1989), pp. 149–164.

[4] Cohen, J.:Constraint Logic Programming Languages,*Comm. ACM*, Vol.33, No. 7, (1990), pp 52–68.

[5] Colmerauer,A.:An Intrduction of PrologIII,*Comm. ACM*, Vol.33, No.7,(1990), pp 69-90.

[6] Dincbas, M., et al.:The Constraint Logic Programming Language CHIP, *Proc. of FGCS'88*,(1988), pp. 693–702.

[7] Fox, M.S.:*Constraint-Directed Search:A Case Study of Job-Shop Scheduling*, Morgan Kaufmann pub., (1987).

[8] 伊藤 謙治 他: 制約階層型探索法を用いた配船スケジューリングシステム, 人工知能学会誌, Vol.6, No.1, (1991), pp 60–71.

[9] Le Provost, T. and Wallace, M.:Domain Independent Propagation, *Proc. of FGCS'92*,(1992), pp. 1004–1011.

[10] Ostroff, J.S. :Constraint Logic Programming for Reasoning About Discrete Event Processes, *Journal of Logic Programming*, Vol. 11 No. 3&4 (1991),pp. 243–270.

[11] Ullman,J.D. :*Principles of database and knowledge-base systems vol. 1 vol. 2*, Computer science press, (1989).