ICOT Technical Memorandom: TM-1201

TM-1201

スケジューリング問題向き言語 GCL と その応用 －組合せ制約と非線形目標制約の並列制約 論理型言語への導入－

上田　晴康（富士通）

August, 1992

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

スケジューリング問題向き言語 GCL とその応用
— 組合せ制約と非線形目標制約の並列制約論理型言語への導入 —

GCL: A Parallel Constraint Logic Programming Language fit for Scheduling Problem
— Adoption of assignment constraint and non-linear goal constraint —

上田晴康*
Haruyasu Ueda
(株) 富士通研究所 国際情報社会科学研究所
International Institute for Advanced Study of Social Information Science,
FUJITSU LABORATORIES LTD.

This paper introduces a parallel constraint logic programming language **GCL**. It is designed to fit to describe and solve the planning problem including the scheduling problem. **GCL** has three extensions from ordinary Prolog; 1. Andorra-like parallel execution, 2. New constraint named "assignment constraint" to describe permutation relation between a set of variables and a set of constants, 3. New pragma to denote the goal function expressed by equations. The assignment constraint is defined because the value assignment to variables often makes the combinational explosion and the explicit definition of assignment helps optimization. **GCL** has built-in control mechanisms for this optimization. They use the goal function denoted by pragma. In the current state, they are Branch&Bound and the Monte-Carlo method.

In this paper, first, these extensions are shown in more detail, then a small example to demonstrate the descriptiveness of **GCL**; a scheduling for short tour. Last, it is shown that the comparison with existing languages, such as CHIP and HCLP.

1 はじめに

近年、制約論理プログラミングは、計算機を用いた問題解決における有効な手段として位置付けられるようになった。しかし、論理プログラミングの持つ宣言的性質がその問題解決に適しているとはいえ、計画型の問題を論理型言語で記述して解を得るのは容易なことではない。その主な原因としては、論理型言語は解を探索するための制御方法として単なる後戻りしか用いないこと、探索に必要な目的関数を記述できないこと、数式を宣言的に扱えないことが挙げられる。よって本研究では、既存の制約論理型言語における言語仕様について、計画型問題の記述とその解決が容易になるように拡張した並列制約論理型言語 **GCL** (Generalized Constraint Language) を提案し、さらにその応用例を示すことにより、**GCL** がスケジューリング問題に対する十分な記述力を有していることを示すこととする。

*連絡先: 上田晴康 〒144 大田区新蒲田 1-17-25
E-mail ueda@iias.flab.fujitsu.co.jp

2 GCL の概要

GCL は、スケジューリング問題を含む計画型問題について適用する目的で、記述能力および解の探索方法に着目して拡張を行なった言語である。計画型問題の記述能力に関しては、従来の線形等式 / 不等式制約、有限体領域上の制約に加え、計画型問題で探索の主要部分となる組合せ生成を直接記述できる割り当て制約を扱

えるように拡張を行なった。解の探索方法に関しては、非決定的な節の選択順序の制御の問題とみなし、プラグマを非線形制約式で記述する拡張を行なった。また、並列探索をより効率的にするために、AKL[1] と同様に or 並列実行を lazy に行なう拡張も行なった。

制約処理に関しては、EPOCH[2] と同様に、必ず満たさなくてはならない『必須制約』とできるだけ満たせば良い『目標制約』の両方を扱うことができる。従来の制約と割り当て制約は必須制約として、プラグマに記述される目的関数は目標制約として扱われる。制約処理系は必須制約を満たす範囲で Branch&Bound を行ない、目標制約を最大化するゴールを推測してその導出を行なう。

2.1 必須制約

GCL では、必須制約として線形等式 / 不等式制約、有限体領域上の制約の他に、

$$assignment([変数\ldots],[定数\ldots])$$

で表される割り当て制約を扱うことができる。割り当て制約とは、それぞれの変数に対して定数を重複しないように適当に選び出せば、同一化できるという制約である。GCL は割り当て制約を解決するために、変数列と定数列をそれぞれ半分の長さの二つの部分に分け、再帰的に割り当て制約を解くという処理を行なう [3]。

2.2 目標制約と並列実行制御

GCL の目標制約は探索を制御するための優先度の拡張として与えられるものであり、これは数理計画法で目的関数を与えることに相当する。目標制約は maxmize プラグマを用いて以下のように表現される。

$$pred(\ldots)@maxmize(Exp)$$

これにより、述語 pred(...) の実行中に非決定的実行が必要になった場合、式 Exp を最大化する候補を推測して実行を行なう。目標制約を最大化する候補を推測するためには、モンテカルロ法を用いている。Exp には、数値として値の得られる任意の数式を記述することができる。Exp 中の変数は、実行中にさらに Var is Exp' という形の制約を加えて、目的関数を動的に詳細化することもできる。これにより、不定長の値を生成するための目的関数を容易に定義することが可能である。

3 応用例

ここでは、以下の制約条件のもとで、満足度が最大となるスケジュールを求めるという問題について考えることとする。

- スケジュールは 10 時以降に始まり、17 時までに終了しなくてはならない。
- 行き先は遊園地、映画館、動物園の 3ヶ所から選び、3ヶ所全部回っても良いし、どこにも行かなくてもよい。ただし重複して行ってはならない。
- 移動時間は考えない。
- 遊園地に関しては、遊ぶ時間は 4 時間以上 6 時間以内、一回に必要な費用は 5000 円、満足度は 0.5+ 遊んだ時間の比例分とする。
- 映画館に関しては、決められたスケジュールで 1 日 4 回放映され、料金は 1500 円、満足度は 0.35 とする。
- 動物園に関しては、遊ぶ時間は 1 時間以上 3 時間以内、料金は 2000 円、満足度は 0.3+ 遊んだ時間の比例分とする。
- 全体の満足度は便宜的に、

  $$\alpha \sum \text{行き先の満足度} - \beta f(\text{総費用}) + 0.5$$

  とする。ただし α、β は満足度に関する重要性の係数である。また $f()$ は費用を満足度に変換する関数で、以下の式で表されるものとする。

  $$f(X) = \begin{cases} \frac{6X}{10000} & X \leq 6000 \\ \frac{X^2}{10000000} & X > 6000 \end{cases}$$

この問題に関して、出来上がるスケジュールは行き先、行き先ごとの開始時刻、終了時刻を示し、プログラム中において各行き先は int(行き

```
plan_generation(P,Cost_Sat,Satis) :-
  plan(P),plan_constraint(P,Cost,Satis),
  cost_satisfaction(Cost,Cost_Sat).
plan(P) :- prefix(P,P1),
  assignment(P1,[int(amsmnt_prk,_,_),
        int(theater,_,_),int(zoo,_,_)]),
  (P=[int(_,Beg,_)|_],10.0 >= Beg,
    last(P,int(_,_,End)),End =< 17.0;
   P=[]).
plan_constraint([],0,0.0).
plan_constraint([H],Cost,Sat) :-
  plan_constraint1(H,Cost,Sat).
plan_constraint([H,H1|T],Cost,Sat) :-
  Cost=CostH+CostT,Sat=SatH+SatT,
  plan_constraint1(H,CostH,SatH),
  H=int(_,_,End),H1=int(_,End,_),
  plan_constraint([H1|T],CostT,SatT).
plan_constraint1(H,Fee,Sat) :- fee(H,Fee),
  time_necessity(H),satisfaction(H,Sat).
cost_satisfaction(Cost,S) :- Cost=<6000,
  S is 6*Cost/100000.
cost_satisfaction(Cost,S) :- Cost>6000,
  S is Cost*Cost/100000000.
fee(int(amsmnt_prk,_,_),5000).
fee(int(theater,_,_),1500).
fee(int(zoo,_,_),2000).
satisfaction(int(amsmnt_prk,Beg,End),
               0.5+(End-Beg)/10).
satisfaction(int(zoo,Beg,End),
               0.3+(End-Beg)/10).
satisfaction(int(theater,_,_),0.35).
time_necessity(int(amsmnt_prk,Beg,End)):-
  4.0 =< End-Beg,End-Beg =< 6.0.
time_necessity(int(zoo,Beg,End)) :-
  1.0 =< End-Beg,End-Beg =< 3.0.
time_necessity(int(theater,10.0,11.5)).
time_necessity(int(theater,12.0,13.5)).
time_necessity(int(theater,14.0,15.5)).
time_necessity(int(theater,16.0,17.5)).
```

図 1: スケジューリング問題のプログラムリスト

先, 開始時刻, 終了時刻) で表現される。また各行き先は開始時刻に従ってリストに並んでいる。

この場合の GCL のプログラムは図1のようになる。ここでは、スケジュール作成のための plan/1 述語の中で、GCL の特徴の一つである割り当て制約を用いている。このプログラムに対して、以下のゴール列を処理系に与えることにより、制約を満たす 12 通りの解が求まる。

```
:- plan_generation(Sc, Cost, Satisfaction)
  @maxmize(0.5+ α *Satisfaction- β *Cost).
```

ここでは目的関数はプログラム実行中に Cost や Satisification という変数に対する制約として、動的に構成される。解の求まる順序であるが、

$\alpha = 0.4, \beta = 0.1$	
映画→遊園地	1.018
映画→遊園地→動物園	1.008
映画→動物園→遊園地	1.008
遊園地→映画	0.958
$\alpha = 0.2, \beta = 0.3$	
映画→遊園地	0.6532
遊園地→動物園	0.653
動物園→遊園地	0.653
遊園地	0.630

表 1: 生成される解; 目的関数値の順

これはおおむね目的関数の値の大きいものから順に生成される。目的関数の値は、α、β の値によって大きく変化する (表1)。このような場合に、制約はプログラム中にではなくゴールにつけるプラグマとして記述できるので、その記述力は高くなるといえる。

4 関連研究との比較

本研究では、GCL と類似の制約論理型の言語仕様を持つ CHIP[4]、HCLP[5]、EPOCH[2]、PARCS[6] を取り上げ、比較を行なった。

4.1 必須制約解消系

必須制約の解消に関して、計画型問題でしばしば問題となる有限体領域上の変数を CHIP が扱える他は、いずれの論理型言語も線形方程式または線形不等式 (あるいはその両方) を扱えるだけである。しかし計画型問題で最も計算量が必要なのは組合せ生成であり、これを効率的に処理できる言語は今のところ存在していない。このため、組合せ制約を扱う際にはユーザが明示的にプログラムするか、処理系が非効率的な数え上げを行なうことになる。この点に関して GCL では二分探索と Branch&Bound を行なうため、より効率的に解生成ができる。

4.2 目標制約

HCLP や EPOCH は制約の望ましさを記述できる点で GCL と類似している。しかし HCLP

の制約階層においては、制約の優先度には上下関係だけが意味を持ち、少しでも優先度の高い制約をより多く満たそうとするため、優先度がわずかに異なっている場合のトレードオフの解決をすることは不可能である。またEPOCHの目標制約は、目標制約の値を最大化するという点で酷似しているが、線形の制約のみを扱っている点がGCLとは異なる。従って、非線形の制約も扱うことが可能なGCLはより柔軟な記述性を実現しているといえる。また、EPOCHは導出がすべて終了した後に目標制約を最大化する変数値を求めるため、非決定的な解の間での最大化を行なうためには、いったん全解探索をしてからソートをしなおさなくてはならない。これに対しGCLは、目標関数の値を動的に推定しながら探索制御を行なうため、すべての非決定的な解の間での準最大の解を求めることができる。

4.3 並列探索制御

PARCSは並列探索の制御をプログラムの内容に応じて行なっている点でGCLと類似している。PARCSでは、目標制約は明示的に与えることはなく、制約の種類、導出の回数、残っている変数の数、および含まれるand-Node の数を元に実行制御を行なう。この方式は、[7]における制約を用いたヒューリスティクサーチを並列に応用したものと考えられる。しかし目標関数を用いない探索制御の場合、探索の失敗は回避できるものの、十分な制御がなされない。なぜなら、多くの成功する探索枝の中から最適解を求めるような問題の場合、その計算量については改善されないからである。

5 まとめ

本研究ではスケジューリング問題に関する記述力および並列実行制御の効率化を目的として、並列制約論理型言語の拡張を行なった。本研究で用いたGCLでは通常の制約論理型言語の線形不等式、線形等式に加えて、組合せ制約も扱うことができる。さらにGCLでは、これらの必須制約に加え非線形の目標制約を扱うこともできる。

これらのことから、本研究はスケジューリング問題に関する記述力および並列実行制御の効率化という点で、既存の言語では不十分であった解の探索を効率良く制御するための方法を提案できたといえる。

以上の点をふまえた上で、現在汎用計算機環境上での GCL 処理系の実現とその処理系における並列探索制御アルゴリズムの有効性の検証を行なっている。

謝辞 本研究の一部は第五世代コンピュータプロジェクトの一環として行なわれました。日頃より御指導を賜りました当研究所の田中二郎第2研究室長に心より感謝致します。本研究を進めるにあたりさまざまな助言を賜りました國藤進教授にも、心よりお礼申し上げます。

参考文献

[1] Sverker Janson et al. Programming paradigms of the Andorra Kernel Language. In *Proc. of ILPS91*, 1991.

[2] 西沢剛他. 目標制約の表現機構を備えた制約論理型言語EPOCH. 人工知能学会誌, Vol. 7, pp. 487–495, 1992.

[3] 上田晴康他. 計画問題向き並列論理型言語GCL. 信学技報, COMP91-64, vol. 91, No. 343, July 1991.

[4] M. Dincbas, et al. The constraint logic programming language CHIP. In *Proc. of FGCS88*, pp. 693–702, 1988.

[5] A. Borning, et al. Constraint hierarchies and logic programming. In *Proc. of ICLP89*, pp. 149–164, 1989.

[6] 小林直樹他. 並列制約論理プログラミングとその制御. In *LPC91*, pp. 191–194, 1991.

[7] Mark S. Fox, et al. Constrained heuristic search. In *IJCAI 89*, pp. 309–315, 1990.