人 工 知 能 89−18
（1993. 8. 19）

ネゴシエーションによる制約緩和の並列実装

和住 誠一郎　津田 宏　相場 亮
(財) 新世代コンピュータ技術開発機構

1 はじめに

分散環境において、個々の問題解決器が部分的な情報しか持たない状況で、全体で意味のある答を求めるためには、ネゴシエーションが必要となる。我々は協調的な問題解決の研究を進めるにあたり、同種の問題解決器によるネゴシエーションのためのプロトコルの検討と、実験環境の実装を行なっている。検討の対象には制約緩和問題を取り上げ、実験環境としてKL1を用いて並列推論マシンPIM上にN super-Queen問題を実装している。

本稿では、プロトコル、問題解決器の動作、実験環境の実装について述べる。

2 分散制約緩和

変数V、領域D、変数間の制約Cにおいて、制約Cが過制約である場合、制約の部分集合$C'(\subset C)$のうち、それを満たすVへの値代入が存在し、かつ最適なものを求めることを制約緩和問題と呼ぶ。

変数や制約が複数の問題解決器に分割されている場合、各々の問題解決器が制約の緩和を分散して行ない、全体として最適な答を求めることを分散制約緩和と呼ぶ。

3 問題解決器

問題解決器は、変数と、その変数が関連する全ての制約を持つ。制約には数値である重要度を付ける。また、問題解決器は数値による閾値を持つものとする。ここでは、閾値以上の重要度を持つ全ての制約を満たす解を求めることを目的とする。

[プロトコル]
問題解決器は以下に示す3種類のメッセージにより情報を交換する[1]。

- $plan(From, To, PlanSet, Th)$
- $accept(From, To, PlanSet, Th)$
- $reject(From, To, PlanSet, Th)$

問題解決器$From$は自分の閾値Thと、Thにおけるプラン集合$PlanSet$を$plan(From, To, PlanSet, Th)$で問題解決器$To$に提示する。$To$は、$PlanSet$から$Th$以上の重要度を持つ制約を満たすプラン集合$PlanSet'$を生成し、$accept(To, From, PlanSet', Th)$で$From$に知らせる。$PlanSet'$を生成できなかった場合は、$reject(To, From, PlanSet, Th)$を返信する。

問題解決器は、プラン集合を提示したすべての問題解決器から$accept$が返ってきた場合、$accept$で受け取ったプラン集合から解を生成する。$reject$が返ってきた場合は、閾値を緩め、新たにプラン集合を生成し、再提示する。

問題解決器はプランに対する3種類の処理を持つ。

$merge(PlanSet1, PlanSet2, Th, NewPlanSet)$: $PlanSet1$と$PlanSet2$を結合し、重要度がTh以上の制約を満たす$NewPlanSet$を生成する。

$make(PlanSet, Th, NewPlanSet)$: $PlanSet$に対して自分の変数への値代入を付け加え、重要度がTh以上の制約を満たす$NewPlanSet$を生成する。

$select(PlanSet, Th, NewPlanSet)$: $PlanSet$からTh以上の重要度を持つ全ての制約を満たすプランを選択し、$NewPlanSet$に格納する。但し$PlanSet$が自分の変数を含まない場合は、$make(PlanSet, Th, NewPlanSet)$を行なう。

問題解決器は、受信したメッセージと内部状態から動作を決定する。問題解決器を簡単にするため、生成したプランや通信に関する履歴は取らないものとする。

[問題解決器の内部状態]

$MyPlanSet$: プラン集合
$PsID$: $MyPlanSet$の識別子
$MyTh$: 閾値
$Neighbors$: メッセージを送信できる問題解決器の集合
MyV: 自分が値を代入できる変数
Rt: 送信した$plan$メッセージの数

[問題解決器の動作] 問題解決器$From$からメッセージを受けとった問題解決器Toの動作を示す。

- $plan(From, To, PlanSet, Th)$を受信した場合

```
select(PlanSet, Th, NewPlan);
if (NewPlanSet ≠ φ) accept(To, From, NewPlanSet);
else                reject(To, From, PlanSet);
if ((Th ≤ MyTh) & (NewPlanSet ≠ φ)){
  if (MyV ∉ PlanSet){
    plan (To, Nb, NewPlanSet, Th) Nb ∈ Neighbors \ {To};
    Rt ← changeRt(Rt, MyTh, Th, |Neighbors \ {To}|);
  }
  MyPlanSet ← NewPlanSet;
  MyTh ← Th;
}else if ((Th ≤ MyTh) & (NewPlanSet = φ)){
  make(PlanSet, Th, NewPlanSet2);
  if (NewPlanSet2 ≠ φ){
    plan (To, Nb, NewPlanSet2, Th) Nb ∈ Neighbors;
```

$MyPlanSet \leftarrow NewPlanSet2$;
$MyTh \leftarrow Th$;
$Rt \leftarrow changeRt(Rt, MyTh, Th, |Neighbors|)$;
}
}else if$(Th > MyTh) plan(To, From, MyPlan, MyTh)$;

$changeRt\ (Rt, MyTh, Th, N)$
if $(MyTh = Th)$ $return(Rt + N)$;
else $return(N)$;

- $accept(From, To, PlanSet, Th)$ を受信した場合

if $(Th = MyTh)$\{
$merge(MyPlanSet, PlanSet, Th, NewPlanSet)$;
if $(NewPlanSet \neq \phi)$\{
$MyPlanSet \leftarrow NewPlanSet$;
$Rt \leftarrow Rt - 1$;
} else {
$NewTh \leftarrow increase(Th)$;
$make(MyPlanSet, NewTh, NewPlanSet)$;
$plan\ (To, Nb, NewPlanSet, NewTh)\ Nb \in Neighbors$;
$MyPlanSet \leftarrow NewPlanSet$;
$MyTh \leftarrow NewTh$;
$Rt \leftarrow |Neighbors|$;}}

- $reject(From, To, PlanSet, Th)$ を受信した場合

if $(Th = MyTh)$\{
$NewTh \leftarrow increase(Th)$;
$make(MyPlanSet, NewTh, NewPlanSet)$;
$plan(To, From, NewPlanSet, NewTh)$;
$MyPlanSet \leftarrow NewPlanSet$;
$MyTh \leftarrow NewTh$;
$Rt \leftarrow 1$;
} else {
$plan\ (To, From, MyPlanSet, MyTh)$;
$Rt \leftarrow Rt + 1$;}

Rt が 0 になると、$MyPlanSet$ と $MyTh$ が答となる。

4 N super-Queen 問題

Queen にナイトの利きをくわえたものを super-Queen と呼ぶ。N×N のマス目に N 個の super-Queen をおき、互いに取り合わない配置を求める。それが不可能な場合、取り合う駒同士をできる限り離した配置を求める。これを N super-Queen 問題と呼ぶ。

第 i 列目の駒の位置を変数 x_i とし、駒の間の位置関係を制約で表す。制約には非負の整数である重要度 $k = 0, 1, ..., N^2 (= k_{max})$ を与える。重要度 k における第 i 列目の駒と第 j 列目の駒の間の制約は以下の通りである。$\forall i, j, C_{ij}^k$ を満たすもっとも小さな k を求める。

$$\begin{aligned} \mathbf{C} = \{C_{ij}^k | C_{ij}^k = ((x_i \neq x_j) \wedge (|x_i - x_j| \neq |i - j|) \\ \wedge (|(x_i - x_j)(i - j)| = 2) \\ \vee ((x_i - x_j)^2 + (i - j)^2 > k_{max} - k)\} \end{aligned}$$

5 KL1 による実装

KL1 は以下の特徴を持つ並列論理型言語である [2][3]。

- ボディのゴールは並列プロセスとして実行可能である。
- 共有変数によりプロセス間通信を実現する。
- ゴールに対応する節の選択時のユニフィケーションにより、通信の同期を実現する。

KL1 では、ネゴシエーションを行なう複数の問題解決器は以下のように記述できる。

- 各々の問題解決器を並列に実行するゴールとして記述する。
- 問題解決器は中継プロセスを経由してメッセージを交換する。

全ての問題解決器の状況を把握する監視プロセスを用意する。監視プロセスは問題解決器の起動と終了を行なう。監視プロセスがある問題解決器に $start$ を送ると、それは初期閾値におけるプラン集合を生成する。そのプラン集合を他の問題解決器に提示することで、ネゴシエーションが開始される。全ての問題解決器が解を得ると、監視プロセスは各問題解決器に $stop$ を送り、処理を終了を知らせる。

6 実験

問題解決器の数を変えて 4 super-Queen 問題を PIM/m 上で実行した。以下にその結果を示す。

問題解決器の個数	1	2	3
10 回の平均時間 (msec)	406	486	3817

7 おわりに

ネゴシエーションにより制約緩和問題を解く同種の問題解決器間のプロトコルを定義し、N super Queen 問題を PIM/m 上で KL1 により実装した。

プロトコルを単純にしたため問題解決器の数が増えるに従い、メッセージの数が増加する傾向にある。今後は問題解決器のプロトコルを検討し、より高度なネゴシエーションを実現したい。

参考文献

[1] 津田宏, 和住誠一郎, 相場亮. ネゴシエーションに基づくアブダクション. SWoPP'93 ポジションペーパー,1993.

[2] K.Ueda and T.Chikayama.Design of the Kernel Language for the Parallel Inference Machine.The Computer Journal,1990.

[3] KL1 言語.bit 別冊 第五世代コンピュータの並列処理, 共立出版,pp.61-85,1993.