ソフトウェア工学 105−10
（1995．9．8）

分散オブジェクトによる分散制約充足システムの開発

渡邊裕之　渡辺慎哉　宮本衛市

北海道大学工学部
〒060 北海道札幌市北区北13条西8丁目

分散協調問題解決とは、複数の自律的な計算主体による協調的な問題解決である。本論文では、分散協調問題解決における基本的な問題である分散制約充足問題を解くアルゴリズムを示す。この問題は問題全体の情報をひとつの統括された計算主体に集中することによっても解くとができるが、本論文では問題の情報を集中しない、データ分散型のアルゴリズムを示す。このアルゴリズムにおいては、分散オブジェクトを計算主体とし、分散オブジェクト間のメッセージ通信によるローカルプロパゲートを用いて制約充足を行なう。

Development of Distributed Constraint Solving System by Distributed Object

Hiroyuki Watanabe, Shin-ya Watanabe, and Eiichi Miyamoto
Faculty of Engineering Hokkaido University
Nishi 8, Kita 13, Kita-ku, Sapporo 060, Japan

Distributed cooperating problem solving is a cooperative problem solving by plural autonomous computation subjects. In this paper, we suggest an algorithm that solves distributed constraint solving problem which is one of the basic problems of distributed cooperating problem solving. This problem can also be solved by an algorithm that concentrates all infomation of problem into one controled computation subject, however we suggest in this paper,Data-Distributed-Type algorithm that doesn't concentrate all infomation of problem. In this algorithm, we use distributed object as computation subject, and solve the constraint by using local propagation between distributed objects.

1 はじめに

分散協調問題解決とは、複数の自律的な計算主体よる協調的な問題解決である。

分散協調問題解決における基本的な問題である、分散制約充足問題を解くアルゴリズムを示す。この問題は問題全体の情報をひとつの統括された計算主体に集中することによっても解くことができるが、本論文では問題の情報を集中しないアルゴリズムを示す。

制約プログラミング言語においては、プログラマーは関係を表す問題を解くためにその関係を維持するための手続きを記述する必要はなく、関係を宣言的に記述するだけで、その関係を維持する解を見付けるのは制約充足システムの仕事である。

制約を維持する制約充足システムは、一般にソルバーと呼ばれる。制約ソルバーは、解く問題に応じた様々な種類の特化されたアルゴリズムが研究されている。

なかでも Blue ソルバーはローカルプロパゲートのみを用いる非常に効率の良いアルゴリズムである。

Blue アルゴリズムでは、ソルバーをひとつの計算主体としてそこに問題全ての情報を集中させ、ソルバーによって求めた解を、それを必要とする全ての計算主体に対して放送することで分散制約充足問題を解くことができる。

本研究ではこの Blue ソルバーのアルゴリズムをもとに、分散された情報をひとつの統括された計算主体に集中させずに、ローカルプロパゲートのみを用いて制約充足を行うアルゴリズムを考案する。

制約や、制約される変数の情報は分散オブジェクトに分散されているものとし、分散オブジェクト間のメッセージ通信によるローカルプロパゲートを行い、制約を充足する解に到達させる。

2 本研究の背景

2.1 制約ソルバー

制約を充足する解を求める制約ソルバーについては様々な研究がなされている。制約ソルバーの一般性が増すとそれにつれてソルバーのコストも高くなる。しかし、様々な種類の制約系を効率的に解くには、それぞれの要求に応じた、特化されたアルゴリズムが必要となる。

例えば、ユーザーインターフェース等の応答性を要求するアプリケーションの場合は、一般性よりも速度を必要とし、一方汎用シミュレーションなどの場合は速度よりも、より多くの種類の制約を扱う、一般性が必要となる。

[1] ではそれぞれ別な点において一般性と効率を取り引きした制約ソルバーアルゴリズムの以下のようなスペクトルを考案した。

Red Bertrand[2] で用いられている、Augmented 項書き換えシステムを基にした、閉路を持つ制約グラフや、連立方程式のある制約を解くことができるアルゴリズム。他のどのアルゴリズムよりも一般的であるが、効率は悪い。

Orange 線形計画問題を解くシンプレックスアルゴリズムに基づいた、線形の等式、不等式によって構成される制約系の 1 つまたは全ての解を求めるために特化されたアルゴリズム。

Yellow 古典的な relaxation である、反復 hill-climbing 法を用いたアルゴリズム。Orange よりも効率は悪いが、非線形の等式による制約を用いることができる。

Green 変数が有限なドメインの値をとる場合に限定した、ローカルプロパゲートとジェネレートアンドテスト木探索を結合したアルゴリズム。

Blue ThingLab[1] で用いられているローカルプロパゲートによるアルゴリズム。非常に効率は良いが、閉路を含んだ制約グラフを解くことはできない。

2.2 インクリメンタルな制約ソルバー

ユーザーインターフェース等のアプリケーションの場合、ユーザーによる変更等はしばしば制約データフローグラフに局所的にしか影響を及ぼさない。従って、[1] では変更のたびに制約グラフを始めから解き直さず、直前の解にインクリメンタルに変更を加えることにより、制約データフローグラフの変更を最小限にとどめることにより、効率良く制約を解くアルゴリズムを提唱している。

2.3 分散制約充足アルゴリズム

分散制約充足問題とは、制約充足問題の問題の情報が複数の計算主体に分散された問題である。ソルバーによる制約充足は、分散された情報をある特定の計算主体に集中させて問題を解決することに相当する。

この解法では、ある状態から別の状態に移る際に送られる総メッセージ数は比較的少なくなるが、問題が大きい場合、情報を集める計算主体に大きな負荷がかかる。

これに対し、[3] ではデータを集中しない、データ分散型の解法を提唱している。

このアルゴリズムは変数が有限なドメインから値を取る場合に限定したものであり、変数を持つ計算主体 (エージェント) がそのドメインから値を選び、その値が可能かどうかを制約の知識をもつ別のエージェントに対して問い合わせる。その値を取ることができない場合は、再び変数の取り得るドメインのなかから別の値を選び、制約の知識をもつエージェントに対して問い合わせを行う。

ドメインの中で取り得る値がなくなった場合は依存関係に基づくバックトラックによって、別の変数の値を変更する。

3 本研究の目的

本研究では [1] で提唱された、Blue ソルバーのインクリメンタルなバージョンである DeltaBlue アルゴリズムを、分散された情報をある特定の計算主体に集中させないデータ分散型アルゴリズムに拡張し、変数の取り得る値のドメインを実数領域に広げた分散制約充足アルゴリズムを確立する。

4 分散制約充足問題

4.1 分散制約充足問題とは

制約とは維持されるべき関係を記述したものであり、制約による記述を用いることにより、その関係を維持するための手続きを書く必要はなくなり、宣言的にそれを表すことができる。

また、制約は通常多方向的なものである。すなわち、例えば c=a+b という制約は、a,b,c それぞれの値のうち任意の未定義の値を求めるために用いることができる。

制約充足問題とは、複数の変数に対し、制約によって表される関係を充足する値の割り当てを決定する問題である。

分散制約充足問題とは、制約充足問題の問題の情報が複数の計算主体に分散された問題である。

4.2 モデル化

通信ネットワークによって結合された複数の自律的な計算を行う主体 (オブジェクト) が存在する。各オブジェクトは非同期、独立に動作し、全体を統括するオブジェクトは存在しない。

オブジェクト間の通信については、以下の仮定を置く。

- 各オブジェクト間の通信はメッセージ通信によって行われる (共有メモリは存在しない)。
- 各オブジェクト固有の ID が通信のために用いられる。あるオブジェクトは ID を知っているオブジェクトに対して、メッセージを通信することができる。
- 通信の遅れは有限で、メッセージが失われることはない。
- ひとつのオブジェクトから別のあるオブジェクトへ送られるメッセージの順序は保存される。

4.3 問題の概要

複数の自律的なオブジェクトが並列に動作している。

ある変数を持ち、その値を決定しようとしているオブジェクトを制約変数オブジェクト、変数間の制約の情報を持っていて、その関係を維持する働きをするオブジェクトを制約オブジェクトと呼ぶ。一般的にはひとつのオブジェクト中に複数の制約変数や制約を持つ場合が考えられるが、ここでは各オブジェクトが制約変数、制約にそれぞれ 1 対 1 に対応するものとする。

各制約変数オブジェクトはその制約変数に接続されている全制約の ID を、各制約オブジェクトはその制約に関係する全ての制約変数オブジェクトの ID を知っている。

いくつかの制約変数オブジェクトと制約オブジェクトが存在し、制約を充足する各制約変数の値を決定しようとする。

求める解は、制約を充足する1つの状態だけであり、可能な全ての状態を求める事はしない。

5 解法

5.1 2変数間のイコール制約に限定した解法

分散制約充足問題の最も簡単なモデルとして、制約として2つの変数の値が同じであるという関係のみを用いる場合を考える。

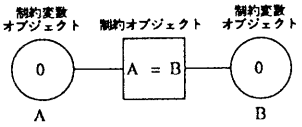

図 1: 2変数1制約のデータフローグラフ

図1は、2変数1制約の場合のデータフローグラフの例である。図中、円で示したのが制約変数オブジェクトであり、正方形で示したのが制約オブジェクトである。制約変数オブジェクトと制約オブジェクトを繋ぐ辺は、制約オブジェクトの持つ制約がその変数を用いることを示し、これはすなわち各オブジェクトがお互いのIDを知っていて、相互にメッセージ通信が可能であることを示している。図中で制約の中に書いた関係式はその制約オブジェクトの持つ制約式であり、制約変数の下に付けられたラベルは制約変数の変数名、円中に書いたのが変数の値である。

ある制約変数の値が変更された時、その変更による影響は、その制約変数オブジェクトに接続されている制約オブジェクトを通して局所的な値の伝達を繰り返すことによって全体に伝えることができる。

制約が2変数間の関係に限定されているので、制約データフローグラフの末端は必ず制約変数オブジェクトになる。末端の制約変数の先には値を伝えるべき制約オブジェクトはないので、そこで伝達は停止する。従って、全ての値の伝達が末端に到達した時点で、全ての制約変数は制約を充足した値を取っていることになる。

5.2 分散オブジェクト間メッセージ通信による問題点

分散オブジェクトはそれぞれ自律的に独立して並列に動作しているため、システム内で同時に複数の制約変数オブジェクトに対して値の変更要求が起こり得る。

一つの変更による値の伝達が全ての末端の変数オブジェクトまで到達する前に、別の変更が起こった場合、前述の解法では伝達途中の経路のどこかの制約オブジェクトと制約変数オブジェクトの間の辺でメッセージがすれ違いが起こり、そのすれ違いの起こった辺の両側ではそれぞれ違った値を取ってしまうことになる。

図2はメッセージがすれ違う例である。制約オブジェクトはメッセージが到達した順に処理を行うので、まず、どちらか先に到達したメッセージ(どちらが先に到達するかは、通信の遅れ等の要因による)に対する処理を行う。図2では、Aからのメッセージが先に到着したとする。Aからの値の変更報告メッセージは制約オブジェクト "A=B" によって処理され、変数Bの新しい値が計算され(図2a)、Bへの値の変更要求として伝達される(図2b)。この時、Bから送られた値の変更報告メッセージは制約とBの間の辺上でAから発生したメッセージとすれ違う。

制約変数オブジェクトBは制約オブジェクト "A=B" からの値の変更要求メッセージを受け、変数Bの値をAから送られた値、1に変える(図2c)。

一方、Bから発生したメッセージはBと制約との間でAからのメッセージとすれ違ったあと、制約オブジェクト "A=B" によって処理され(図2c)、制約変数オブジェクトAにはBから送られた値、2への変更要求メッセージが送られ(図2d)、制約変数オブジェクトAはそれに従って変数Aの値を2に変更する。最終的に、AとBは違う値を取り、制約違反の状態のままメッセージ待ちの安定状態になってしまう。

メッセージのすれ違いは局所的に同期を取る事

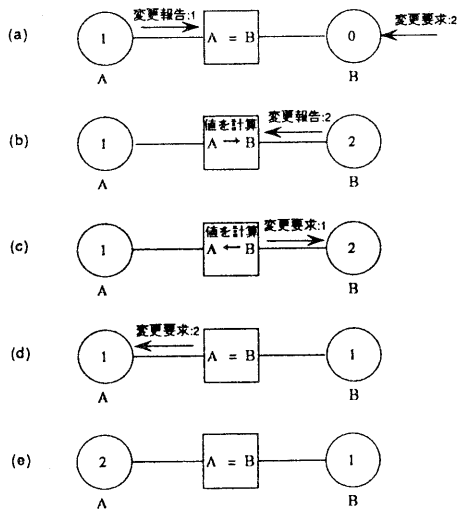

図 2: メッセージのすれ違いによる充足の失敗

によって回避する事ができる。

値の変更要求メッセージを送った制約オブジェクトは、そのメッセージを送った相手の制約変数オブジェクトからメッセージの受領を伝えるメッセージを受けるまでは、その制約変数オブジェクトに対しては、受領メッセージ待ち状態 (図 3b) とし、その制約変数オブジェクトから送られて来る値の変更報告メッセージは受け入れないようにする。

オブジェクト間の通信における 4 つ目の仮定により、ひとつの制約変数オブジェクトからある制約オブジェクトへ送られるメッセージの順序は保存されるので、受領のメッセージより先に制約オブジェクトが受けたメッセージは、制約オブジェクトが直前に送ったメッセージとすれ違ったことになる。

この方法によってメッセージのすれ違いを回避することにより、メッセージのすれ違いが発生した場合、どちらか一方のメッセージは破棄される事になる。メッセージのすれ違う場所はメッセージの発生のタイミングやデータフローグラフの構造によって、非決定的である。2 つのメッセージのうちどちらが破棄されるかは、メッセージのすれ違う場所によって決まってしまうので、どちらのメッセージが破棄されてしまうかも非決定的になってしまう。

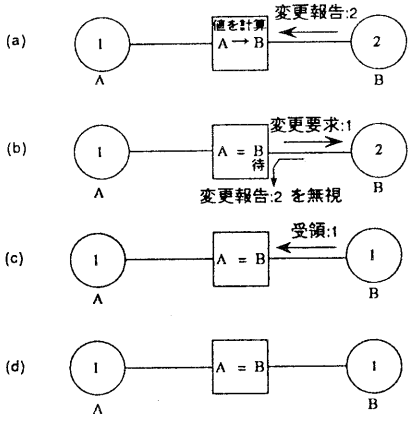

図 3: すれ違いの回避

5.3 2 変数線形制約への拡張

2 変数の線形制約式のうちの一つの変数の値が与えられれば、もう一方の変数の値は制約式から計算することができる。

従って、2 変数間のイコールの制約のみに限定した解法において、制約変数オブジェクトが送った値の変更報告をうけた制約オブジェクトは自身が持つ制約式と、報告を受けた値からもう一方の変数の値を計算し、もう一方の制約変数オブジェクトに対してその値への変更要求メッセージを出す事により、制約を充足する事ができる。

5.4 多変数制約への拡張

制約オブジェクトの持つ制約式中に 1 つ以上の有限個の変数を持つ場合を考える。

2 変数の制約のみの場合は値を計算して伝える方向が自明であったのに対し、多変数に拡張すると、制約変数オブジェクトから値の変更報告メッセージを受けた制約オブジェクトが、制約式中で用いられる複数の変数のうちどの変数の値を計算して値を伝達すべきかという選択が生じる。

まず、制約オブジェクト内で制約式中に現れる

全ての変数に対して全順序をつける。図4において制約に繋がる辺に付けられた数字がその順序であるとする。

制約オブジェクトがある制約変数オブジェクトから値の変更報告メッセージを受けたときには(図4b)、その全順序に従って値の変更報告メッセージを送った制約変数オブジェクトの次の順番の変数の値を計算してその変数に対応する制約変数オブジェクトに対して値の変更要求メッセージを出すことにする(図4c)。一番最後の順番の変数から値の変更報告メッセージが送られた場合は、一番最初の順番の変数の値を計算し、メッセージを送ることとする。

1変数のみの制約(すなわち変数の値を固定する制約)は、その変数の値が変更されると制約違反状態になるが上の解法に従って、制約式によって値を計算して、その変数に対して再び正しい値になるような値の変更要求を送ることになる(図4d)。

5.5 多変数制約の解法における問題点

この解法では、全ての制約を充足する可能な値が無い場合、無限にメッセージが制約と変数の間で行き来してしまうことになる。

5.6 インクリメンタルな制約充足への拡張

インクリメンタルな変更とは、現在の制約データフローグラフに対して新たに制約を加えたり、取り除いたりしながら次の状態の制約データフローグラフを作るものである。

データ集中型のアルゴリズムの場合は、問題全体を把握することが可能なため、インクリメンタルな制約系の変更は、制約を連続的に効率良く解くためのひとつの手法に過ぎない。しかし、データ分散型のアルゴリズムの場合は問題全体を把握することはせず、従ってデータフローグラフ全体を作り直すことはできない。そのため、データ分散型アルゴリズムでは制約系に変更を加えるためにはインクリメンタルな変更が必要である。

このインクリメンタルな変更に対して制約充足が行なえるようにアルゴリズムを拡張する。

制約系に対しては、制約オブジェクトの生成、削除、制約変数オブジェクトの生成、削除の4つの操作のみを行う。

初期状態では系内には制約変数、制約オブジェクトは存在しない。制約変数オブジェクトはそれに接続される制約オブジェクトが生成される前に生成され、制約変数オブジェクトを取り除くと、同時にその制約変数オブジェクトに接続されている全ての制約オブジェクトが系内から取り除かれる。

制約オブジェクトは生成されると、内部での順位の最も低い制約変数オブジェクトに対してその値を自分の持っている制約式と他の変数の値から計算し、値の変更要求を発行する。

制約オブジェクトを取り除く場合は、単純に接続されている制約変数オブジェクトを切り離し、取り除かれる。

制約変数オブジェクトは生成されるときには、その変数の初期値を与えられる。制約変数オブジェクトが生成された直後にはまだその変数に対しての制約はないので、初期値はユーザーやアプリケーションが設定することができる。

制約変数オブジェクトが取り除かれるときには、まずその制約変数オブジェクトに接続されている全ての制約オブジェクトに対して削除要求を出し、接続されている全ての制約オブジェクトが取り除かれてから取り除かれる。

前述の多変数線形制約の解法に対して以上の変更を行う事により、インクリメンタルな制約構成の変更における制約充足を行う事ができる。

6 制約構成エラーへの対処

制約データフローグラフに閉路がある場合は値の変更要求メッセージが無限に閉路内をまわり続けることになるため、制約の構造に閉路がないことをチェックできる必要がある。

また、制約が競合して可能な値が存在しない場合も同様に無限に変更要求が行き来してしまうので、これもチェックできるようにする必要がある。

インクリメンタルな変更に拡張した解法においてこの二つを検出するために、値の変更要求、変更報告メッセージに、その変更の元となった制約オブジェクトのIDを引数として付加する。

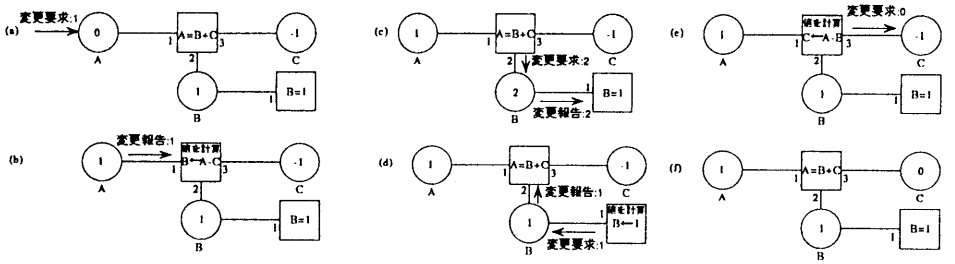

図 4: 多変数に拡張した伝達

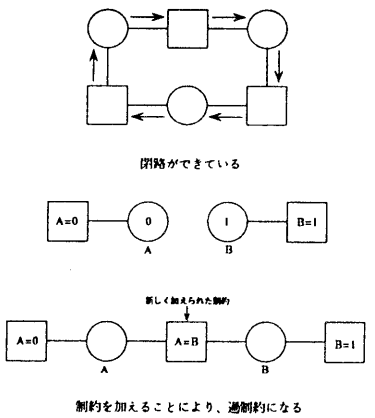

図 5: 解くことのできない制約グラフ

制約オブジェクトに自分の ID を引数としたメッセージが戻って来るのは二つの場合がある。ひとつめはその制約が値を計算して値の変更要求を出した変数が、その変更要求に対して取り得る値がない場合である。この場合は直前に変更要求を出した相手からメッセージが戻って来る。その制約オブジェクトに接続されている全ての変数オブジェクトから自分の ID を引数としたメッセージが戻って来た場合は過制約である。

もしも直前に値の変更要求を出した制約変数オブジェクト以外から自分の ID を引数に持つメッセージが戻って来た場合は制約データフローグラフに閉路があることになる。

7 アルゴリズムの実現

現在、我々の研究室では分散環境を有効にかつ安全に利用できるような分散ソフトウェア開発を支援するために、Kamui 環境という分散プログラミング環境を構築中である。

KamuiC[5] は、C++をベースにした分散オブジェクト指向言語である。KamuiC は Kamui 環境の構築の基礎となる言語であり、分散環境に透過な分散オブジェクトを単位としたプログラミングにより、安全で柔軟な分散アプリケーションを構築できる言語である。

我々は KamuiC を用いてこのアルゴリズムの実現を行なった。

図 6は、この KamuiC によって実現した制約充足システムを用いた例である。これは 4 つの点の座標それぞれに制約をつけることにより、ある点がマウスによるドラッグされても、長方形を保つようにしたものである。実際に行なっている操作は以下のようなものである。

- 4 つの点を作る (各点を点 1 、 2 、 3 、 4 とする)
- 点 1 と点 4 の x 座標が同じ値であるという制約を加える。
- 点 2 と点 3 の x 座標が同じ値であるという制約を加える。
- 点 1 と点 2 の y 座標が同じ値であるという制約を加える。
- 点 3 と点 4 の y 座標が同じ値であるという制約を加える。

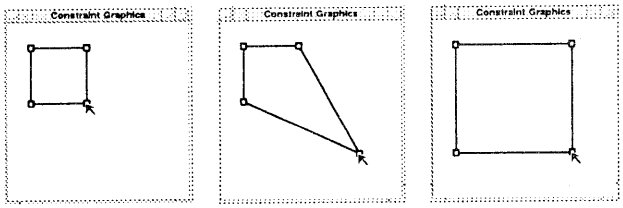

図 6: 制約による長方形の例

- マウスによるドラッグが行なわれるたびに、ドラッグされた点の座標がマウスの座標と同じであるような制約を加え、すぐに取り除く。

8 まとめ

実数領域をドメインとする変数を用いた、線形の関係を制約とする分散制約充足アルゴリズムを考案した。

分散された情報をひとつの統括された計算主体に集中させずに分散制約充足問題を解くことで、特定の計算主体にだけ大きな負担をかけることなく問題を解くことができる。

今後は以下の課題に取り組んでいきたいと考えている。

制約強度による制約の階層構造の導入 制約にその制約を充足する要求の強度を付けることにより、制約を階層的に組み立てることができるようにする。

伝達による誤差の蓄積についての考察 理論上は制約オブジェクトによる値の計算、伝達は実数を扱うことになるが、実装上では、実数の計算時には誤差が生じ、伝達が繰り返されるたびに誤差が蓄積されてしまうことになる。この誤差の蓄積による影響についての考察が必要である。

非線形制約への拡張 制約式に非線形関係を用いることができるようにする。

制約分散オブジェクト指向言語の作成 [4] では、命令法プログラミング言語と制約プログラミング言語のパラダイムの融合をはかっている。

これをもとに、KamuiC 上で構築した制約充足システムを、制約分散オブジェクト指向言語として実装することを目指したいと考えている。

参考文献

[1] Bjorn N. Freeman-Benson, John Maloney, and Alan Borning: An Incremental Constraint Solver, *Communications of the ACM* 33(1), 1990, pp.54-63

[2] William Leler: Constraint Programming Languages: Their Specfication and Generation, Addison-Wesley, 1988

[3] 横尾 真:分散制約充足アルゴリズム, 情報処理学会ソフトウェア基礎論プログラミング言語合同研究会 (89-SF-33,89-PL-23), 1989, pp.9-16

[4] Gus Lopez,Bjorn Freeman-Benson,Alan Borning : Implementing Constraint Imperative Programming Languages: The Kaleidoscope'93 Virtual Machine, *OOPSLA '94*, 1994, pp.259-271

[5] 原田 康徳, 浜田 昇, 渡辺 慎哉, 三谷 和史, 宮本 衛市 : 並列オブジェクト指向モデルに基づく分散型アプリケーション構築法, 情報処理学会研究報告 vol.89-ARC-77-9, 1989.