人　工　知　能　98－5
（1995．1．18）

制約充足問題における各解法の分散協調問題解決への拡張に関する考察

内角 真† 今成 文明 小川 均

立命館大学理工学部情報学科

分散制約充足問題は分散協調問題解決の分野における多くの問題を定式化することが可能であり，近年注目を集めている重要な研究分野の１つである．しかし，この分野は最近発展してきたものであり，現在までに分散制約充足問題に対するアルゴリズムとして提案されているものは少ない．本論文では既存の制約充足アルゴリズムを分散制約充足アルゴリズムへと拡張することを考え，まず既存の制約充足アルゴリズムを分類・分析し，分散環境へ適合するための性質について考察を行なった．そしてその考察の結果より，いくつかの新しい分散制約充足アルゴリズムの形を得た．またそれらのアルゴリズムの性能について定性的な評価・考察を行なった．

Towards the Extension of Constraint Satisfaction Algorithms for Distributed Constraint Satisfaction Problem

Makoto Uchikado, Fumiaki Imanari, Hitoshi Ogawa

Department of Computer Science,
Faculity of Science and Engineering,
Ritsumeikan University

Distributed Constraint Satisfaction Problem(DCSP), which can formalize many problems in Distributed Cooperative Problem Solving, is one of an important research field. But, this research field has developed recently, so only few algorithms are proposed for DCSP. Therefore, in this paper, we analyze existing constraint satisfaction algorithms, and discuss what kind of character is needed in order to extend them to solve DCSP. From the conclusion of analyses, some new algorithms are formalized, and their performances are evaluated qualitatively.

†連絡先 滋賀県草津市野路町 1916　E-mail: uchikado@airlab.cs.ritsumei.ac.jp

1 はじめに

分散制約充足問題はマルチエージェントモデルにおいて複数のエージェント間に分割された制約を充足するような解を求める問題であり，AIの多くの問題を定式化することができる制約充足問題を分散環境へ適用したものであるとみなすことができる．また，分散制約充足問題は分散協調問題解決が対象とする問題の1つであり，分散協調問題解決の分野はAIにおいて近年注目を集めている重要な分野の1つである[石田 92]．

しかし，分散制約充足問題に対するアルゴリズムとして現在までに提案されているものは少ない．

一方，制約充足問題に関してはすでに数多くの研究が行なわれており，この問題に対する有効なアルゴリズムもいくつか提案されている．

そこで，本論文では制約充足問題(constraint satisfaction problem, CSP)に対するアルゴリズムを基にした分散制約充足問題(distributed constraint satisfaction problem, DCSP)を解くアルゴリズムへの拡張に関して考察をする．

始めに現在までに提案されているいくつかの制約充足問題を解くアルゴリズムについて述べ，その分類を行なう．次に，分散制約充足問題を解くためのアルゴリズムが持つべき性質を挙げる．そしてこの性質をふまえた上で，制約充足問題を解くアルゴリズムの分散制約充足問題への拡張を試み，その拡張されたアルゴリズムに対し分散環境，問題の大きさの観点からの実行効率への影響に関しての考察を行なう．

2 制約充足問題とアルゴリズム

2.1 制約充足問題

制約充足問題(CSP)は，一般にn個の変数$(X_1, X_2, \ldots, X_n)$，各変数の取りえる値の集合(定義域)$(D_{X_1}, D_{X_2}, \ldots, D_{X_n})$，変数間の制約の集合Cを与えることによって定義される．本論文では変数間の制約は述語によって内包的に定義されるものとする．例えば変数 $X_1 = 1$ に対して変数 $X_2 = 3$ が制約として認められることを$\{(X_1,1), (X_2,3)\}$のように表す．

なお本論文では，上の制約充足問題を分散制約充足問題との対比を明確にしたい際には特に集中型制約充足問題と呼ぶこととする．

制約充足問題の解とは，問題中の全変数に値が割りあてられ，かつ問題中のすべての制約を満足している状態における全変数の値の組のことである．また本論文では問題に対する可能な解をすべて求めた時，制約充足問題が解けたという．

2.2 集中型制約充足アルゴリズム

本論文では集中型制約充足アルゴリズムのうち以下のものを考察の対象とする．

1. backtrack
2. 山登り法 + min conflict [Minton 90]
3. weak commitment [Yokoo 94]
4. breakout,fill algorithm [Morris 93]

2.3 制約充足アルゴリズムの分類

制約充足アルゴリズムは，変数に値を割当てる際の処理に注目することにより次の2つに大別できる．

backtrack系 変数間に順序関係が設定されており，上位の変数から順に変数値を束縛していく．ある変数の値を決定する際には，すでに束縛されている変数値と矛盾しない値を選ぶことで解を得る．探索処理中束縛されている変数の集合について，その集合内での制約違反数は常に0である．

iterative improvement系 各変数には常に値が代入されており，“全変数の中から変数を1つ選び出し，その値を評価値が良くなるような値に変更する”処理を繰り返すことで解を得る．探索中問題全体の制約違反数は0ではなく，探索は制約違反数をしだいに下げるような方向へ進む．

2.2で示した各アルゴリズムのうち，2,3はbacktrack系であり，1,4はiterative improvement系である．

なお制約充足問題に適用するためのアルゴリズムとして上記の他にarc consistency [Nadel 88]がある．この手続きは探索開始前に，問題の制約を見て明らかに制約違反をおこすことがわかるような値をその変数のドメインから削除することを目的とした事前処理の手続きである．

arc consistency手続自体には解を探索する能力がないことから本論文では考慮しない．

3 制約充足アルゴリズムの分散化

3.1 分散制約充足問題

分散制約充足問題 (DCSP) とは，制約充足問題の変数と制約が複数エージェントに分散して配置されるような問題である．本論文ではさらに以下のような仮定をする．

- 変数の数，変数のドメインの大きさ，制約の数は有限である．また各変数のドメインの大きさは等しいとする．
- 制約は 2 変数間の制約 (binary constraint) に限定する．
- 各エージェントは自分に配置された変数に関する制約をすべて知っている．

またエージェント間の通信に関しては以下のような仮定をする．

- 直接通信が可能である．
- 通信コストは一定値をとる．
- エージェントの出すメッセージの発信順序は保存される．
- 非同期で通信を行なうものとし，各エージェントは届いたメッセージを自動的にキューに保存しておいて適当な時刻に参照できる．

3.2 分散の種類

一般的に分散制約充足問題を考える時，以下のような特徴によって場合分けすることができる．

分散形態： 変数や変数のドメインをエージェントへ分割した際の形態であり以下のものが考えられる．

変数分散: 1 つのエージェントが 1 つの変数を担当するような形態

問題分散: 1 つのエージェントが複数の変数を担当するような形態

探索空間分散: 集中型制約充足問題の探索空間を縦にわったそれぞれの空間がエージェントにわりあてられるような形態

各分散形態と探索木上のノードのエージェントへの割り当ての関係を図 1に示す．この図の (a) は変数分散，(b) は問題分散，(c) 探索空間分散をそれぞれ示している．

メッセージの種類： 各エージェント間でのメッセージの種類には 現在の値状況の告知などのエージェントの内部情報の一部，値変更要請などの他エージェントへの制御メッセージの 2 種類が考えられる．

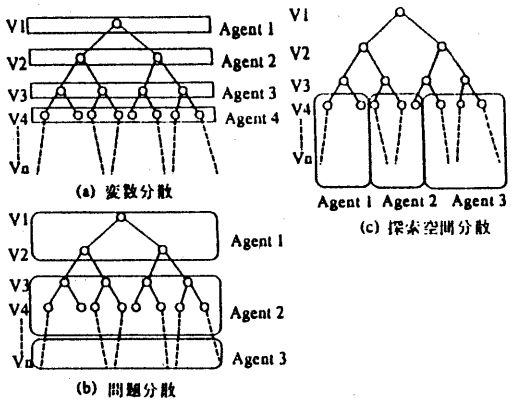

図 1: 分散の各形態

メッセージの交換方法： 他エージェントの内部の様子や，メッセージの受け渡し方法としては，エージェント間通信，共有ブラックボードの利用，他エージェントの情報を自由に参照できるものの 3 形態が考えられる．

本論文では上記のうちエージェント間通信によるものを対象とする．

3.3 アルゴリズムの分散化における考慮点

まず分散形態についての考察を行なう．探索空間分散はそれぞれのエージェントに割当てられる処理が他エージェントとの協調動作を必要としない点において他の形態と比べ特殊であり集中型の考え方をほぼそのまま利用できる利点がある．しかしこの形態はマルチエージェントによる協調問題解決のモデルと適合しない点が多く，むしろ並列計算を行なうようなモデルに合った形態であると言える．

以下では，主に変数分散，問題分散に関して考察を行なう．

今，ある変数と制約関係のある変数の集合をその変数の近傍と呼ぶことにする．またあるエージェントに配置された変数と制約関係にある変数を含むようなエージェントをそのエージェントの近傍と呼ぶことにする．

backtrack 系のアルゴリズムでは変数の値の決定の際にその時点で束縛されていない変数の

中から1つを選択し，値決定処理を行なう．しかし，分散環境においてはこのような処理を行なうと分散環境での利点の1つである並列性を生かせない．よってこのような性質は分散アルゴリズムには適当でないと言える．

分散環境では一般的に常に全てのエージェントの値を知るようなエージェントは存在しない．これより次のことが言える．

1. 問題中の全変数の値を見て次の動作を決定するような処理を正確に実現することはできない．近似的に実現することは可能ではあるが，これは非常に効率の悪く不正確なものとなるおそれがある．
2. アルゴリズムの完全性を保証することが難しい．

1. については min-conflict がその例である．

分散環境においては，ある時点での全エージェントの様子を正確に知ることはできない．しかしエージェントは自分の値が他エージェントの値と制約充足しているかを確認するために最低限近傍の値に関しての情報を得る必要がある．このような機能をエージェントに付加するために view と呼ばれる概念を導入する [横尾 92]．

view はエージェントに対して個別に用意され，自エージェントの近傍の変数の値に関する最新情報を保存しておく領域である．これは変数の ID とその変数の値の組の列で構成される．

本論文では，エージェントは自分の持つ変数の現在の値が制約充足しているかを判定するために view を参照するものとする．

本論文では，アルゴリズムが次のような性質を持つとき「そのアルゴリズムは完全性を持つ」と呼ぶ．

完全性の性質：問題に解があれば全て表示する．1つも解がなければないことを判定できアルゴリズムが停止する．

以下ではアルゴリズムに完全性を持たせるために必要な性質について考察を行なう．

分散制約アルゴリズムの完全性を保証するためには少なくとも無限ループ，局所最適，デッドロックの回避をすることが重要である．ここで無限ループとは探索を続けてもその先に解がないような状態に問題全体が何度もおちいることであり，局所最適とは，エージェントが自分の現在の値をより良く変更できない状態，デッドロックは複数のエージェントがお互いにお互いの応答を待つ状態におちいることであるとする．

集中型の backtrack アルゴリズムでは変数に全順序関係を付け，順序関係で上にある変数は順序関係が下の変数が張る探索空間をすべて探索しつくし，しかも解が存在しなかった時点ではじめて別の値への変更処理が行なわれる．このような機構はアルゴリズムの完全性を保証するために大変重要なものであると言える．

iterative improvement 系は一般に，上のようなことが行なわれないためアルゴリズムが全空間を探索しつくしたことの判定ができない．一般に iterative improvement 系のアルゴリズムが停止するのは解を得た状態のみであるため，解が1つも存在しないような過制約な問題を解くことを考えるとこのようなアルゴリズムは停止することができない．従って完全性を保証することができない．

4 分散制約充足アルゴリズム

分散形態について 3.2での 3 形態について個別にアルゴリズムの検討を行なう．

4.1 変数分散

変数分散アルゴリズムの基本的な形として以下のようなものを考える．

並列性を生かすために各エージェントは非同期に各自の持つ変数への値割当てを行なう．そして近傍との間での制約違反の判定を行ない，その結果によりさらに値の変更を行なうかどうかを判定する．

2.3の集中型のアルゴリズムの分類にならい，変数分散アルゴリズムの種類として，まず処理中の制約違反数に関して

(v_a1) 制約違反数を常に 0 に保つ
(v_a2) 制約違反数をしだいに 0 へ下げる

の 2 通りについて考慮する．

ここで完全性についての考察を行なう．

(v_a1) では，局所最適におちいった場合に，近傍エージェントの1つに対して値変更要求を出し近傍の値を変更することで脱出をする方法が考えられる．しかし，この機構だけでは何度も同じ局所最適におちいる可能性がある．そこで局所最適におちいった時点の状態 (関係する変数の値など) を記録して近傍エージェントへ送

信するとする．これにより近傍エージェントは値を変更する際にこのエージェントから送られたnogood情報を参照し同じ状態へ遷移しないようすることでこの可能性を除去することができる．

次に無限ループについてであるが，自分の値がview中の変数と矛盾している際にそれを解消する必要があるが，この時双方がともに値を変更するアルゴリズムでは一般に，完全に無限ループを回避することはできない．従ってどちらか一方だけが値を変更するようなアルゴリズムにする必要があり，各エージェントは制約違反を発見した時に自分の値を変更するかしないかを判断しなければならない．

この判断をするために同期をとる方法を用いるとデッドロックにおちいる可能性が出てくるため完全性を保証することができなくなる．よってこの方法は適当ではない．そこでエージェントに順序関係を定義し，自分と相手の順序関係のうち下位になる方のエージェントが値を変更することとする．

[横尾 92]においてはこの順序関係を完全性を保証するために入れた全順序を使用しているが，ここで使用する順序関係は必ずしも先に定義した全順序である必要はなく，例えば各エージェントの過去の値変更回数を用いて順序を決定するものが考えられる．

(v_a2)は，iterative improvement系の性質を持つものであり，問題に解がない(過制約)場合に停止できない．よってこのアルゴリズムは以下では考慮しないものとする．

次に，viewを得るための方法としては以下の2つのものを考慮する．

(v_b1) 値変更を行なったエージェントがその時点でメッセージ(noticeメッセージ)により自分の値の変更を他に知らせる方法．

(v_b2) ある変数について値を知りたい時にその値を持っているエージェントへ値の確認要求(requestメッセージ)を出し，受け取ったエージェントがそれに答える方法．

(v_b2)では，requestを出した相手エージェントからの応答がない限りそのエージェントはviewの更新ができない．このためにデッドロックが発生する危険性が生じる．

そこでこれを解消するためにrequestメッセージを出した後，他エージェントからのrequestメッセージへの応答処理を行ないながら，返答を待つ形をとるものとする．

(v_b1)はこの点の問題はない．

また(v_b1)では，noticeメッセージの送信先として以下の2通りのものが考えられる．

- (v_c1) 全エージェントへ送信する方法
- (v_c2) 近傍エージェントにのみ送信する方法

以上の考察から変数分散のアルゴリズムとしては

(V1) (v_a1) + (v_b1) + (v_c1)
(V2) (v_a1) + (v_b1) + (v_c2)
(V3) (v_a1) + (v_b2)

の3種類の形のアルゴリズムが妥当であると言える．

これらのアルゴリズムにおいて，エージェント間で受け渡されるメッセージの基本的なものは以下のものである．

notice(ID,value) agent IDからの値告知メッセージ

nogood(Valuelist) nogoodメッセージ(値変更要求の意味は持たない) Valuelistがnogoodである

change(ID,your_value,my_value) agent IDからの値変更要求メッセージ．送信側の値をmy_valueに受信側の値をyour_valueに代入し送る．

request(ID) エージェントIDからの値告知の要請．(v_b2)版の時に使用．

さらにエージェントにはviewのほかにnogoodを記録しておく機能も必要である．これをnogoodリストと呼ぶ．

変数分散アルゴリズムのうちV2についてその流れを図2に示す．他のアルゴリズムについては5.の考察によりV2よりも効率が悪いことがわかるため詳細は省略をするとし，ここではV2との違いについて言及する．V1は値の変更後に他エージェントへ向けてnoticeメッセージを発信する際に近傍だけでなく全エージェントへ向けて発信を行なう．またviewと自分の値との制約充足を調べる際にもすべてのエージェントの情報と判定を行なうことになる．

V3はviewを得る際の処理がnoticeメッセージからではなく，他エージェントへ値告知を要求し，その返答によって変更するという点で異なる．このためアルゴリズム上はviewを更新する箇所で近傍へrequestメッセージを送信しその返答を待つ処理が加わる．この時deadlockを避けるために他エージェントからのrequestメッ

```
(1) 初期値を notice を用いて近傍エージェントへ送信
(2) 他エージェントからの，notice，nogood メッセージ
    を受けとり view,nogood リストの更新．
(3) 近傍エージェントからの change メッセージを得る．
(4) if (view の中に自分の値との間で制約違反を
    おこしている変数があり，さらにその変数の
    属するエージェントが自分より順序が上である
    or view と自分の値が nogood である
    or (3) において change メッセージを受けた) then
      if (自分の変数の値の変更が可能である) then
          値を変更し，その値を notice メッセージで
          近傍エージェントへ送信する．
      else
          view を nogood リストに追加．
          この nogood を近傍のエージェントへ送信．
          change メッセージを view の中の 1 つの
          エージェントへ向けて送信．
      endif
    endif
    (2) へ
```

図 2: 変数分散用アルゴリズム V2

セージの処理を同時に行なうものとする．また，自分の値を変更した際の notice メッセージを送る処理が不要である点が異なる．

4.2 問題分散

問題分散において，個々のエージェントから見た時の問題中の制約は次の 2 種類に分類される．

外部制約：制約関係のある 2 変数がそれぞれ別のエージェントに配置されている時のその 2 変数間の制約

内部制約：制約関係のある 2 変数が両方とも同じエージェント内にある時のその 2 変数間の制約

このことより，アルゴリズムの形として以下の 2 通りのものが考えられる

(p_a1) 内部制約を先に充足してから外部制約を充足しようとするもの

(p_a2) 外部制約を先に充足してから内部制約を充足しようとするもの

外部制約を充足するために変数分散のアルゴリズムが，内部制約を充足するために集中型のアルゴリズムがそれぞれ利用できる．

変数分散のアルゴリズムは 4.1 で述べた 3 通りのものが考えられる．集中型のアルゴリズムは 2.3 で述べたように 2 通りのものが考えられるが，3.3 の考察により iterative improvement 系

```
(1) 内部問題を解く
    if (解がない) then
      その問題には解がない．失敗．全体を停止
    else
      その解を notice で近傍エージェントへ送信
    endif
(2) notice,nogood を受けとり処理
(3) change メッセージを受けとり処理
(4) if (view の中に自分の値の組と制約違反をおこしている
       変数があり，さらにその変数の属するエージェント
       が自分より順序が上である
     or view と自分の値の組が nogood である
     or change メッセージを受信した) then
      if (今外部と違反しているとわかった変数 (nogood 時
      は view からランダムに 1 つ選ぶ) と外部との制約が
      ない変数のみを変更して内部問題の解が
      得られるか) then
        その値に変更して notice を近傍へ送信．
      else
        今の view を nogood として登録し近傍へ送信．
        外部変数の束縛をすべて解放し，内部問題を解く．
        if 解がある then
          その値の組を代入
          notice メッセージを近傍へ送信
        else
          問題に解がない．失敗．全体を停止
        endif
      endif
    endif
    (2) へ
```

図 3: 問題分散用アルゴリズム P2

のものは完全性を保証することができない．よってここでは backtrack 系のみを考慮する．

以上より問題分散のアルゴリズムは次の 6 通りの組合せが可能である．

変数分散 + 先に処理する制約	
(P1) V1	内部
(P2) V2	内部
(P3) V3	内部
(P4) V1	外部
(P5) V2	外部
(P6) V3	外部

上記のアルゴリズムのうち内部制約を先に処理するものの中から P2，外部制約を先に処理するものの中から P5 の 2 つのアルゴリズムについての大まかな流れを図 3, 図 4 に示す．

5. の考察により他のアルゴリズムに比べ P2,P5 は効率が良いことがわかるため本論文では以下，問題分散のアルゴリズムとしては P2,P5 のみを考慮する．よって他のアルゴリズムについての詳細はここでは省略する．

```
(1) 初期値を近傍へ送信
(2) notice メッセージ, nogood メッセージの受信と
    view, nogood リストの更新をする
(3) change メッセージを受信
(4) if (view の中に自分の値の組と制約違反をおこしている
       変数があり，さらにその変数の属するエージェント
       が自分より順序が上である
     or view と自分の値の組が nogood である
     or change メッセージを受信した) then
      今違反しているとわかった変数 (nogood 時は view から
      ランダムに 1 つ選ぶ) のみを unbind して
      外部制約を充足する解が得られるか
      if (そのような値を得た) then
         その値を notice で近傍へ送信
         内部制約からなる問題を解く
         (この時外部との間に制約のある変数は
          動かさない)
         if (解がある) then
           その値の組を割当てる
         else
           今の view を nogood として登録し
           近傍 agent へ送信.
           そのうちの 1 つに change メッセージを発信.
         endif
      else
       今の view を nogood として登録し
       近傍 agent へ送信.
       そのうちの 1 つに change メッセージを発信.
      endif
    else
       内部制約からなる問題を解く
       (この時外部との間に制約のある変数は動かさない)
       if (解がある) then
        その値の組を割当てる
       else
         今の view を nogood として登録し
         近傍 agent へ送信.
         そのうちの 1 つに change メッセージを発信
       endif
    endif
    (2) へ
```

図 4: 問題分散用アルゴリズム P5

しかし，他のアルゴリズムについての流れは外部制約を処理する際の手続きが P2,P5 と異なるのみであるので，これらのアルゴリズムを基に一部変形することで容易に得ることができる．

なお，これらの図中 内部問題とはエージェントに配置されている変数についてそれらの間に定義されている制約 (内部制約) のみを考慮した制約充足問題を指す．

4.3 探索空間分散

このアルゴリズムでは各エージェントはそれぞれ探索空間の部分空間を与えられ，それぞれ集中型のアルゴリズムを用いてその空間の探索を行なうものである．

集中型のアルゴリズムは 2 種類が考えられるが，完全性を保証するために backtrack 系のアルゴリズムを用いるとする．

さらに探索を行なっている途中，これ以上探索を続けても解を発見できないことが判明した時点で束縛されている変数の値の組を nogood として記録しその情報を他エージェントへ送信するものとする．この nogood 情報を各エージェントの探索に利用するために，集中型 backtrack 系のアルゴリズムに対して以下の機能を付加する必要がある．

- 値割当ての際に nogood を見ての枝刈り
- 探索途中にメッセージキューを見て届いたメッセージ (nogood) の処理

また生成する nogood を他エージェントにとって意味のあるものにするために各エージェントは完全な変数を優先的に探索する必要がある．

ここでエージェントにとっての完全な変数とは，エージェントがその変数のドメインの全体を探索する変数のことを指す．また，不完全な変数とはエージェントがその変数のドメインに関してその一部を探索するような変数のこととする．

例えば図 1(c) において，変数 V4~Vn は完全な変数である．また，変数 V3 は不完全な変数である．

集中型のアルゴリズムは backtrack 系のみ考慮するので探索空間分散についてのアルゴリズムは 1 通りである．

このアルゴリズム (T1 と呼ぶ) を図 5に示す．

```
VARS_LEFT <- 変数のリスト (完全な変数を前に配置する)
VARS_DONE <- []
で以下の手続き bt を呼び出す.

procedure bt(VARS_LEFT,VARS_DONE)
    VARS_LEFT = [] なら今の値割当てが解
    V <- VARS_LEFT の先頭
   D <- 変数 V のドメイン
    for each val in D do
        V に値 val を代入
        他エージェントからの nogood メッセージを
        受信. 送られてきているなら nogood リスト
        へ追加
        if (VARS_DONE と V が制約充足 &
            nogood ではない) then
          bt(VARS_LEFT - V, VARS_DONE +(V,val))
          を呼び出す
        endif
    endfor
    VARS_DONE を nogood として追加
    他エージェントへ送信
```

図 5: アルゴリズム T1

5 アルゴリズムの評価・考察

以下では各分散アルゴリズムについて，そのコストを計算し，各アルゴリズム間での比較・評価を行なう.

ここでは解を 1 つ求めた状態になるまでのコスト (時間) でアルゴリズムを比較することとする.

まず考察対象としたアルゴリズムについて，そのコストを示す.

コストの計算のために使用している変数について以下に示す.

- N : 問題に含まれる変数の数
- K : 各変数のドメイン
- Tsrchva : 変数分散において view が全エージェントの内容を含む際の，今の値が他の変数と違反しているかを判定するコスト.
- Tsrchv : 変数分散において view が近傍エージェントの内容を含む際の，今の値が他の変数と違反しているかを判定するコスト.
- Tchk : 2 つの変数値が制約充足しているかをチェックするコスト. (nogood をチェックするコストも含む)
- Tcom : 2agent 間の通信コストの期待値 (送り先での受信処理のコストを含む)
- Nvarcon : 1 つの変数について，その近傍である変数の数の平均
- Nitev : 変数分散において，エージェントが解を得るまでの間に値変更処理を実行する回数の期待値
- Tsrchp : 問題分散において，今の値の組が外部の変数と違反しているかを判定するコスト.
- m : 分散アルゴリズムにおいて，エージェント数
- Ni : 問題分散において，エージェント i の変数の数
- Nagtcon : 問題分散において，1 つのエージェントの近傍のエージェント数の期待値
- NIi : 問題分散において，エージェント i の変数のうち内部制約のみを持つような変数の数
- Nitcp : 問題分散において，エージェントが解を得るまでの間に値変更処理を実行する回数の期待値
- NCi : 探索空間分散においてエージェント i に配置された完全な変数の数
- NDi : エージェント i に配置された不完全な変数の数
- Nki : エージェント i に配置された不完全な変数のドメインの大きさの期待値
- d : 通信コスト/制約チェック の比. $\frac{Tcom}{Tchk}$ で表される. 通常 1 より大きいと考えられる.

5.1 各アルゴリズムの計算コスト

4. であげた各分散形態の各アルゴリズムについてそのコストの評価結果を示す.

ここでは各アルゴリズムの条件分岐の条件部はすべて $\frac{1}{2}$ の確率で真となるという近似を行なっている.

なお，後の比較のため集中型の backtrack アルゴリズムに関してのコストについてもあげておく.

[変数分散]

(アルゴリズム V1)

最良 $(N-1)*Tcom+Tsrchva$

最悪 $(N-1)*Tcom+[Tsrchva+K*Tchk+(Nvarcon+1)*Tcom]*K^{N-1}$

平均 $(N-1)*Tcom+Nitcv*Tsrchva+Nitcv*[\frac{1}{2}*\{\frac{K}{2}*Tchk+\frac{1}{2}*((N-1)*Tcom)+\frac{1}{2}*(Nvarcon+1)*Tcom\}]$

(アルゴリズム V2)

最良 $Nvarcon*Tcom+Tsrchv$

最悪 $Nvarcon*Tcom+[Tsrchv+K*Tchk+(Nvarcon+1)*Tcom]*K^{N-1}$

平均 $Nvarcon*Tcom+Nitev*[Tsrchv+\frac{1}{2}*\{\frac{K}{2}*Tchk+\frac{1}{2}*Nvarcon*Tcom+\frac{1}{2}*(Nvarcon+1)*Tcom\}]$

(アルゴリズム V3)

最良 $Nvarcon*2*Tcom+Tsrchv$

最悪 $[Nvarcon*2*Tcom+Tsrchv+K*Tchk+(Nvarcon+1)*Tcom]*K^{N-1}$

平均 $[Nvarcon*2*Tcom+Tsrchv+\frac{1}{2}*\{\frac{K}{2}*Tchk+\frac{1}{2}*Nvarcon*Tcom+\frac{1}{2}*(Nvarcon+1)*Tcom\}]*Nitev$

[問題分散]

(アルゴリズム P2)

最良 $Ni*Tchk+Nagtcon*Tcom+Tsrchp$

最悪 $K^{Ni}*Tchk+Nagtcon*Tcom+[Tsrchp+K^{NIi+1}*Tchk+(Nagtcon+1)*Tcom+K^{Ni}*Tchk+Nagtcon*Tcom]*K^{N-Ni}$

平均 $(\frac{K}{2})^{Ni}*Tchk+Nagtcon*Tcom+Nitcp*[Tsrchp+\frac{1}{2}*\{(\frac{K}{2})^{NIi+1}*Tchk+\frac{1}{2}*Nagtcon*Tcom+\frac{1}{2}\{(Nagtcon+1)*Tcom+(\frac{K}{2})^{Ni}*Tchk+\frac{1}{2}Nagtcon*Tcom\}\}]$

(アルゴリズム P5)

最良 $Nagtcon*Tcom+Tsrchp+NIi*Tchk$

最悪 $Nagtcon*Tcom+[Tsrchp+K*Tchk+(Nagtcon+1)*Tcom]*K^{N-Ni}$

平均 $Nagtcon*Tcom+Nitep*[Tsrchp+\frac{1}{2}*\{\frac{K}{2}*Tchk+\frac{1}{2}*\{Nagtcon*Tcom+(\frac{K}{2})^{NIi}*Tchk+\frac{1}{2}*(Nagtcon+1)*Tcom\}+\frac{1}{2}*(Nagtcon+1)*Tcom\}+\frac{1}{2}*\{(\frac{K}{2})^{NIi}*Tchk+\frac{1}{2}*(Nagtcon+1)*Tcom\}]$

[探索空間分散]

最良 $(NCi+NDi)*Tchk$

最悪 $K^{NCi}*Nki^{NDi}*Tchk+K^{NCi}*Nki^{NDi}*(m-1)*Tcom$

平均 $(\frac{K}{2})^{NCi}*(\frac{Nki}{2})^{NDi}*Tchk+(\frac{K}{2})^{NCi}*(\frac{Nki}{2})^{NDi}*(m-1)*Tcom$

[集中型]

最良 $N*Tchk$

最悪 $K^{N}*Tchk$

平均 $(\frac{K}{2})^{N}*Tchk$

5.2 同一分散形態内でのアルゴリズムの比較

まず各分散形態について，同じ分散形態内でアルゴリズムが複数存在するものについてそのコストの評価を行なった．

5.2.1 変数分散

変数分散については V2, V1 と V3, V2 についてそれぞれ最良値，最悪値，平均値についての比較を行なった．そして以下のような結果を得た．

- V1，V2 の比較に関してはすべての値で $V1 > V2$，つまり V2 のほうが良い．
- V2，V3 の比較に関してはすべての値で $V3 > V2$，つまり V2 のほうが良い．(ただし平均値の比較の際に $Nitev > \frac{1}{2}$であると仮定した)

以上より変数分散においてはアルゴリズム V2 が最も良いアルゴリズムであるという結果を得た．

5.2.2 問題分散

問題分散については 変数分散の箇所にアルゴリズム V2 を用いた P2,P5 が他に比べて良いと推測されるので，ここではこの 2 つについての最良値，最悪値，平均値についての比較のみを行ない，以下のような結果を得た．

- 最良値，最悪値に関しては $Ni > NIi, N > Ni$ であることより，ともに P5 のほうが良い．
  平均値に関しては，$N = 100, Ni = 10 (m = 10)$ という状況を想定し考察を行なった．
  もし K が大きい (K=100) ならば，$d < 10^{17} (= (\frac{K}{2})^{Ni})$ を仮定すれば P5 のほうが良いことがわかった．この仮定は成立すると言えるものであろう．
  また K が小さい (K=2) 場合には P2 のほうが良いという結果を得た．

以上より 最良値，最悪値についてはどのような場合においても P5 が良いと言え，平均値に関しては，変数のドメインの大小により効率の良さが変わるという結果を得た．

5.3 各分散形態間

以下異なる分散形態間と集中型との間の比較の結果を示す．ここでは簡単のため平均コストのみを考慮して比較・評価を行なった．

ここでの計算には次のような仮定・近似を用いている．

- 変数分散に関して
  Tsrchv = $\frac{Nvarcon}{2}$ *Tchk, Nitev $\geq$ 1, N-1 > Nvarcon $\geq$ 1
- 問題分散に関して
  Tsrchp = (Ni- NIi)* Tsrchv, Nagtcon > 0, NIi < Ni, Ni < N, 問題が各エージェントに均等に配置されるとして Ni = $\frac{N}{m}$
- 探索空間分散に関して
  Nagtcon < Nvarcon, 探索空間が各エージェントに均等に配置されるものとして NCi = N - log_K m -1, NDi = 1, Nki = $\frac{K}{m}$

また，問題分散と変数分散の比較の際には Nitep = Nitev, 問題分散と探索空間分散に関しては両者のエージェント数 m は等しいという近似を行なっている．

ここでは集中型アルゴリズムを Ct と呼ぶことにする．

以下の比較では変数のドメインの大きさと 通信/制約チェック コストの大きさによる各アルゴリズムの効率を比較することを目的とするため，N,Ni,m に関しては適当と思われる値 $N = 100, Ni = 10, m = 10$ を仮定した．問題分散との比較においては 5.2.2 の結果より問題の大きさによって P2,P5 のうちの良いほうをとっている．

集中・変数分散

- K が大きい (K=100) 時には，$10^{170} (= (\frac{K}{2})^{N})$ > d*Nitev であれば V2 が良い．
- K が小さい (K=2) 時には，Ct が良い．

集中・問題分散

• K が大きい (K=100) 時には，$10^{169}(=(\frac{K}{2})^{N}\frac{8}{9Nagtcon+5}) > d$ *Nitep であれば P5 が良い．

• K が小さい (K=2) 時には，Ct が良い．

集中・探索空間分散

• K が大きい (K=100) 時には，$d > 7.7(=\{m(\frac{K}{2})^{\log_K m}-1\}\frac{1}{m-1})$ であれば Ct が良く，$d < 7.7$ であれば T1 が良い．

• K が小さい (K=2) 時には，$d > 1(=(\frac{2m}{K}-1)\frac{1}{m-1})$ であれば Ct が良い．

変数分散・問題分散

• K が大きい (K=100) 時には，V2 が良い．

• K が小さい (K=2) 時には，V2 が良い．

変数分散・探索空間分散

• K が大きい (K=100) 時には，$10^{169}(=(m-1)\frac{K}{2m}(\frac{K}{2})^{N-\log_K m-1}) >$ Nitev であれば V2 が良い．

• K が小さい (K=2) 時には，T1 が良い．

問題分散・探索空間分散

• K が大きい (K=100) 時には，$10^{169}(=(m-1)\frac{K}{2m}(\frac{K}{2})^{N-\log_K m-1}) >$ Nitep であれば P5 が良い．

• K が小さい (K=2) 時には，T1 が良い．

以上のことから次のようなことが言える．

- 問題が大きい，つまり変数のドメインが大きい場合には集中型は変数分散，問題分散型のアルゴリズムより効率が悪くなる
- 逆に問題が小さい，つまり変数のドメインが小さい場合には集中型は他のどの分散型アルゴリズムよりも効率が良くなる
- 問題が大きい場合に各分散アルゴリズムを比較すると，V2, P5, T1 の順によい．
- 問題が小さい場合に各分散アルゴリズムを比較すると，T1, V2, P2 の順によい．

よって以上のことから，問題が大きい状況では分散型，特に変数分散が効率が良く，問題が小さい状況では分散型，特に問題分散が効率が悪くなり集中型が一番良いという結果を得た．

6 おわりに

本論文では，現在までに提案されている集中型制約充足アルゴリズムをその特徴により分類し，分散制約充足アルゴリズムを考える際に考慮すべき性質をあげた．またそれに基づいて，既存の制約充足アルゴリズムを基に分散制約充足問題を解くアルゴリズムへの拡張を行ない，分散制約充足アルゴリズムの形をいくつか示した．さらにそれらのアルゴリズムの特定の状況における計算コストに関しての評価・考察を行なった．

その結果として，変数のドメインが大きくなれば集中型アルゴリズムに比べ分散型アルゴリズムの効率が上がることを示した．

この考察では多くのパラメータをあらかじめ仮定しているため，その値が妥当であるかについては実際に実験を行なって確認を行なう必要があると思われる．

今後の課題として，実際の問題に対して実験を行ない，より詳細な評価を行なうことが重要である．

参考文献

[横尾 92] 横尾 真, エドモンド H. ダーフィ, 石田 亨, 桑原 和宏: 分散制約充足問題の定式化とその解法, 電子情報通信学会 D-I, VolJ75-D-I, pp704-713,1992.

[平山 93] 平山 勝敏, 山田 誠二, 豊田 順一: 山登り法を用いた分散制約充足における組織化, 第7回人工知能学会全国大会論文集, pp269-272, 1993.

[Minton 90] S.Minton, MarkD.Johnston, Andrew B.Philips and Philip Larid : Solving Large-Scale Constraint Satistaction and Scheduling Problems Using a Heuristic Repair Method, In *Proceedings of AAAI-90*, pp17-25, 1990.

[Morris 93] Paul Morris : The Breakout Method For Escaping From Local Minima, In *Proceedings of AAAI-93*, pp40-45, 1993.

[Yokoo 94] Makoto Yokoo : Weak - commitment Search for Solving Constraint Satisfaction Problems, In *Proceedings of AAAI-94*, pp313-318, 1994.

[石田 92] 石田 亨, 桑原 和宏 : 分散人工知能 (1): 協調問題解決, 人工知能学会誌, Vol.7, No.6, pp945-954,1992.

[Nadel 88] Bernard A. Nadel : TREE SEARCH and ARC CONSISTENCY in CONSTRAINT SATISFACTION ALGORITHM, Search in Artificial Intelligence, pp287-pp342, 1988.