3H－05

エージェント指向に基づいた反復改善型分散制約充足アルゴリズムの実環境における実装と評価

成 孝徳　石本 純一　栗原 正仁
北海道工業大学

1 はじめに

制約充足問題は、人工知能の分野において注目されている基礎的でかつ重要なクラスの問題の一つであり、また分散人工知能の分野においては、制約充足問題の変数と制約が複数の自律したエージェントに分散された分散制約充足問題として定義される。

本稿では、文献[1]で提案されている分散制約充足問題の解法の一つである反復改善型分散制約充足アルゴリズム(以下、分散 breakout)を、分散オブジェクト環境を実現した HORB で実装する。文献[1]ではシミュレーション実験で評価を行い、オーバーヘッドを考慮していなかったが、本研究では実環境で分散 breakout の性能評価を行うことを目的とする。

2 分散制約充足問題

制約充足問題は一般に次のように定義される。n 個の変数、$x_1, x_2, \cdots, x_n$ と、それぞれの変数が値を取る有限で離散的な領域 $D_1, D_2, \cdots, D_n$、および制約の集合が存在する。本稿では制約は述語によって内包的に定義されるとする。すなわち、制約 $p_k(x_{k1}, x_{k2}, \cdots, x_{kj})$は、直積 $D_{k1}, D_{k2}, \cdots, D_{kj}$ に対して定義され、これらの変数の値が互いに整合の取れている場合に真となる。制約充足問題の解を求めることは、すべての制約を満足する変数の値の組を求めることである。制約充足問題は一般に NP 完全であり、最悪の場合の計算量は変数の個数に対して指数的となることが予想される。しかしながら、探索を効率化し平均的な計算を小さくするための様々なヒューリスティックス、探索手法が提案されており、大規模で解が少数しか存在しない難しい問題においても、現実的な時間内で解を得ることが可能となっている。

分散制約充足問題とは、この制約充足問題の変数が複数のエージェントに分散された問題である。

3 分散 breakout アルゴリズム

分散 breakout アルゴリズムは文献[1]で詳細に解説されているため、ここではこのアルゴリズムを簡潔に説明する。エージェントは一つの変数、それに関する制約を持ち、他のエージェントと協調しながらエージェント全体の制約を満足させていく。初期の段階ではいくつかの制約を満足しない不完全な解を構成しているが、その不完全な解を局所的な変更で改善していくことにより、完全な解を得る。個々の制約に関して重みが定義され（初期値は１）、違反している制約の重みの和を不完全な解の評価値とする。この評価値を減少させるように、個々のエージェントの値を変更していく。そして準局所最適†に陥った場合、その状態で違反している制約に対して重みを１増加させる。このように評価関数を変更することにより局所最適から脱出し、全体を満足する値の割当を見つける。

4 実装

4.1 実環境実装とシミュレーションとの相違

各エージェントは WAIT_OK、WAIT_IMPROVE といった二つのモードのいずれかをとる。シミュレーションではシミュレータによりこれらの同期がとられ常に全エージェントが同じモードになっているが、実環境実装ではそのような同期はとらない。したがって、WAIT_OK モードのエージェントに IMPROVE メッセージが到着することを考慮して実装してある。このことはソフトウェアの実装を複雑化しているが、同期のためのメッセージを流さず通信回線の空き時間を有効に利用して随時メッセージを流せるので通信量はたとえシミュレーションと同じでも、実際には通信時間の削減に寄与しうる。

またシミュレーションでは、解が求まったと判定したエージェントは終了（停止）するが、実環境実装ではそのようなエージェントにも種々のメッセージが到着するので終了させるわけにはいかない。

4.2 アルゴリズムの実装

Agent はその値を value、アクセスしてきたエージェント数を counter、各エージェントの値の表を

Implementation of Distributed Breakout Algorithm in Agent-Oriented Programming And Its Evaluation on Real Environment
Takanori Nari, Junich Ishimoto, Masahito Kurihara
Hokkaido Institute of Technology

† エージェント a_i からみて、a_i の制約違反が存在し、a_i 及び a_i のすべての近接エージェントで可能な改善量が 0 の状態

agentView、近接のエージェント間の制約とその重みを arcWeight として所有する。

エージェント間で行われるメッセージには二つある。一つは ok メッセージとしてオブジェクト OkMessage であり、これは現在の値を知らせるものである。もう一つは improve メッセージとしてオブジェクト ImproveMessage であり、これは改善可能な量を知らせるものである。これらのメッセージは receive (OkMessage)、receive (ImproveMessage) として AgentCommunication インターフェースにより宣言される。またエージェントの状態に応じてこれらのメッセージを処理するフラグとして ok メッセージを処理する WAIT_OK、improve メッセージを処理する WAIT_IMPROVE がある。(fig.1)

4.2.1 エージェント通信

エージェント間通信は受取側のインターフェースとして次のメソッドで行われ、

i) void receive(OkMessage)

ii) void receive(ImproveMessage)

メッセージオブジェクトは以下のようになる。

a) OkMessage(送信エージェント名、値)

b) ImproveMessage(送信エージェント名、改善量、評価値、終了判定カウンタ値)

OkMessage は現在の値を知らせるものであり、送信主の名前と問題の値をもつ。ImproveMessage は改善可能な量を知らせるものであり、送信主の名前、改善量は評価値の可能な改善値、評価値は局所的なエージェント間の評価、終了判定カウンタ値は局所的に制約を満足するエージェントの数である。これらのメッセージのやり取りは近傍のエージェント間で行われ、終了判定カウンタ値が上限に達すると全体の制約を満足する割当が見つかったとみなす。

4.2.2 エージェントの状態

receive()によって受け取ったメッセージは OkMessage なら okStream に、ImproveMessage なら improveStream に保存され、監視スレッドがどのメッセージを待っているのか、メッセージは届いているかどうか状態をみて、以下の処理を行う。

i) OkMessage 待ち (WAIT_OK)

メッセージが届いていることを確認すると、counter の値を 1 増やし、agentView に送られてきたエージェント名、値を記録する。そして、近接するすべてのエージェントからの OkMessage を処理すると、ImproveMessage を送り返し、counter を 0 にリセットする。

ii) ImproveMessage 待ち (WAIT_IMPROVE)

メッセージが届いていることを確認すると、counter の値を 1 増やし、自分の終了判定カウンタに受信終了判定カウンタをセットする。受信改善量と自分の改善量を比較し、エージェント値の変更権、準局所最適かどうかを判定する。また、受信評価値から制約充足を判定する。そして、近接するすべてのエージェントからの ImproveMessage を処理すると、OkMessage を送り返し、counter をリセットする。

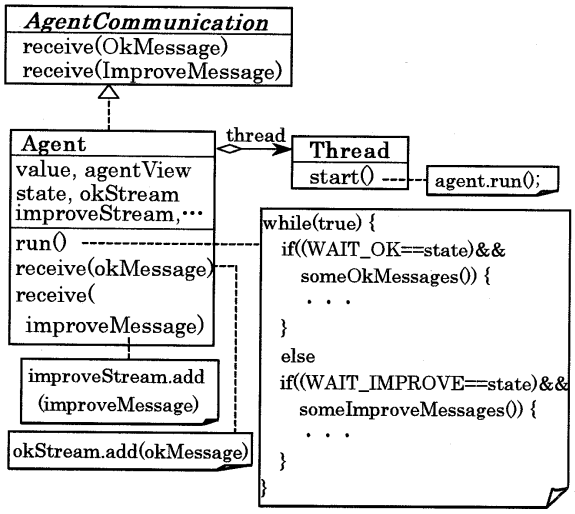

fig. 1 クラス図

5 おわりに

本稿では HORB を用いてエージェント指向プログラミングを心がけ、分散 breakout を実環境で実装した。今後は本実装を評価するシステムを構築し、シミュレーション実験と実環境実験の比較・検討を行う予定である。

参考文献

[1] 横尾真、平山勝敏「分散 breakout：反復改善型分散制約充足アルゴリズム」, 情報処理学会論文誌 Vol.39 No.6 pp.1889-1897 (1998)

[2] 横尾真ほか「分散制約充足による分散協調問題解決の定式化とその解法」,電子情報通信学会 D-I Vol.J75-D-I NO.8 pp.704-713 (1992.8)