不完全論証の完全論証の補強による協調的議論エージェントシステムの実現

武田 豊[1] 若木 利子[2]

芝浦工業大学 システム工学部*

1 はじめに

分散コンピュータネットワーク上のマルチエージェントに人間が行う議論をシミュレートさせる先行研究のアプローチに対話的証明論の適用 [2] があり, そこではエージェントは根拠付き意見 (完全論証) の提出のみ認められていた. しかし, 現実の人間の世界では根拠が不十分な意見 (不完全論証) を提出しても, 提案者と同じ目的を持つ人達が, 自分の知識で提案者の意見を補い, 根拠付き意見にして提案者の論争に協力することがある. このような協調を実現する一つの提案として文献 [1] の先行研究がある. この先行研究のアプローチでは不完全論証を不完全論証で補強することも許され, それのシーケンスの結果, 計算時間上の問題が予想される. 本研究では効率と人間らしい協調を配慮して不完全論証を完全論証のみで補強を行う方法を採用する. また各エージェントに, 議論における味方となるエージェント達を表現した協調エージェントリストを持たせることにより, 現実の人間の議論における敵, 味方と認識する心理的情報を表現した. このような議論の形態を不完全論証の完全論証の補強による協調的議論と称し, この機能を [2] の対話証明論に組み込む方法の提案と, それに基づくシステムの実装, 評価を行った.

2 システム概要

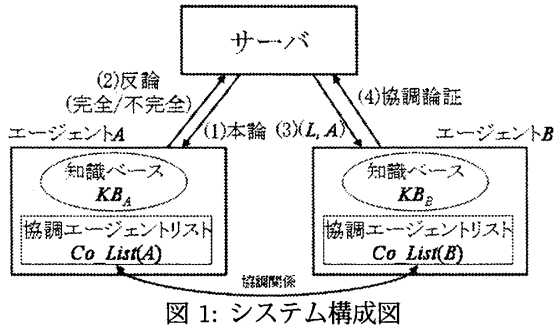

図 1: システム構成図

各エージェント A_i は ELP で記述された知識ベース KB_i と, 当該エージェントが協調する相手を明示した協調エージェントリスト $Co\_List(A_i)$ を持つ. サーバはマルチエージェントを介して対話的証明論に基づく対話木の構築を行う. エージェントが不完全論証による反論を行った場合 (図 1. (2)) は, サーバは不完全論証を完全論証に補強する協調の処理 ((3), (4)) を行う.

3 議論フレームワーク

人間が日常的に行う議論の形式化を以下に示す. 人間の持つ知識を表現する言語として拡張論理プログラムを用い, 人間の意見, 意見のぶつかり合い, 議論に勝つことをそれぞれ形式化したものに論証, 攻撃関係, 対話的証明論がある.

Cooperative argument-based systems completing imcomplete arguments with complete arguments

[1]Yutaka Takeda, [2]Toshiko Wakaki

*Shibaura Institute of Technology

定義 1 (拡張論理プログラム) リテラル L はアトム A またはその論理否定 $\neg A$ である. $not\ L$ を NAF リテラルと称する. 拡張論理プログラム (ELP) とは次のような形をしたルールの集合である.

$$r : L_0 \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_{m+l}\ (m, l \geq 0)$$

L_i はリテラルである. ルール r の頭部 L_0 を $head(r)$, $\{L_1, \ldots, L_m\}$ を $body^+(r)$, $\{not\ L_{m+1}, \ldots, not\ L_{m+l}\}$ を $body^-(r)$ で表す.

定義 2 (論証) ELP KB の論証とはルール $r_i \in KB\ (1 \leq i \leq k)$ の有限系列 $Arg = [r_1; \ldots; r_k]$ である. 但し, $r_i (1 \leq i < k)$ の本体にあるすべてのリテラル L_j に対し L_j が r_s の頭部であるような $s > i$ が存在する. 論証 Arg の全ての結論と全ての仮定の集合をそれぞれ $conc(Arg)$, $assm(Arg)$ で表す. 即ち

$$conc(Arg) = \{L \mid L \in head(r)\ for\ r \in Arg\}$$

$$assm(Arg) = \{not\ L \mid not\ L \in body^-(r)\ for\ r \in Arg\}$$

また論証 $Arg = [r_1; \ldots; r_k]$ に対して,

$body^+(r_k) = \phi$ ならば, 完全論証

$body^+(r_k) \neq \phi$ ならば, 不完全論証 [1] と定義する.

定義 3 (マルチエージェントの論証集合) エージェント $A_i (1 \leq i \leq n)$ の知識ベース ELP KB_i から構成される完全及び不完全論証の全ての論証集合を $Args_{KB_i}$ とする. この時, マルチエージェント $MAS = \{A_1, \ldots, A_n\}$ の論証集合 $Args_{MAS}$ を $Args_{KB_1} \cup \cdots \cup Args_{KB_n}$ で定義する.

定義 4 (攻撃関係) マルチエージェントの攻撃関係 R ($R \subseteq Args_{MAS}{}^2$) の種類には以下の $rebut$, $undercut$ の他, $attack$, $defeat$ (略して r, u, a, d) 等がある. 所与の R について $(Arg_1, Arg_2) \in R$ であるとき, "Arg_1 は Arg_2 を R で攻撃をする"と称する.

1. $(Arg_1, Arg_2) \in r$ iff $L \in conc(Arg_1)$ かつ $\neg L \in conc(Arg_2)$
2. $(Arg_1, Arg_2) \in u$ iff $L \in conc(Arg_1)$ かつ $not\ L \in assm(Arg_2)$

その他の攻撃関係の詳細は [2] を参照.

定義 5 (議論フレームワーク) マルチエージェント MAS の論証集合 $Args_{MAS}$ から構成される議論フレームワーク AF を, $AF \stackrel{def}{=} (Args_{MAS}, R)$ で定義する.

4 協調的議論エージェントシステム

まず完全論証のみを扱った対話的証明論を示す.

定義 6 (対話的証明論)[2] x, y を攻撃関係とする. $Args_{MAS}$ を完全論証の集合とする. x/y-対話は提議 $move_i = (Player_i, Arg_i)(i > 0)$ の空でない有限列である. 但し, $Player_i \in \{P, O\}$, $Arg_i \in Args_{MAS}$ かつ

1. iが奇数ならば $Player_i = P$ であり, iが偶数ならば $Player_i = O$
2. $Player_i = Player_j = P$ かつ $i \neq j$ ならば $Arg_i \neq Arg_j$
3. $Player_i = P$ かつ $i > 1$ ならば Arg_i は $(Arg_i, Arg_{i-1}) \in y$ であるような論証
4. $Player_i = O$ ならば $(Arg_i, Arg_{i-1}) \in x$

x/y-対話木とはすべての根から葉までの経路が x/y-対話の提議の木である．対話木の全ての葉が P であるとき，"対話木の根の論証が正当化された"という．

定義 7 (協調論証) $Arg = [r_1; \ldots; r_k]$ を不完全論証とする．$L \in body^+(r_k)$ なる任意のリテラル L に対し，$L \in conc(Arg_L)$ なる完全論証 Arg_L を L に関する協調論証と称する．なおこのようなリテラル L を"根拠無しリテラル"と称す．

定義 6 の対話的証明論を不完全論証も扱えるよう拡張した協調機能付き対話的証明手続きを以下に示す．不完全論証が存在しない場合は，step0〜2 のみ実行され，存在する場合は step3〜5 の手順が追加で実行される．

協調機能付き対話的証明手続き (概要)

step0 サーバはユーザから入力された議題に対する論証*をエージェントから求めて，P の本論とし，これを対話木の根とする．
step1 サーバは，本論 Arg をエージェントに送り定義 6 の $Player$ (P または O) に応じた反論 Arg' を要求する．(図 1. (1))
step2 サーバがエージェントより反論を受け取る．もし反論が完全論証ならばその反論を対話木に追加をし，goto step1. (図 1. (2))
step3 不完全論証 Arg' で反論したエージェントを A とし，Arg' の根拠無しリテラル L に対して，サーバは $(L,\ A)$ を全エージェントに送り協調を促す．(図 1. (3))
step4 $(L,\ A)$ を受理したエージェント B は，$A \in Co\_List(B)$ かつ L に関する協調論証 Arg_L を構成できれば，Arg_L をサーバに送信する．(図 1. (4))
step5 A の不完全論証 Arg' の全ての根拠無しリテラル L (*i.e.* $L \in body^+(r_k)$) に対する協調論証 Arg_L を受理したサーバは，A の不完全論証 Arg' を $Arg' \bigcup \cup_{L \in body^+(r_k)} Arg_L$ なる完全論証に補強して，これを対話木に追加．goto step1

5 実装と評価

協調的議論エージェントシステムは Java を用いて実装した．対話木の構築にはスタックを用い，step2 でサーバが受理した反論をキューに格納する．キュー内に不完全論証が存在した場合，step5 でキュー内で補強論証と差し替えて，対話木に追加する．

ネットオークションにおけるトラブル事例など現実の人間の世界における問題を用いて，本システムが正しく計算していることを検証した．以下の例で構築したシステムの説明を行う．

例 1 (プロジェクトは終わるか) あるプロジェクトにおいてリーダ，部下 (次郎)，部品の販売員をそれぞれエージェント *leader, ziro, salesperson* (L, Z, S と略記) とする．各エージェントは知識ベース KB_L, KB_Z, KB_S と協調エージェントリスト $Co\_List(L), Co\_List(Z), Co\_List(S)$ を持つ．

各エージェントの知識ベース

$$KB_L = \{finish(project) \leftarrow work(taro), arrive(parts).\ work(taro) \leftarrow\ \}$$
$$KB_Z = \{\neg finish(project) \leftarrow not\ arrive(parts)\}$$
$$KB_S = \{arrive(parts) \leftarrow\ \}$$

協調エージェントリスト

$$Co\_list(L) = [ziro, taro]$$
$$Co\_list(Z) = [\ ]$$
$$Co\_list(S) = [leader]$$

*議題に対する論証が不完全論証の場合は，step3〜5 が実行され完全論証に補強される．

それぞれの論証集合は以下のようになる

$$Args_{KB_L} = \{Arg_1 = [finish(project) \leftarrow work(taro),\ arrive(parts);\ work(taro) \leftarrow\ ]$$
$$Arg_2 = [work(taro) \leftarrow\ ]\}$$
$$Args_{KB_Z} = \{Arg_3 = [\neg finish(project) \leftarrow not\ arrive(parts)]\}$$
$$Args_{KBS} = \{Arg_4 = [arrive(parts) \leftarrow\ ]\}$$

Arg_2, Arg_3, Arg_4 は完全論証であり，Arg_1 は不完全論証である．

議題 $\neg finish(project)$ で議論を行う．サーバは各エージェントに議題 $\neg finish(project)$ を送信し，議題に対する論証を要求する．これに対してエージェント *ziro* が論証 Arg_3 をサーバに送信する．*step1* よりサーバは論証 Arg_3 を攻撃する論証を各エージェントに求める．

(i) 協調機能が無い場合

エージェント *leader* の論証 Arg_1 が論証 Arg_3 に対して反論可能だが，不完全論証であるので反論することができず，$\neg finish(project)$ (即ち，プロジェクトは終わらない) が正当化されるという結果を得る．

(ii) 協調機能が有る場合

エージェント *leader* が不完全論証 Arg_1 で反論する．*step2* よりこの反論はリテラル $arrive(parts)$ に関して不完全であるので *step3* よりサーバは各エージェントに対して $(arrive(parts),\ leader)$ を送信し協調を促す．*step4* よりエージェント *salesperson* が $leader \in Co\_List(S)$ かつ $arrive(parts)$ を結論に持つ協調論証 $Arg_4 = [arrive(parts) \leftarrow\ ]$ を構成できるので，協調論証をサーバに送信する．*step5* より協調論証を受理したサーバは不完全論証 Arg_1 を $Arg_1' = [finish(project) \leftarrow work(taro), arrive(parts);\ work(taro) \leftarrow\ ;\ arrive(parts) \leftarrow\ ]$ なる完全論証に補強し，Arg_3 に反論可能となる．*step1* よりサーバは Arg_1' を攻撃する論証を各エージェントに要求するが，どのエージェントも反論を作成することができないので，$\neg finish(project)$ (即ち，プロジェクトは終わらない) は正当化されない．

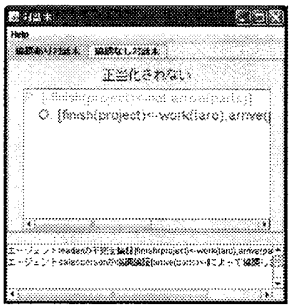

図 2: 協調有り対話木

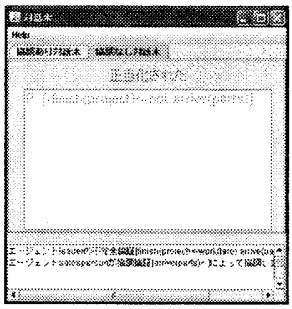
図 3: 協調無し対話木

6 終わりに

論証の補強方法として協調機能だけでなく，対話から他のエージェントの知識を学習し，エージェント固有の知識を増やして論証を補強するといったアプローチも考えられる．このような補強方法も扱える議論システムの拡張と評価が今後の研究課題である．

参考文献

[1] I. de. A. Mora, J.J. Alferes, M. Schroeder: Argumentation and Cooperation for Distributed Extend Logic Programs, Non-monotonic Reasoning Workshop'98, 1998.

[2] R. Schweimeier, M. Schroeder: A Parameterised Hierarchy of Argumentation Semantics for Extended Logic Programming and its Application to the Well-founded Semantics, Theory and Practice of Logic Programming, pages 207-242, 2005.