平成22年度 卒業論文

論文題目

遺伝的Particleによる コミュニティのモデル

神奈川大学 工学部 電子情報フロンティア学科

学籍番号　200702996

伊藤　直人

指導担当者　木下宏揚　教授

目 次

図 目 次

表 目 次

第1章

序論

インターネット上で様々な種類の情報を交換しながら築かれる人間関係によって, 多様なコミュニティが形成される. 例えば, コミュニティの例として企業を挙げると, 企業は企業間や顧客との多様でダイナミックな関係の中でビジネスするようになってきた. そのような現代において, 自分のコミュニティの価値を守りつつも, 他のコミュニティの価値を取り入れて, 常に活動を続ける社会が望まれる. そのような社会を実現させるためには, 情報漏えいをいかに防いでいくかが課題となる. そこで, コミュニティからの情報漏えいを制御するシステムを研究する目的で, 情報漏えいは covert channel から生じるという点に着目した. そのシステムを研究していくにあたって, コミュニティ同士の相互作用, 特に「連携」と「競争」を表現することが重要である. そこで, 連携や競争の中で情報を作りだしてゆく相互作用を連想するモデルを提案し, そのアナロジーからこの問題を解いて行く. Boid で連携, 遺伝的プログラムで競争を表現し, 連携や競争の相互作用から, 多様な社会をアナロジーとして記述するための構造を研究する.

第2章

基礎知識

2.1　Boid

Boid とは 1987 年にアメリカのアニメーション・プログラマであるクレイグ・レイノルズによって考案・作製された人工生命シミュレーションプログラムである. Boid というモデル名は, 鳥もどきという意味の言葉である“ bird - android（バード・アンドロイド）”が短くなったことに由来している. Boid の群れを実現させる振る舞いは, ３つの要素からなり, 「衝突の回避」,「群れの中心に 向かう」,「向きをあわせる」といった３つのルールを規定するだけで鳥の群れをシミュレーションすることができる [1][2][7]. このように単純な行動規範をそれぞれの個体が持ち, 全体として複雑な群れ行動が創発する.

これらをまとめると時刻 t での i 番目の Boid の速度ベクトル $[\overrightarrow{v_i}(\mathrm{t})]$ の更新式は次のようになる.

$$\overrightarrow{v_i}(\mathrm{t})=\overrightarrow{v_i}(\mathrm{t\text{-}1})+\overrightarrow{Next_i}(\mathrm{t\text{-}1})+\overrightarrow{G_i}(\mathrm{t\text{-}1})$$

ここで $\overrightarrow{Next_i}(\mathrm{t\text{-}1})$ は個体 i の最も近くの Boid の速度ベクトル, $\overrightarrow{G_i}(\mathrm{t\text{-}1})$ は個体 i から重心へ向かうベクトルである. なお, 慣性で動くことを実現するため, 1 タイムステップ前の速度 $\overrightarrow{v_i}(\mathrm{t\text{-}1})$ を加えている.

それぞれの Boid は自分の視界を有している. 最も近くにいる Boid を探す場合, 自分の視界内の Boid だけを考える. ただし, 群れの重心を計算するのには他の Boid も含めた全体の座標位置を使う [2].

2.1.1 Separation（引き離し）

Separation は, 近くの鳥や物体に近づきすぎたらぶつからないように離れるルールである. 衝突の回避のために,Boid はそれぞれ自分にとっての最適距離を持っている. そして自分の最も近くにいる仲間との間で, この距離を保ちたいと考えて振る舞う. もし,Boid 同士が近づきすぎてしまったら, 前を飛んでいる Boid はスピードを速くし, 後ろを飛んでいる Boid はスピードを遅くする. 障害物, 例えば柱や壁などに対しては, それにぶつからないように方向転換して衝突を避けるようにする [1].

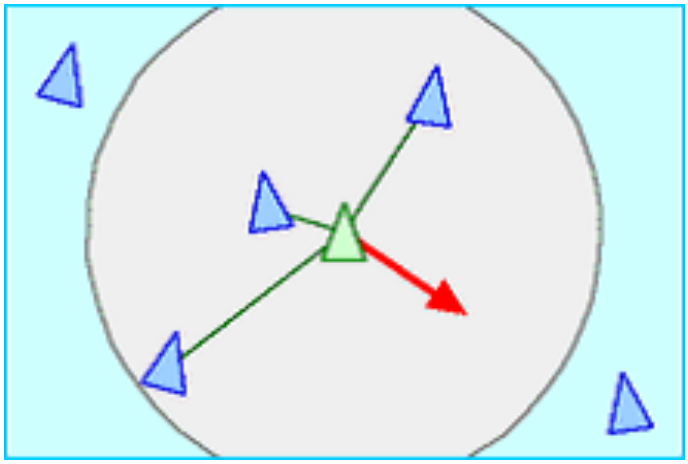

図 2.1 衝突の回避

2.1.2 Alingment（整列）

Alingment は, 近くの鳥たちと飛ぶスピードや方向を合わせようとするルールである. すなわち, 同じ方向にあまり距離を空けないように飛ぶようにする. このルールは, ある一定の距離より遠ざかりすぎてしまったら前を飛んでいる Boid はスピードを遅くし, 後ろを飛んでいる Boid はスピードを速くするようにすることで実現することができる [1].

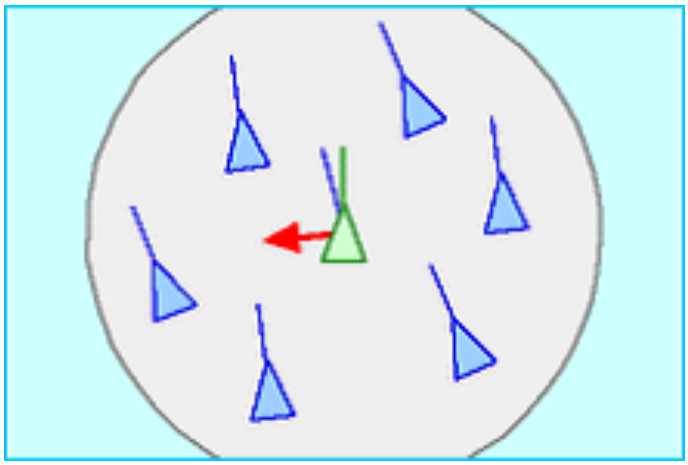

図 2.2 群れの中心に 向かう

2.1.3 Cohesion （結合）

Cohesion は, 鳥たちが多くいる方へ向かって飛ぶルールである. 鳥が多くいる方向というのは, 大ざっぱにいうと群れの中心（重心）方向ということになる. つまりこのルールは,Boid に群れの中心の方向へ飛んでいくことを指示している. この群れの中心は, 全 Boid の位置（座標）の平均として求める [1].

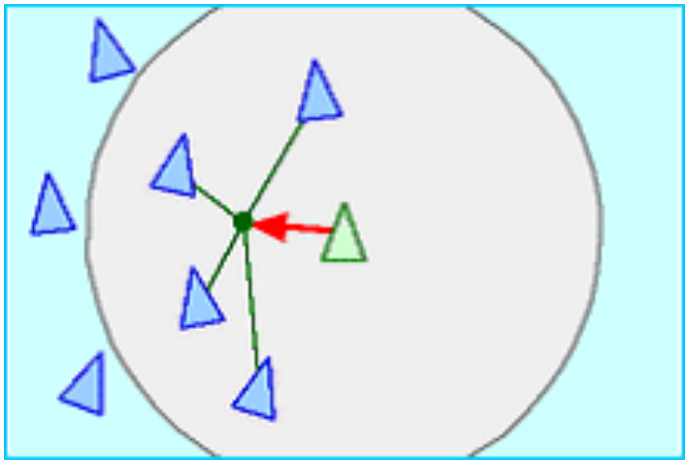

図 2.3 向きをあわせる

2.2 進化論的計算手法

進化論的計算は, 生物の進化のメカニズムをまねてデータ構造を変形, 合成, 選択する方法であり, その代表例が遺伝的アルゴリズム（genetic algorithms,GA）, 遺伝的プログラミング（genetic programming,GP）である [2].

2.2.1 遺伝的アルゴリズム（GA）

GA で扱う情報

遺伝的アルゴリズム（GA）は, 進化論的な考え方に基づいてデータを操作し, 最適解探索や学習, 推論を行う手法であり, GA で扱うデータ構造は,GTYPE（genotype）とPTYPE（phenotype）の二層構造からなる. GTYPE（遺伝子コードともいい, 細胞内の染色体に相当する）は遺伝子型のアナロジーで,GA のオペレータの操作対象となる. PTYPE は表現型（発現型）であり,GTYPE の環境内での発達に伴う大域的な行動や構造の発現を表す. 環境に応じて PTYPE から適合度（fitness value）が決まり、そのため適合選択は PTYPE に依存する（図 2.4 参照）. 適合度は大きい数値をとるほどよいものとする [3].

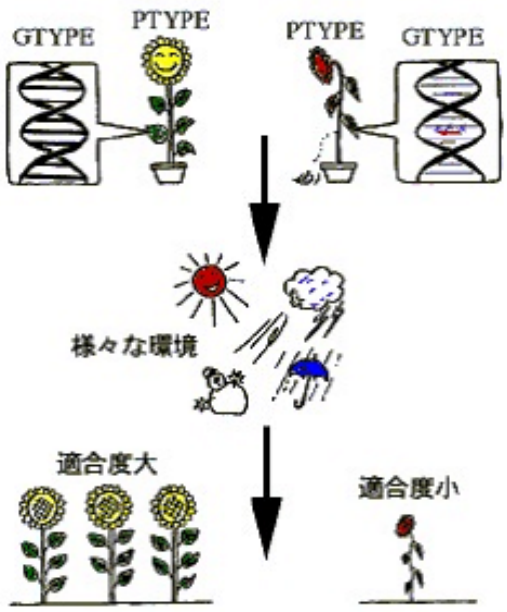

図 2.4 GTYPE と PTYPE

GA の基本的な仕組み

図 2.5 のように, 何匹かの猫が集まって集団を構成しているとする. これを世代 t の猫とし, おのおの GTYPE として遺伝子コードを有し、それが発現した PTYPE に応じて適合度が決まっている. 適合度は図では円の中の数値として示されている. これらの猫は生殖活動（recombination,reproduction）を行い, 次の世代 t+1 の子孫を作り出す. 生殖の際には, 適合度の高いものほどをより多くの子孫つくりやすいように, 適合度の低いものほど死滅しやすいようにする（選択もしくは淘汰）. 図では生殖によって表現型が少し変わっていく様子が模式的に描かれている. この結果, 次の世代 t+1 での各個体の適合度は, 前の世代 t よりも良いことが期待され, 集団全体として見たときの適合度が上がっているはずである. 同様にして, 世代 t+1,t+2,…と繰り返していくと世代が進につれ, 次第に集団全体が良くなっていく, というのが GA の基本的な仕組みである [2][3].

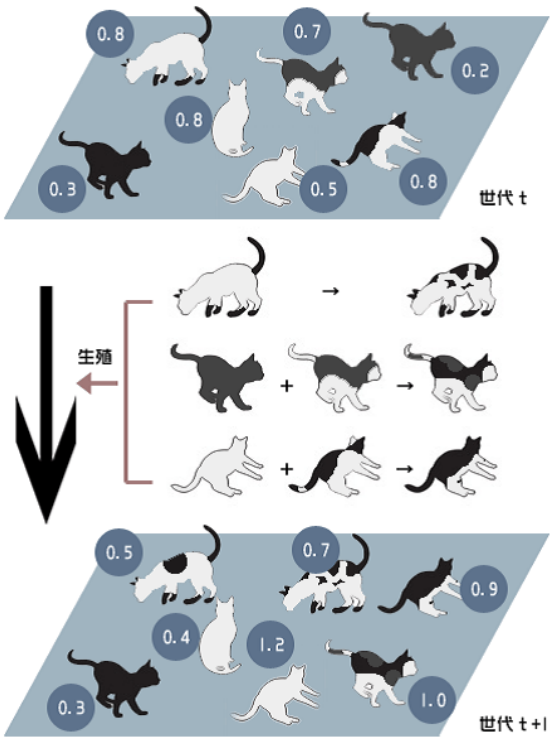

図 2.5　GA の基本的な仕組み

GA のオペレータ

生殖の際には,GTYPE に対して図 2.6,2.7 に示す交叉（crossover）と突然変異（mutation）のオペレータ（遺伝子操作）が適用され, 次の世代の GTYPE を生成する. 図では簡単のために GTYPE を 1 次元配列として表現している. 各オペレータは遺伝子の組換え, 突然変異のアナロジーであり, これらは確率的に適用される [2][3].

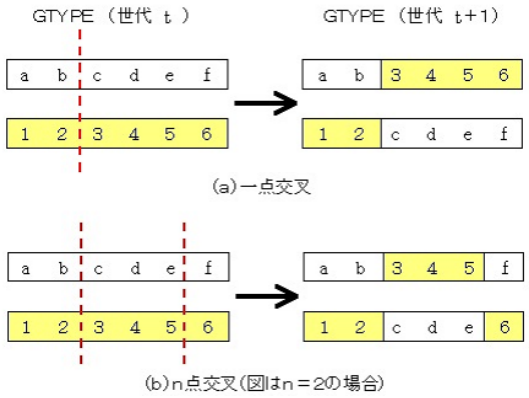

図 2.6 交叉

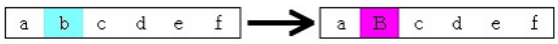

図 2.7 突然変異

GA のアルゴリズム

1. 対象となる問題の解を GTYPE にコーディングする
2. ランダムに初期世代を生成する
3. 現在の集団の各個体について適合度を計算する
4. 終了条件（世代数など）を判定する
5. 現在の集団から適合度の高い個体を選択する
6. 選択された個体に GA オペレータを作用させて, 次の世代の集団を生成する
7. Step3 に戻る

2.2.2　遺伝的プログラミング（GP）

遺伝的プログラミング（GP）は, 遺伝的アルゴリズム（GA）の遺伝子型を木やグラフなどの構造的な表現が扱えるように拡張し, プログラム生成や学習, 推論, 概念形成などに応用することを目指している. 図 2.8 に GP のイメージ（プログラムやロボットの進化する様子）を示した.GP ではプログラムを進化させて目的とするロボットの制御や構造物の設計を試みる.

GP の基本的思想はスタンフォード大学のジョン・コーザらにより提案された.GP は進化論的計算において一分野を確立している. GP の考えを AI に適用し, 学習, 推論, 問題解決を実現する手法を進化論的学習（evolutionary learning）と呼ぶ. これは表現された知識を変換し, 選択淘汰により適切な解を残していく適合的な学習方法である. 進化論的学習はクラシファイア・システムなどに代表される GBML（genetic-based machine learning, 進化型手法による機械学習）とも多くの共通点を有する [2][3].

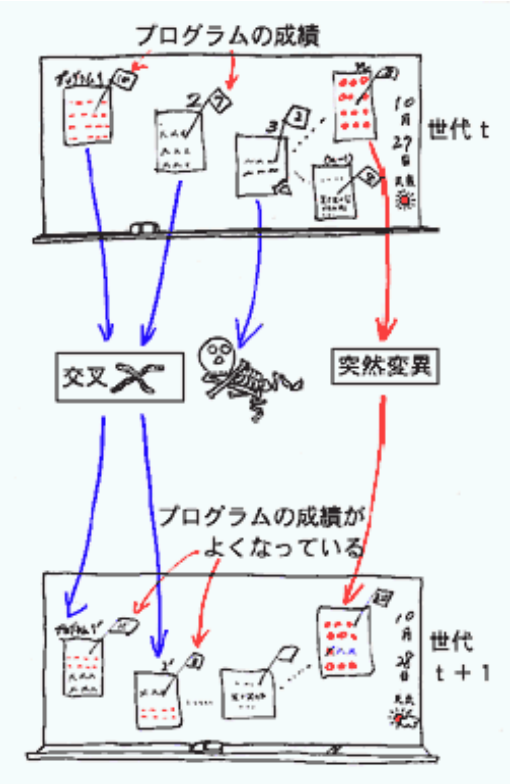

図 2.8　GP のイメージ

GP の基本

GP では, グラフ構造（特に木構造）を扱えるように GA の手法を拡張する. 一般的に木構造は LISP の S 式 で記述できるため,GP では遺伝子型として LISP のプログラムを扱うことが多い. 更に木構造に対する GP オペレータとして次のようなものを用意する.

- 突然変異 … 部分木の変更
- 逆位 … 兄弟木の並び替え
- 交叉 … 部分木の取り替え

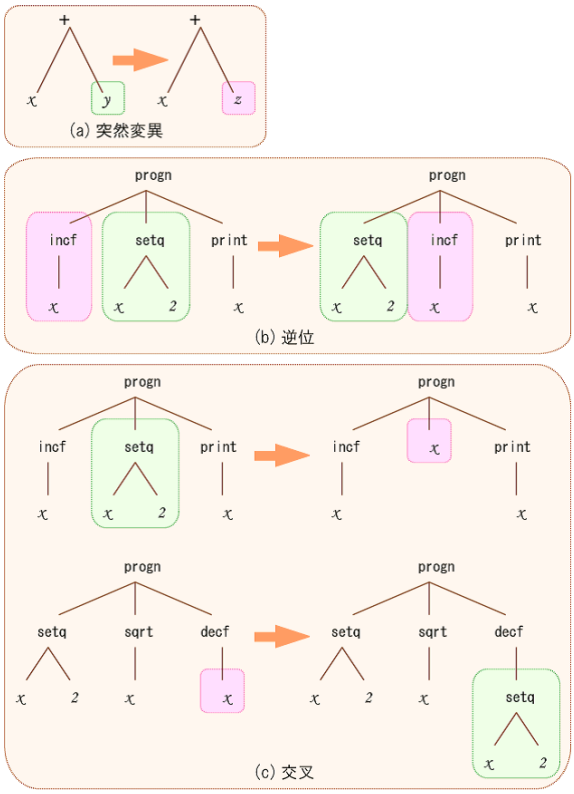

図 2.9 木構造

後は GA と同様に, 選択淘汰, 生殖を繰り返す. そうすれば, 図のオペレータにより少しずつプログラムの構造が変化し, より適した（賢い）プログラムが残っていき, 最終的に目的の（最適な）プログラムが探索することができる [3].

2.3 covert channelの定義

アクセス行列に於いて，subject，object，パミッションをアクセストリプルと定義し，そのアクセストリプル間で起きる不正な情報の経路covert channelを定義する．

【定義】covert channel アクセス行列において，アクセス禁止のパミッションに矛盾する情報フローをcovert channelと呼ぶ．

即ち，
$subject S_i(i = 1, 2, \cdots), S_j(j = 1, 2, \cdots)$, 但し $i \neq j$
$object O_n(n = 1, 2, \cdots), O_n(m = 1, 2, \cdots)$, 但し $n \neq m$
パミッション $P\{RW, \neg RW, \neg WR, \neg(RW)\}$, 但しR(READ),W(WRITE)
とする時，アクセストリプル $\langle S_i, O_n, \mathrm{P} \rangle$ について，
If $\langle S_i, O_n, \neg R \rangle$,
And if $\langle S_j, O_n, R \rangle \wedge \langle S_j, O_m, W \rangle \wedge \langle S_i, O_m, R \rangle$,
Then covert channel $(S_i, S_j, O_m(O_n))$.
と定義する．この定義では，subject S_j はobject O_n と言う名前とその内容をREAD禁止 $\langle S_i, O_n, \neg R \rangle$ であるにも拘わらず，subject S_j がobject O_n の内容をREADし，object O_m にそれをWRITEし，subject S_i が O_m の内容からobject O_n の内容をREADする事（$\langle S_i, O_n, R \rangle$ と表現される）によって，アクセストリプル $\langle S_i, O_n, \neg R \rangle$ と矛盾する結果が生じる事が示される．また，covert channelが引き起こされる時，それに関わるsubjectの数によってcovert channelの程度(フローレベル)を定義する．

【定義】フローレベル covert channelに於いて，あるsubjectからあるsubjectへと情報が流出する回数をフローレベルと定義する．

また，フローレベル k はフローレベル2が基になっている[4]．図2.10にcovert channelとフローレベルの例を示す．

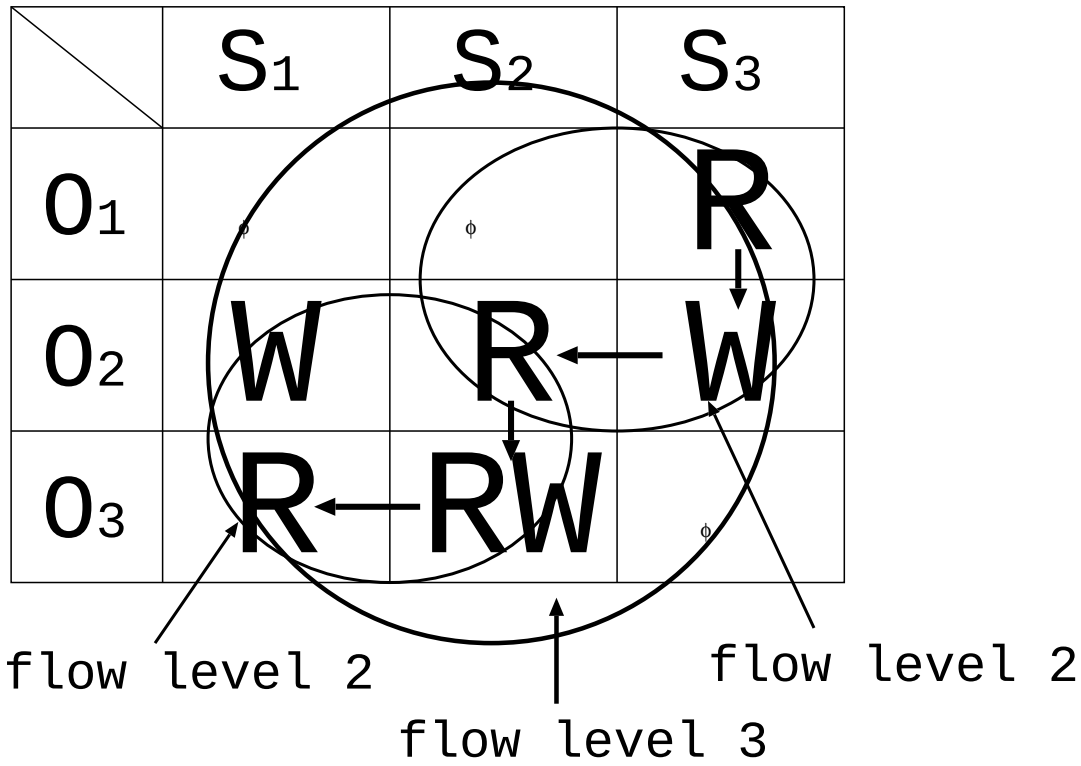

図 2.10 covert channel とフローレベル

2.4 マルチエージェント・シミュレータの種類

2.4.1 artisoc

もっぱら理科系のツールとして開発されていたマルチエージェント・シミュレーションが, 社会科学の方法のひとつとしても有望ではないかと言われはじめてから10年以上たつ. しかしこの手法が今でもあまり浸透していないのは, 社会科学の研究者や学生にとって、まずプログラミング言語から学ぶ必要があり, 要するに敷居が高い手法だったことに起因すると思われる. この事情は, アプリケーションソフトが多数市販されている統計分析の手法と比べると一目瞭然である.

たしかにアメリカを中心に, 人工社会を念頭においた汎用シミュレータはいくつか存在しているが, 使いやすいシミュレータでは単純で定型的なモデルしかつくれない. 複雑で様々なモデルを作れる強力なシミュレータを操作するには高度な専門知識が必要である.

そのような状況で, 数年前に登場したものがWindowsパソコン用のシミュレータKK-MASである. これは構造計画研究所が開発した, 日本語環境のパソコンの上で, プログラミング言語やプログラミング技法を学ばないでも利用できる, しかも汎用性のあるマルチエージェント・シミュレータである. このKK-MASをプロトタイプにしつつ, 社会現象のモデルを作り, シミュレーションを実行することを念頭に置いて開発された第2世代のシミュレータがartisoc[5]である.

2.4.2 Swarm

Swarmは, 生物エージェント（虫など）と無生物エージェント（壁や障害物など）からなる人工世界のボトムアップモデルに基づくシミュレータである. エージェントの動きを簡単な規則で記述し, エージェント間の相互作用による創発現象を観察する.

Swarmの基本的な特徴は次のようになる.

1. シミュレーション・プログラミングを補助するオブジェクト指向型のソフトウェアライブラリーの集まりである.
2. ユーザは自分のプログラムからSwarmライブラリーのオブジェクトを取り込むことでシミュレーションを構築する.
3. SwarmのライブラリーはJavaとObjective-Cに対して提供されている.

Swarm のシミュレーションは

- Model（モデル）:人工世界を抽象的にモデル化しシミュレートする.
- Observer（オブザーバ）:モデルをシミュレーションを観察し表示などを行う.

の 2 つの部分からなる [2].

Swarm のソースコードと Swarm を動作させるために必要なソフト類は, http://www.santafe.edu/projects/swarm/ から入手することができる.

2.4.3 MASON

2 次元・3 次元の空間を使用した物理・社会モデル用のシミュレーションライブラリ. 適切なクラスを継承してモデルを実装することでそれに合わせた GUI が提供される. モデル開発というより, MASON ライブラリを使用したアプリケーション開発という色合いがどうしても強くなってしまう. 分析機能はなく, 動きを見て楽しむというのが主眼のようである [6].

MASON を動作させるために必要なソフト類は, http://www.cs.gmu.edu/ eclab/projects/mason/ から入手することができる.

第3章

提案手法

3.1　アクセス行列で生じる covert channel の問題

図 3.1 の場合, 矢印の流れで subject2 が本来読めないはずの object1 を読めてしまう. covert channel 流出の流れは以下のようになっている. これは間接情報フローとも呼ばれる.

- subject1 が object1 を読み込む.
- subject1 が object2 に object1 で読んだ内容を書き込む.
- subject2 が object2 を読む.
- covert channel により間接的に object1 の内容を読めてしまう.

つまり図 3.1 の場合, 情報漏えいが起きたことになる.

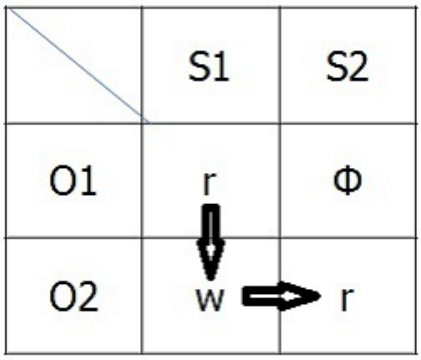

図 3.1 covert channel（間接情報フロー）

（各 S:subject, 各 O:object, r:読みこみ可能, w:書き込み可能, φ:読み込み, 書き込み不可）

このままであれば, このパターンを否定するのが妥当だが, それでは使いづらいため, 工夫を施す必要がある. そもそも covert channel は,S1 と S2 が「連携しているか」,「競争しているか」によって状況が変わるはずである. また, アクセス権を動的に許可, 不許可することによっても covert channel を制御できるだろう.

あるいは下図のようにすれば,covert channel はなくなるだろう.

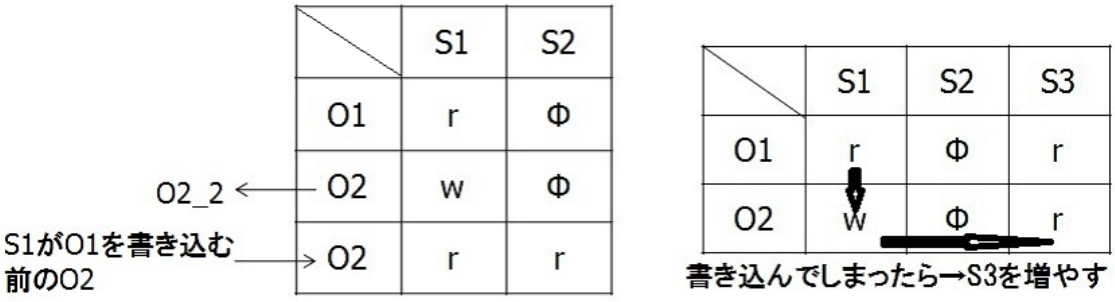

図 3.2 図 3.1 に object, subject を増やす方法

図 3.1 の covert channel は感染（遺伝）に相当する（創発ともいえる）. その影響を受けても生き残るために, 図 3.2 では object 或いは subject を増やした. それは即ち生命システムのアナロジーである. つまり, 進化のシステムともいえる. そこでは, 連携かつ競争がある. そういった意味から, 連携は Boid, 競争は GP でシミュレーションを行っていく.

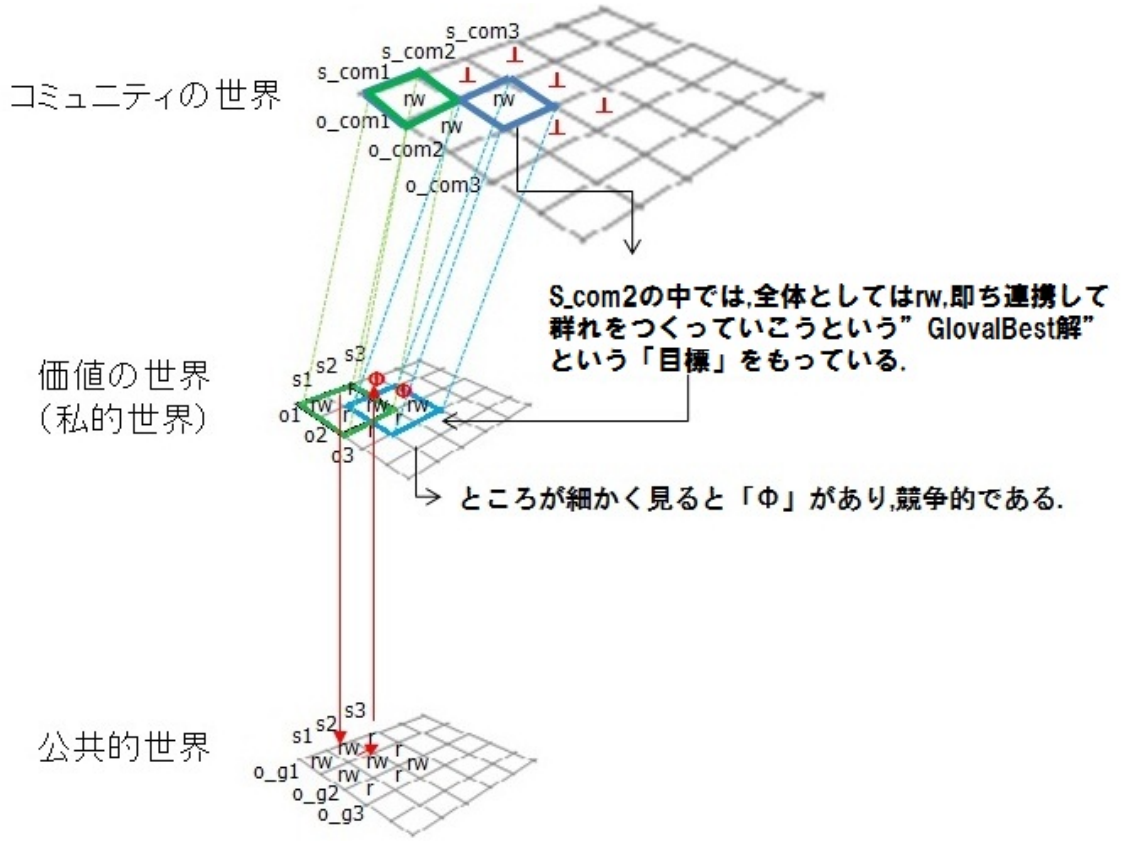

図 3.3　各世界におけるアクセス行列

（Scom$x$:Sx と S$x+1$ からなる subject のコミュニティ, Ocom$x$:Ox と O$x+1$ からなる object のコミュニティ（$x = 1, 2, 3, \cdots$））
進化の過程のメタヒューリスティクスにおいて

- LocalBest …… 局所的な競争に勝つ戦術
- GlobalBest …… 環境に生き残る戦略

図 3.3 にあるように, コミュニティの世界で GlobalBest 解という目標を持っているが, 細かく見ると, そうなっていない. そこで,LocalBest 解を前項の方法によって求めたい. つまり LocalBest の近傍に,GlobalBest 解という目標に向かって「メタヒューリスティクス」に迫る. ゆえに, マルチエージェント・シミュレータを使って研究を行う.

3.2　社会システムにおける公私の価値循環を色で表現する

「色の混色」と「鳥の群れ」を社会システムのアナロジーとして考える. 着目するアナロジーは,「色の混色」という操作と,「鳥の群れ」という相互作用である. 情報の意味を創発することのアナロジー, 競争して意味を作るアナロジーは, 色光の三原色の加法混色と色料の三原色の減法混色を使って表現される.

色彩遺伝子の交叉規則は,「色光三原色の加法混色規則」,「色料三原色の減法混色規則」である.

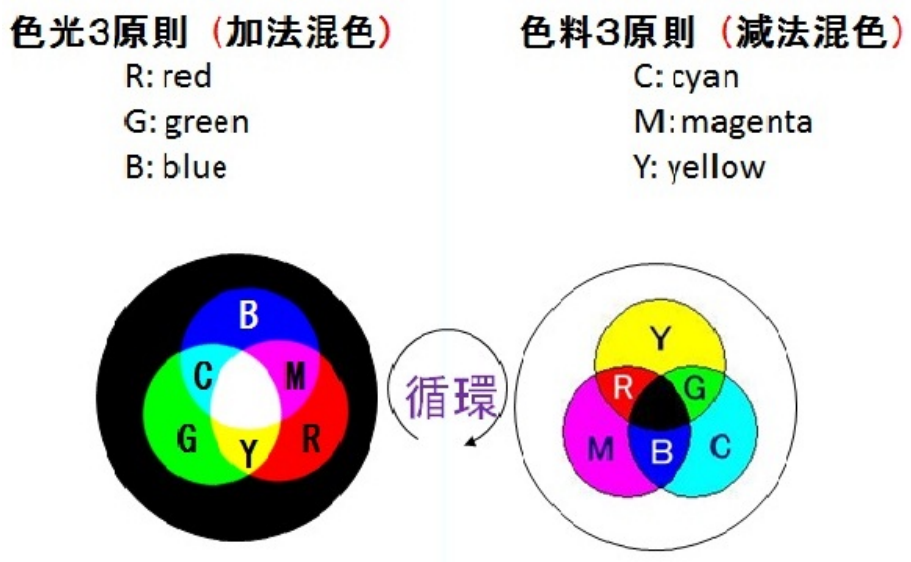

図 3.4　色の混色規則

3.3 方針

図にあるように, 1 つ 1 つの丸は Particle であり, それぞれの Particle は人を表し,Particle の集まりは人が集まるコミュニティを表している. 同じ色は, 考え・利害の一致（連携）, 違う色は, 考え・利害の不一致（競争）である. 各 Particle 間のシンプルな規則によってコミュニティの多様性を維持するメカニズムを究明する.

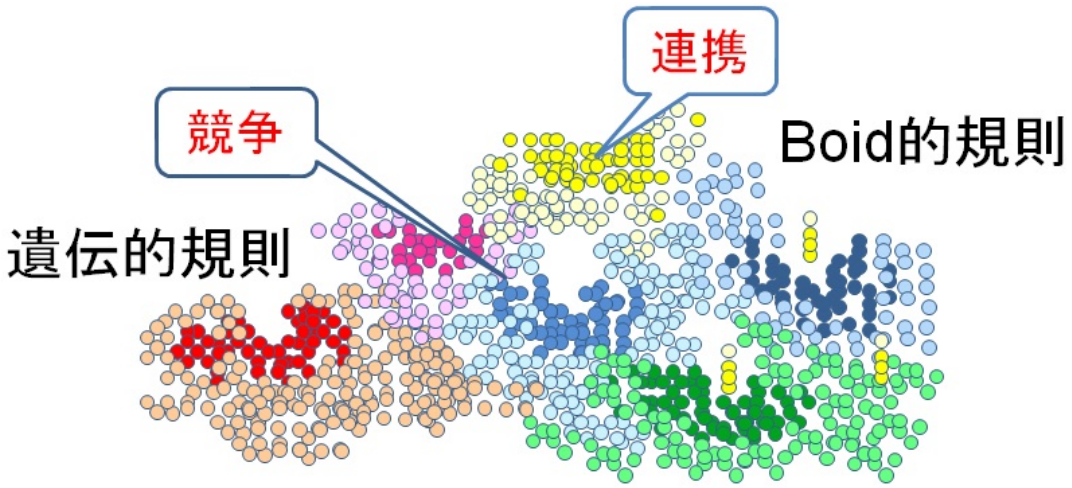

図 3.5 Particle の集まり

3.4 Boids と GP の結合モデル

Boids の構造と遺伝的な生成のシステム (遺伝的プログラミング：GP) を併せ持つエージェントを遺伝的 Particle と呼ぶことにする.

Boid 構造は連携を表現し, Particle は群れを作り動き回るための「衝突の回避」,「群れの中心に向かう」,「向きをあわせる」といった規則を持つ. また,GP の構造は,Particle は新しい Particle が生まれ, 淘汰されて進化する規則を持つ. 遺伝的 Particle は Boid 的規則と遺伝的規則によって増殖し進化しながら群れで動き続ける.

“ 遺伝的 Particle モデル ”は収束解を得るためのモデルではなく, 変化しながらも秩序を保つ, その構造とは何かを究明するものである.

3.5 Boid 的規則（連携）

Boid 的規則, 即ち連携は,

- Particle は互いに衝突を避ける
- 同じ Color の Particle が Color の中心に集まる
- 同じ Color の Particle は重心に向かって移動する

といった規則を持つ.

色彩三原色 RGB の Particle の群れが RGB に関わらずばらばらにある状態から, RGB それぞれの Particle が,R の群れ,G の群れ,B の群れに分離するプログラムの構造を考える.

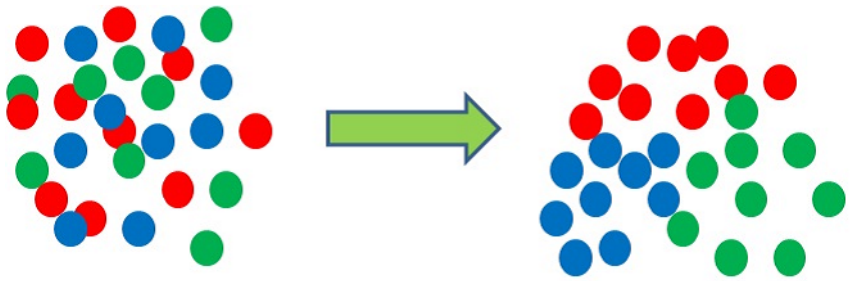

図 3.6 RGB それぞれの Particle が 3 つに分離するイメージ

3.6 遺伝的規則（競争）

遺伝的規則, 即ち競争は,

- Particle の増殖と消滅の構造を埋め込む
- Color の群れ同士に、競争の構造を埋め込む

という規則を持つ.

連携の規則によって色空間に指定されたどこかに RGB 同系色の Particle が集まろうとする. つまり RGB の各重心点，即ち集まる場所は，解ではなく，競争を引き起こす動機である. 群れにとって, 何かが有利になって, 何かが不利にはたらくが, 常に有利, 常に不利という状況になり特定の群れが独り勝ち, 或いは群れの死滅がないようにしたい. そのためには, どのようなはたらきを盛り込めばよいかを考える.

3.7 方法

コミュニティ同士の連携や競争といった相互作用を研究することで, コミュニティの価値の漏洩を防ぎつつ, 常に活動し続ける社会を表現するということが研究の目的である.

それを実現するためには,artisoc というマルチエージェント・シミュレータの様々なサンプルモデルの中から, 次の機能が特に必要なのではないかと考えた.

- 「friendship モデル」でのエージェントを集合させる機能 …… A
- 「infection モデル」での病気が感染する機能 …… B
- 「plankton モデル」での生成と抹消の機能 …… C

friendship モデル

friendship モデルの入力コマンドと実行結果の図を下記に記述する.

```
Agt_Init{  //シミュレーションの最初だけに実行されるルール
My.X = Rnd()*50  //50 × 50 の空間の中で
My.Y = Rnd()*50
My.Direction = Rnd()*360  //ランダムな方角に進む
ClearAgtset(My.friends)  //friends を初期化（中を空に）する
}
Agt_Step{  //毎ステップ実行されるルール
Dim s As Integer  //s という変数を整数型として（As Integer）
Dim one As Agt
Dim temp As Agtset
Dim close As Agtset
Dim neighbor As Agtset
Turn(Rnd()*360)  //とりあえず、好みはランダムに変わろうとする
//自分の好みと似ている人を友人にする
MakeAllAgtSetAroundOwn(neighbor, 3, False)
If CountAgtset(neighbor) > 0 then
   one = GetAgt(neighbor, Int(Rnd()*CountAgtset(neighbor)))
   AddAgt(My.friends, one)  //one を My.friends に追加
   TurnAgt(one)  //one の好みにあわせようとする
   //My.friends の重複をなくす
   DuplicateAgtset(My.friends, temp)
End if
Forward(1)
}
```

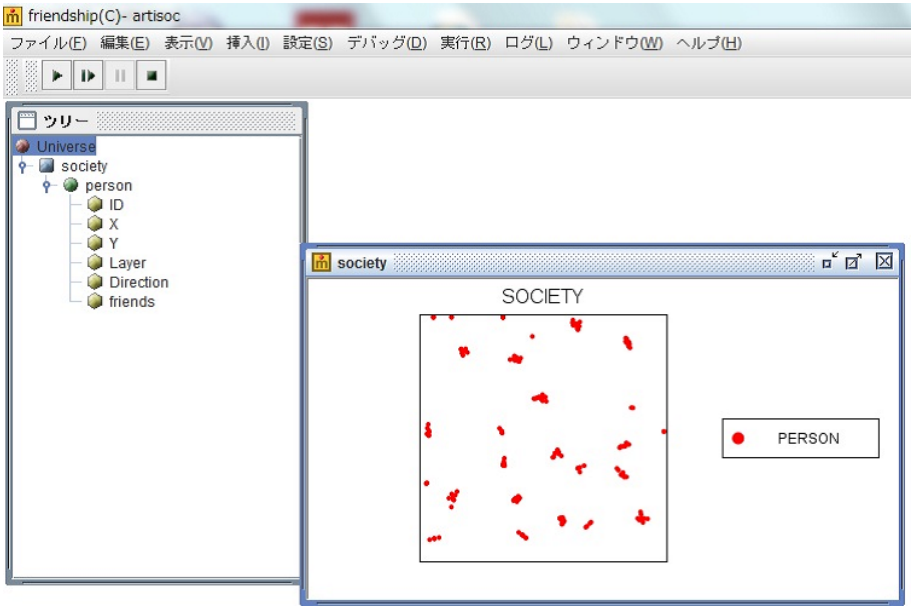

図 3.7 friendship モデルの実行結果

friendship モデルは, 好みの近い者同士が友達になる過程をモデル化したものである. コマンドの大まかな意味は, //の後に示した.

infection モデル

infection モデルの入力コマンドと実行結果の図を下記に記述する.

```
Univ_Init{
Dim i As Integer
Dim people As Agtset
Dim one As Agt
For i = 0 To Universe.pop -1
   one = createAgt(Universe.society.person)
   one.Direction = Rnd()*360
   one.cure = 0
   If Rnd() < Universe.initratio Then
      one.condition = Color_Red
   Else
      one.condition = Color_Cyan
   End if
Next i
MakeAgtSet(people, Universe.society.person)
RandomPutAgtSet(people)
}
Univ_Step_Begin{
}
Univ_Step_End{
Dim people As Agtset
Dim one As Agt
Universe.number = 0
MakeAgtSetSpace(people, Universe.society)
For each one in people
```

```
   If one.condition == Color_Red Then
      Universe.number = Universe.number + 1
   End if
Next one
}
Univ_Finish{
}

Agt_Init{
}

Agt_Step{
Dim neighbor As Agtset
Dim one As Agt
If My.condition == Color_Red Then  //風邪をひいているなら
   My.cure = My.cure + 1
//何もしない
Else  //風邪をひいていないなら
   MakeAllAgtSetAroundOwn(neighbor, 2, False)
   For each one in neighbor  //neighborの中から順次取り出す
      If one.condition == Color_Red Then  //風邪引きなら
         If Rnd() < 0.3 Then
            My.condition = Color_Red
         End if
         End if
      Next one
   End if
         Turn(Rnd()*20 - 10)
         Forward(1)
         If My.cure >= 7 Then  //7ステップで全快
            My.condition = Color_Cyan
            My.cure = 0
End if
}
```

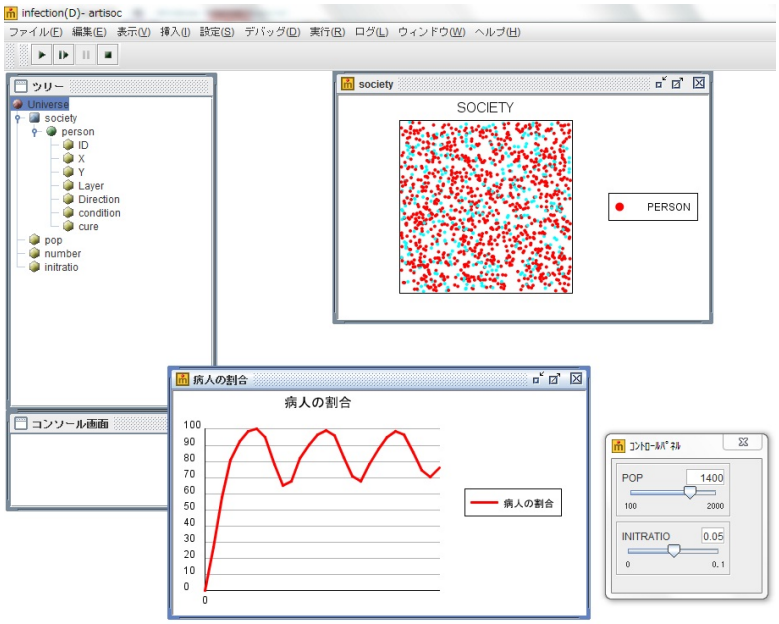

図 3.8 infection モデルの実行結果

infection モデルは, 病気の流行をモデル化したものである. 社会に風邪をひいている人が少数いて, 自分の周囲に風邪をひいている人が多いと, 健康な人も風邪をひく（赤くなる）. しかし, 風邪を引いても 7 ステップたつと健康になる（前項の場合）. //の後にコマンドの意味を示した.

plankton モデル

plankton モデルの入力コマンドと実行結果の図を下記に記述する.

```
Univ_Init{
}
Univ_Step_Begin{
}
Univ_Step_End{
Universe.numphyto = CountAgt(Universe.aquarium.phyto)
Universe.numzoo = CountAgt(Universe.aquarium.zoo)
If Universe.numphyto <= 0 Or Universe.numzoo <= 0 Then
   ExitSimulationMsgLn("Extinct after" & GetCountStep() & "steps")
End if
}
Univ_Finish{
}
//植物プランクトン（phytoplankton）のルール
Agt_Init{
My.X = Rnd()*50
My.Y = Rnd()*50
My.Direction = Rnd()*360
}
Agt_Step{
Dim surround As Agtset
Dim daughter As Agt  //生成されるエージェントの「仮の姿」
Turn(Rnd()*120 - 60)
Forward(0.5)
If Rnd() < 0.05 Then
   daughter = CreateAgt(Universe.aquarium.phyto)
   daughter.X = My.X
   daughter.Y = My.Y
   daughter.Direction = Rnd()*360
End if
MakeOneAgtSetAroundOwn(surround, 1, Universe.aquarium.phyto, False)
If CountAgtSet(surround) >= 4 Then
   DelAgt(My)
End if
}
//動物プランクトン（zooplankton）のルール
Agt_Init{
My.X = Rnd()*50
My.Y = Rnd()*50
My.Direction = Rnd()*360
My.power = (Rnd()*9 - 4)
}
Agt_Step{
Dim surround As Agtset
Dim food As Agt
Dim daughter As Agt
```

```
//動きまわる基本パターン
Turn(Rnd()*60 - 30)
Forward(2)
My.power = My.power - 1
If Rnd()*My.power <= 0.2 Then
   DelAgt(My)
End if
//餌をさがして、あれば食べる
MakeOneAgtSetAroundOwn(surround, 2, Universe.aquarium.phyto, False)
If CountAgtset(surround) > 0 Then
   food = GetAgt(surround, Int(Rnd()*CountAgtset(surround)))
   DelAgt(food)  //食べる
   My.power = My.power + 3
End if
If My.power >= 10 Then
   daughter = CreateAgt(Universe.aquarium.zoo)
   daughter.X = My.X
   daughter.Y = My.Y
   daughter.Direction = Rnd()*360
   daughter.power = 4
   My.power = My.power - 4
End if
}
```

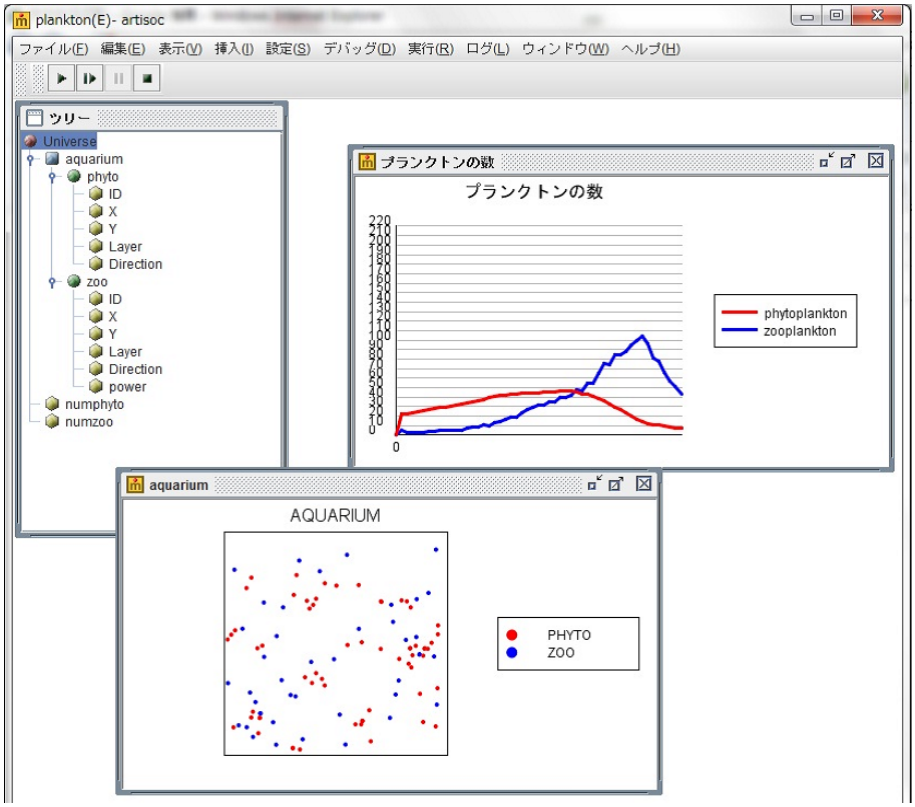

図 3.9 plankton モデルの実行結果

plankton モデルは, プランクトンの食物連鎖をモデル化したものである. 水槽に植物プランクトンと動物プランクトンが浮遊しており, 後者が前者を餌にしている. 植物プランクトンは増殖するが, 増えすぎると環境が劣化して, 自分自身が死滅する. 動物プランクトンは, 餌を食べないでいると体力を消耗してやがて死んでしまうが, たくさん食べると増殖する. このシミュレーションはプランクトンを想定しているが, 本質的には捕食者と被捕食者との相互作用をモデル化したものである.
//の後にコマンドの意味を示した.

3.8 提案する遺伝的 Particle モデル

friendship モデルと infection モデルのコマンドをベースに, 同色の Particle は集まり, 異色の Particle が近くにいると影響を受けて変色するというモデルを作成することを目指す.

そこで赤と青の各 Particle は同色ごとに集まり, 近くにいると青い Particle が赤い Particle から影響を受けて黄色に変色（感染）し, 時間がたつと元に戻るというモデルを作成した. 近くにいる遺伝的 Particle が黄色に変色するという仕組みは「object を read し, 別の object に write する行為」, 即ち covert channel が起きた事実のアナロジーである．つまり,Boid と GP による進化系によって covert channel を分析制御するシステムは，covert channel を完全になくすのではなく，covert channel が起こった後に, 回復するシステムである.

遺伝的 Particle モデルの入力コマンドと実行結果の図を下記に記述する.

```
Univ_Init{
Dim i As Integer
Dim one As Agt
Dim people As Agtset
For i = 0 To Universe.ninzu - 1
   createAgt(Universe.society.particle1)
   one = createAgt(Universe.society.particle2)
      one.cure = 0
   If Rnd() < Universe.initratio Then
      one.condition = Color_Yellow
   Else
      one.condition = Color_Blue
   End if
Next i
MakeAgtsetSpace(people, Universe.society)
RandomPutAgtset(people)
}
Univ_Step_Begin{
}
Univ_Step_End{
Dim p2 As Agtset
Dim one As Agt
Universe.number = 0
MakeAgtset(p2, Universe.society.particle2)
For each one in p2
   If one.condition == Color_Red Then
      Universe.number = Universe.number + 1
   End if
Next one
}
Univ_Finish{
}
```

```
//Particle1 のルール
Agt_Init{
My.X = Rnd()*20
My.Y = Rnd()*20
My.Direction = Rnd()*360
ClearAgtset(My.friends)  //friends を初期化（中を空に）する
}
Agt_Step{
Dim s As Integer
Dim one As Agt
Dim temp As Agtset
Dim close As Agtset
Dim neighbor As Agtset
Turn(Rnd()*360)  //とりあえず、好みはランダムに変わろうとする
//自分の好みと似ている人を友人にする
MakeOneAgtSetAroundOwn(neighbor, Universe.shiya, Universe.society.particle1, False)
If CountAgtset(neighbor) > Universe.nakama then
   one = GetAgt(neighbor, Int(Rnd()*CountAgtset(neighbor)))
   AddAgt(My.friends, one)  //one を My.friends に追加
   TurnAgt(one)  //one の好みにあわせようとする
   //My.friends の重複をなくす
   DuplicateAgtset(My.friends, temp)
End if
//離れた友人の好みに合わせようとする
DuplicateAgtset(temp, My.friends)  //友達全員のコピー
DelAgtset(temp, neighbor)  //好みの近い人を除く
If CountAgtSet(temp) > Universe.nakama Then  //好みの離れた友人がいれば
   one = GetAgt(temp, Int(Rnd()*CountAgtset(temp)))
   TurnAgt(one)  //友人の好みに合わせようとする
Else
//好みの近すぎる友人から離れようとする
   MakeOneAgtSetAroundOwn(close, Universe.shiya, Universe.society.particle1, False)  //
好みが近すぎる人々
   MakeCommonAgtset(temp, close, My.friends)  //好みが近すぎる友達
   If CountAgtSet(temp) > Universe.nakama Then  //好みが近すぎる友人がいれ
ば
      one = GetAgt(temp, Int(Rnd()*CountAgtset(temp)))
      TurnAgt(one)
      Turn(180)  //結局、反対方向を向く
   End if
End if
Forward(1)
}

//Particle2 のルール
Agt_Init{
My.X = Rnd()*20
My.Y = Rnd()*20
My.Direction = Rnd()*360
ClearAgtset(My.friends)  //friends を初期化（中を空に）する
}
Agt_Step{
Dim s As Integer
Dim one As Agt
Dim temp As Agtset
Dim close As Agtset
Dim neighbor As Agtset
//↓たした部分（感染）
If My.condition == Color_Yellow Then    //感染したなら
```

```
   My.cure = My.cure + 1
Else
   MakeOneAgtsetAroundOwn(neighbor, 1, Universe.society.particle2, False)
   For each one in neighbor
      If one.condition == Color_Yellow Then   //感染しているなら
         If Rnd() < Universe.initratio Then
            My.condition = Color_Yellow
         End if
      End if
   Next one
End if
If My.cure >= Universe.suteppu Then   //X ステップで青に戻る
   My.condition = Color_Blue
   My.cure = 0
End if
//↑たした部分（感染）
Turn(Rnd()*360)  //とりあえず、好みはランダムに変わろうとする
//自分の好みと似ている人を友人にする
MakeOneAgtSetAroundOwn(neighbor, Universe.shiya, Universe.society.particle2, False)
If CountAgtset(neighbor) > Universe.nakama then
   one = GetAgt(neighbor, Int(Rnd()*CountAgtset(neighbor)))
   AddAgt(My.friends, one)  //one を My.friends に追加
   TurnAgt(one)  //one の好みにあわせようとする
   //My.friends の重複をなくす
   DuplicateAgtset(My.friends, temp)
End if
//離れた友人の好みに合わせようとする
DuplicateAgtset(temp, My.friends)  //友達全員のコピー
DelAgtset(temp, neighbor)  //好みの近い人を除く
If CountAgtSet(temp) > Universe.nakama Then  //好みの離れた友人がいれば
   one = GetAgt(temp, Int(Rnd()*CountAgtset(temp)))
   TurnAgt(one)  //友人の好みに合わせようとする
Else
//好みの近すぎる友人から離れようとする
   MakeOneAgtSetAroundOwn(close, Universe.shiya, Universe.society.particle2, False)  //
好みが近すぎる人々
   MakeCommonAgtset(temp, close, My.friends)  //好みが近すぎる友達
   If CountAgtSet(temp) > Universe.nakama Then  //好みが近すぎる友人がいれ
ば
      one = GetAgt(temp, Int(Rnd()*CountAgtset(temp)))
      TurnAgt(one)
      Turn(180)  //結局、反対方向を向く
   End if
End if
Forward(1)
}
```

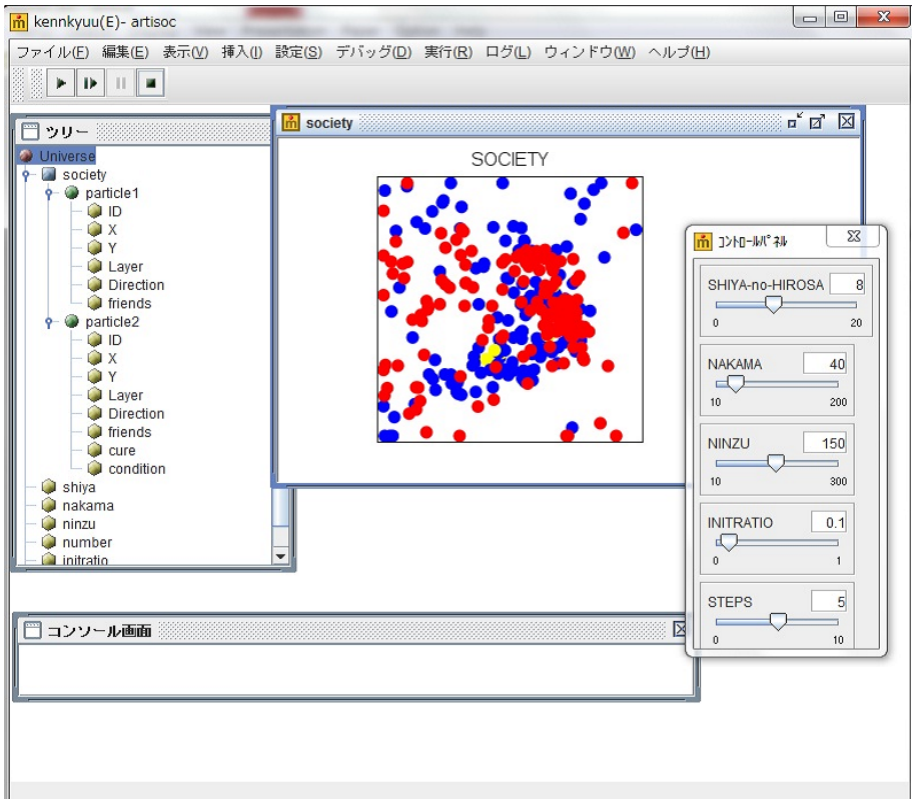

図 3.10 Particle モデルの例

赤と青の Particle の数（NINZU）は 150 人づつ, 感染時間（STEPS）は 5 ステップで固定とし, 視野（SHIYA-no-HIROSA）と仲間の数（NAKAMA）と初期状態の感染率（INITRATIO）のパラメータを変えて, 赤，青の Par が均衡 (等しい数で収束＝最終的に青い Particle が全回復）する場合を集計した. ちなみに視野は 2 刻み, 仲間は 20 刻み, 感染率は 10 % を調べた後に 20 % から 10 % 刻みで調べた. その結果を表にまとめたものを次項に示す.

初期状態の感染率10%	
視野の広さ	仲間の数
2	20
2	40
4	40
2	60
4	60
6	60
2	80
4	80
6	80
8	80
2	100
4	100
6	100
8	100
10	100

初期状態の感染率20%	
視野の広さ	仲間の数
2	10
4	10
6	10
8	10
10	10
2	20
2	40
4	40
2	60
4	60
6	60
2	80
4	80
6	80
8	80
10	80
2	100
4	100
6	100
8	100
10	100

初期状態の感染率100%	
視野の広さ	仲間の数
2	20
4	20
6	20
8	20
10	20
2	40
4	40
6	40
8	40
10	40
2	60
4	60
6	60
8	60
10	60
2	80
4	80
6	80
8	80
10	80
2	100
4	100
6	100
8	100
10	100

図 3.11 赤と青の Particle が等しい数で収束する場合の各パラメータ

上図 3.11 を見てもわかるように, 初期状態の感染率が 10 %,20 %,100 % の場合にのみ, 赤と青の Particle が等しい数で収束した.

つまり,10〜20 % 程度の低い感染率しか持たない黄色い Particle は, 近くに青い Particle がいても, 80〜90 % の確率で感染することができない. もしも感染しても,5 ステップ後には青い Particle に戻るため, 感染率が低めである場合は, いずれは感染している Particle はすべて青に戻ることが予想できる. このことから, 感染率が低いと赤と青の Particle が同じ数に収束しやすいことがわかる. また,10〜20 % の感染率であっても, 毎ステップに感染した Particle が存在するパターンも数多くみられた. そのパターンのほとんどは, 視野が広め, かつ仲間の数が少なめという場合であった. これらは, 各 Particle が仲間を見つけては群れをつくっていた. Particle が群をつくる, 即ち密集す

るということは, 視野内の多くの他の Particle に影響を受ける（感染しやすくなる）ため, 収束しにくい. 逆に言えば,Particle が散らばっていれば, 収束しやすくなるだろう. ちなみに, 図 3.11 ほとんどのパターンがばらばらの状態であった.

また, 初期状態の感染率が 100 % の場合は, すべてのパターンが赤と青に収束するという結果となった. これは, 実行直後の 1 ステップ目の時点ですべての青い Particle が感染しているという意味である. つまり,5 ステップ後に感染している Particle は存在していないため,5 ステップ以降は感染する確率は 0 % となる. この結果は実行する前から容易に予測できる.

第4章

結論

コミュニティ同士の情報漏えいを防ぎ, 常に活動し続ける社会を表現するため, 情報漏えいは covert channel から生じるという点に着目した.

covert channel を制御して情報漏洩を防止する方法として，木下研究室ではアクセス行列のアクセス権限を変更する方法を研究してきた．本研究では，object や subject の数を増やすことで covert channel を阻止する手法に着目した.

object や subject というシステム要素数を増やし，covert channel という脅威に対応してゆくダイナミックなシステムは，システムが進化して環境に適応してゆく様子のアナロジーである，と見做せる．そこで本研究では,covert channel 制御を進化系で制御することを目標として研究を行った. 即ち，生命的なシステムで covert channel 分析制御が可能かどうかを，covert channel の発生メカニズムを抽象化し，マルチエージェント・シミュレータでシミュレーションした．研究したモデルは，Boid 構造と遺伝的構造を持つようにし，コミュニティという「集まる作用」は Boid 構造で, 進化するという「選択と淘汰の増殖作用」は遺伝的プログラミング（GP）で記述した．シミュレーションは，マルチエージェントシミュレータを使い，この 2 つの機能を融合するモデル（遺伝的 Particle モデル）を提案した. 作成したモデルは, 赤と青の Particle がそれぞれ同色ごとに集まり, 近くにいると青い Particle が赤い Particle から影響を受けて黄色に変色（感染）し, 時間がたつと元に戻るというものであり, 遺伝的 Particle が黄色に変色するという仕組みは「object を read し, 別の object に write する行為」, 即ち covert channel が起きた事実のアナロジーである．本研究でシミュレーションした Boid と GP による進化系によって covert channel を分析制御するシステムは,covert channel を完全になくすのではなく, covert channel が起こった場合は, それを発見し, 回復するシステムである．これは生命体が, 完全には病気にならないのではなく, 病気になるとそれを発見して免疫機能によってウィルスを退治する, 或いはウィルスはいつまでも一定限度以下で体内に寄生し続けるが, 生命体自身は「健康体」として振る舞うシステムに類似する. 一連のシミュレーションの結果，covert channel を生命的な遺伝システムで抑止する可能性を見いだせたのではないかと思う．covert channel は変色の仕組みに置き換えられたのであるが，本研究でわざわざ「色彩」を使った理由は，実際の covert channel はファイルの内容の概念的類別や，ファイルにアクセスする subject の役割など，現実が非常に複雑であることを色彩で置き換えようとした

ためである．その試みはまだ道半ばではあるが，一定の成果が得られたと考える．今後は，現実の covert channel 問題と色彩のモデルの対応関係を更に明確にし，モデルから得られるメタヒューリスティクス的な結果を，現実のシステム制御パラメータに置き換える研究に繋がって行けばと思う．

謝辞

本研究を行本研究を行うにあたり，終止熱心にご指導して頂いた木下宏揚教授と鈴木一弘助手，ご多忙の折研究室に足を運び様々な面で有益なご助言をして頂いた森住哲也氏に深く感謝いたします．さらに，公私にわたり良き研究生活送らせて頂いた木下研究室の方々に感謝いたします．

2011 年 2 月
伊藤　直人

参考文献

[1] ”Boid とは”
http://members.jcom.home.ne.jp/ibot/boid.html

[2] 伊庭斉志：”複雑系のシミュレーション - Swarm によるマルチエージェント・システム - (2007)”

[3] ”東京大学工学部電気系工学専攻融合情報学コース伊庭研究室”
http://www.iba.t.u-tokyo.ac.jp/

[4] 小松充史, 木下宏揚：”Covert Channel 分析制御のために推論を導入した情報フィルタに関する研究”

[5] 山影進：”人工社会構築指南 - artisoc によるマルチエージェント・シミュレーション入門 - (2010)”

[6] ”マルチエージェントシミュレータ比較”
http://www.gpgsim.net/gpgsim/comp-mas.html

[7] ”ウィキペディア（Wikipedia)”
http://ja.wikipedia.org/wiki/

[8] ”GA-GP Lab”
http://gagplab.com/gagp.html

質疑応答

Q1:シミュレーションを実行する際の Particle の数はどうなっているのか.(能登教授)

A1:赤い Particle と青い Particle の変化がよくわかるように, 等しい数で設定しました.

Q2:１つ１つの Particle の強さは異なるのか.(能登教授)

A2:今回は, 個々の Particle の能力を無視し, 全ての Particle が同じ能力を持っている体でシミュレーションしました.

Q3:一連の研究の結果, covert channel を防げると言えるのか.(松澤教授)

A3:現段階では, covert channel を防げるとは言えませんが, 今後は covert channel の問題と色彩のモデルの対応関係をより明確にすることで, 現実のシステム制御パラメータに置き換える研究に繋がって行けばと思います.