Tierra 的手法を用いた植物系の進化のシミュレーションとその解析

東京工業大学大学院 情報理工学研究科
数理・計算科学専攻 学籍番号 99M37314
山下 真
指導教官 小島政和教授 戴陽講師

2001.01

.

目 次

第1章　はじめに

人の宇宙においての生存区域を得るために考えられている方法の一つに、テラフォーミング、つまり惑星の地球化というものがある。テラフォーミングの考え方自身は、1961 年のカール・セーガンの金星の環境改造についての論文 'The Planet Venus' [Sagan1961] がその始まりとされ、これ以降多くの科学者が惑星の地球化というとても大きなテーマに取り組んでおり、現在では火星が惑星の地球化の最初の候補として考えられている。

カール・セーガンは [Sagan1961] において、地球上の生命にとってはあまりに高温である金星の大気の温度を下げるために、blue-green algae という藻の一種を最初に金星に送り込み、藻の光合成を利用するという方法を提案している。

しかし、テラフォーミングは 1000 年以上という時間がかかるとされており、そのような長い時間を過ぎたときに、生態系がどのような進化を遂げているのかがはっきりしないという問題点がある。たとえば、どれだけたくさんの種へと進化するかということや、それぞれの個体がどれだけの時間生存できるのかということが、長い時間の進化にどれだけ影響されるのかわからないのである。

そこで、シミュレーションを行うことで進化の方向をある程度推測できないかどうかが重要になる。

本論文では、人工生命の分野で研究されている Tierra を応用して、進化の方向の推測のために植物系の進化のシミュレーションを行う。

進化の方向では、たとえば

- 生存時間の最大化
- 収穫量の最大化
- 貯えられるエネルギーの最大化
- 消費する二酸化炭素の量の最大化

などを含めてさまざまな方向が考えられるが、本論文ではその中から注目すべき方向として

- 種の多様性の最大化

を扱うことにしている。これは、種の多様性のある環境ほど安定性の高い環境になりやすいことによる。

植物系の進化のシミュレーションによってパラメータと種の多様性の関係を導き、その関係から注目している進化の方向である多様な種の存在する環境が得られることを示す。

シミュレーションに応用をした Tierra は、人工生命の分野で研究が進んでいる進化シミュレーションであり、ソフトウェアの進化の可能性を示したということで画期的な成果をあげている。Tierra は一群の仮想マシン語の命令セットで記述されたアルゴリズムを進化させている。このため Tierra 応用したシステムは、仮想マシン語の命令セットを入れ替えることによりさまざまなアルゴリズムを進化させることができる。本論文自身では、

シミュレーションによる進化の方向の解析に力を入れるために植物系の進化のみを扱っているが、Tierra をそのベースにおいたシミュレーションであるためそのコンセプトは将来的に非常に拡張しやすく、植物系に限らずさまざまなアルゴリズムへ発展できる。このように、将来的な発展を考えたときに、より一般的なアルゴリズムを扱えるという Tierra をベースにおいたシミュレーションの解析には重要な意味がある。

進化についてのアルゴリズム的な側面からの研究では、L-System と遺伝的アルゴリズムを組み合わせたもの [Kawasaki,Kikuchi,Shinoda1998] などがある。遺伝的アルゴリズムでは、成長した個体同士は比較しやすいが、片方は成長した個体でありもう片方が生まれたての個体といった、より一般的な比較をしにくい。このようなときに、 Tierra を応用すれば成長段階の異なる個体の比較をより自然に行うことが可能である。

また現在までの Tierra についての応用という点では、Postscript などと組み合わせたもの [Kibe1998] や、本論文と同じように植物をモデル化したもの [Kimezawa1997, Kimezawa1999] などがあげられる。しかし、これらの研究は Tierra の進化のメカニズムを応用して複雑なものをつくりだそうということに力が注がれており、進化がどの方向に進むのかということ解析しようというものではない。本論文では、これらの研究と異なり、複雑なものを作ることを目的とせず、進化がどの方向に起こるのかということを解析するための方法として Tierra を応用したいと考えている。

本論文では、植物系の進化のシミュレーションとその解析を行うために、まずシミュレーションを行うために十分と考えられる生命の定義として 3 つの項目を用意し、それに基づいて植物を数学的にモデル化し、モデル化された植物が仮想的な環境の中でどのように進化をしていくのかを、Tierra を応用してシミュレーションを行う。次に、シミュレーションで得られた情報を利用することで、設定しているパラメータとそれに依存する進化の方向の関係を、分散分析を用いて解析する。そして、その結果から、もっとも種の多様性を得ることができるパラメータを導き、そのパラメータのもとで実際に多様性のある環境を得ることができるということを示す。

本論文の構成は次の通りである。まず、第 2 章で Tierra とは、そして Tierra 的手法とは何かを要約しておく。第 3 章では、今回行ったシミュレーションがどのような植物の数学的なモデルを用いたかを述べる。第 4 章は本論文の中心的な部分であり、シミュレーションをするとどのように進化が起こるかを述べた後で、パラメータとそれに依存する進化の方向の関係を解析し、その関係から多様性のある環境を意図的に得ることができるということを示す。第 5 章では、本論文のまとめとこれからの課題について述べることにする。

第2章　Tierra 概説

この章では、Tierra 及び Tierra 的手法とは何かについて簡単に触れてみることにする。この章は、[Kimezawa1999] をベースにしているので、詳しくはそちらを参照して欲しい。

2.1　Tierra とは何か

ダーウィンの唱えた進化論は、地球上にかつて存在してきた化石などによって発展してきた側面を持っている。しかし、化石や熱帯雨林の中での動植物の観測だけでは、進化を研究する上で十分ではないと考える生態学者もいた。進化生物学者のトム・レイもそのような一人であった。十分でないと考える理由は大きく 2 つあり、その一つは進化のプロセスを観察しつづけるには人間があまりに短命であること、そしてもう一つは、地球の生態系以外の生態系が発見されていないことである。そこで、彼はこの 2 つを克服するために、1990 年 1 月、コンピュータ内部の世界に自己複製をし進化をするシステムを実現した。

それが、 Tierra である。

Tierra には、通常のコンピュータプログラムと違う点が 2 つある。まずは、その内部で発生した自己複製プログラムが外に漏れ出す危険性を防ぐために、オペレーティング・システムの上に仮想的なコンピュータを作り、自己複製プログラムはその内部でのみ実行可能にしたということである。

もう一つは、自己複製プログラムはわずか 32 種類のアセンブラのような仮想マシン語の命令で構成され、ジャンプなどには相補的なパターンによるアドレッシングという独自のアドレッシングを用いているということである。このことによって通常のフォンノイマン型のプログラムでは、たとえ一部であってもランダムな改変を受け付けることができないのに対して、Tierra の自己複製プログラムはランダムな改変に耐えられるようになり、突然変異による進化が可能になったのである。

仮想プログラム Tierra と、自己複製以外の能力を持たない最初の自己複製プログラムが、1990 年 1 月にトム・レイの手によって実行されると、彼の当初の予想をはるかに越える多様な進化の過程が観察されるようになった。

トム・レイにとって Tierra はデジタル・コンピュータの上でカンブリア爆発を実現するためのステップである。カンブリア爆発とは、多様性のない世界から、短期間で多様な生物の世界に変化した 6 億年前の出来事であり、現在の生物の主要な形態はこのときに決定されたと考えられている。もし、 Tierra がカンブリア爆発と同じような現象を示すことができれば、それは生物学的にも、そしてもちろん情報学的な観点からも重要な側面を持つ。

2.2　Tierra のしくみとその進化

ここでは、Tierra がどのようにして動いているのか、そして自己複製プログラムがどのようにして進化するのかを簡単に扱う。

2.2.1　Tierra の基本概念

Tierra の世界は、仮想マシン語で書かれた独立したプログラムとそれを実行する仮想 CPU から構成されるデジタル生物と呼ばれる住人と、彼らの棲息のために仮想的なメモリ空間と仮想 CPU を実行するための時間の割り当てを行う Tierra オペレーティングシステムという 2 種類の構成要素から成り立つ、仮想コンピュータである (図 2.1)。

地球上の生態系の生物は、太陽エネルギーをもとにした限りある資源や物理的空間を獲得し、自己複製をその目的とした活動を行っている。Tierra 内部でそれぞれのデジタル生物が獲得できる資源は 2 種類ある。まず一つ目は、CPU を利用できる時間である。これは、太陽エネルギーをモデル化したものであり多く割り当てられるほど、たくさんの情報処理ができ、より多くの自己複製が可能になる。もう一つは、メモリ空間である。これが大きくなるほど、たくさんの命令を置くことができ、複雑な情報処理を行える可能性が高くなるのである。

Tierra の中で生きるそれぞれのデジタル生物は、メモリ空間内に存在する自分の命令群を、割り当てられた CPU 時間で処理することによって、メモリ空間上に自己複製を行う。したがって、自己複製が効率よく、高速に行える生物はメモリ空間内に広がり、その逆に自己複製がうまく行かないデジタル生物は、リーパーと呼ばれるシステムによって、メモリ空間から取り除かれてしまう。Tierra では、メモリ空間内の命令がランダムに変わってしまったり、デジタル生物が命令を間違って行ってしまうことにより、自己複製などに突然変異が起こり、進化がもたらされるのである。

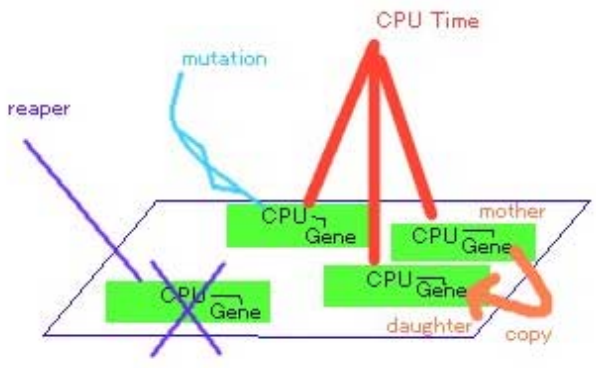

図 2.1: Tierra の基本概念

2.2.2　仮想コンピュータと Tierra オペレーティングシステム

Tierra 自体は、仮想コンピュータであり、デジタル生物はこの仮想コンピュータの上で Tierra オペレーティングシステムに管理されている仮想プロセスとして実装されている。

デジタル生物は、それぞれ仮想 CPU を持っており、この仮想 CPU は Tierra オペレーティングシステムから割り当てられたタイムスライスの時間だけ動くことができる。さらに、 AX,BX などのレジスタと、次にメモリ空間内のどの仮想マシン語の命令を実行すべきかを把握しておく命令ポインタ (IP) を持っている。

たとえば、メモリ空間に表 2.1 のような仮想マシン語のシークエンスが存在したとすると、IP = 01 なら、仮想マシン語命令 incA (レジスタ AX の内容を 1 増やす) を実行し、IP = 02 を設定する。

表 2.1: 仮想マシン語のシークエンスの例

00	movAB
01	incA
02	shl
03	incC
04	jmpo
05	nop0
06	nop1
07	pushA
08	popB

仮想 CPU は、このように一連のサイクルとして、[メモリ空間の IP の位置からの命令の読み込み] → [命令の解釈] → [命令の実行] → [IP を次の命令に設定する] といったものを行い、このサイクルを Tierra オペレーティングシステムによって割り当てられた時間内で繰り返して行くことで情報処理が行われるのである。

2.2.3　Tierra 言語の特徴

Tierra の仮想 CPU によってメモリ空間から読み込まれ実行される命令は、Tierra 言語と呼ばれる仮想マシン語で構成される。この仮想マシン語は生物の DNA,RNA の考え方を取り入れ、大きく 2 つの特徴を持っている。

一つ目は、命令セットにおける命令の種類が実在のコンピュータのマシン語と異なり、たったの 32 種類しかないということである。さらに実在のコンピュータでは、レジスタの指定などにオペランドを利用するが、Tierra 言語の命令にはオペランドは使わない。このことは、生物の DNA が、たった 20 種類のアミノ酸を指定するコードの連なりとして解釈されていることをモデルにしている。

命令セットにおける 32 種類という命令の種類を実現するために Tierra では、たとえば [movAB] という命令は [レジスタ AX の内容をレジスタ BX に移す] といったように、命令自体に計算対象となるレジスタが固定されている。

図 2.2: オペランドの存在しない命令 movAB

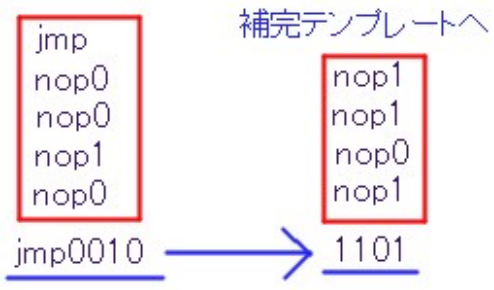

図 2.3: 補完テンプレートによる jmp

命令の位置を示す IP を必要に応じて設定する [jmp] などのように基本的に IP についてのアドレスを必要とす

る命令に対しても、オペランドを指定しない。アドレスを指定する場合には、テンプレートによるアドレスを用いる。実際に IP を目的の位置にジャンプさせる [jmp] 命令を使用する際には、その後に no-operation 命令を複数個伴う。また no-operation 命令は nop0,nop1 と 2 種類ある。例えば

jmp , nop0 , nop0 , nop1 , nop 0

という 5 個の命令が連なっているときは、[jmp 0010] と解釈をし、この 5 個の命令の連なりの最も近くで [0010] と補完の関係にある [1101] になる部分、つまり

nop1 , nop1 , nop0 , nop 1

となる命令群を探し、この命令群の直後に IP を [jmp] させることになる。このような補完テンプレートによるアドレスの方式はたんぱく質などの科学的性質をモデル化したものである。複数のたんぱく質がお互いに反応し合うときには、お互いの 3 次元的な位置を正確に把握しているのではなく、補完する凹凸によって、鍵と錠のような関係で反応が起こるということのモデル化である。

Tierra 言語は、少ない命令数と補完テンプレートによるアドレスという特徴を持つことによって、フォンノイマン型マシンのマシン語のように一部でもランダムな改変を受けるとプログラムの機能を失ってしまうというようなことがなくなり、たとえコードの一部に突然変異が起きたとしてもプログラム自体は依然として実行可能という頑強性を持っている。

2.2.4　先祖種

Tierra では、そのメモリ空間に最初に置かれるデジタル生物を先祖種と呼ぶ。トム・レイが最初に設計した先祖種 0080aaa は、基本的に自己複製の機能を持つだけであり、意図的に進化が起こるような仕組みは含まれていないという。

先祖種である 0080aaa のコード (図 2.4)、つまり命令群は大きく 3 つのブロックとして考えることができる。

自己検査ブロック (0-22)

自己検査ブロックの行うことの目的は、自分のコードがメモリ空間内のどこからどこに存在するかを調べることであり、それぞれ adrb,adrf という命令を用いる。

まず、0080aaa はそのコードが

nop1 , nop1 , nop1 , nop 1

にはじまり、

nop1 , nop1 , nop1 , nop 0

で終わっている。これに対して、

adrb , nop0 , nop0 , nop0 , nop 0

を行うと、この 4 つの nop と補完の関係にあるテンプレートをアドレスが減少する方向に探しコードの開始位置を特定する。また同様のことが adrf でも行われ、コードの終了位置が特定される。このことによって、自分のコードの開始位置と終了位置を得ることができるのである。

自己複製ループブロック (23-39)

自己複製ループでは、mal 命令によってコピー先のメモリ空間を確保する。そして call 命令によって次のコピープロシージャへと IP を移動させるのである。

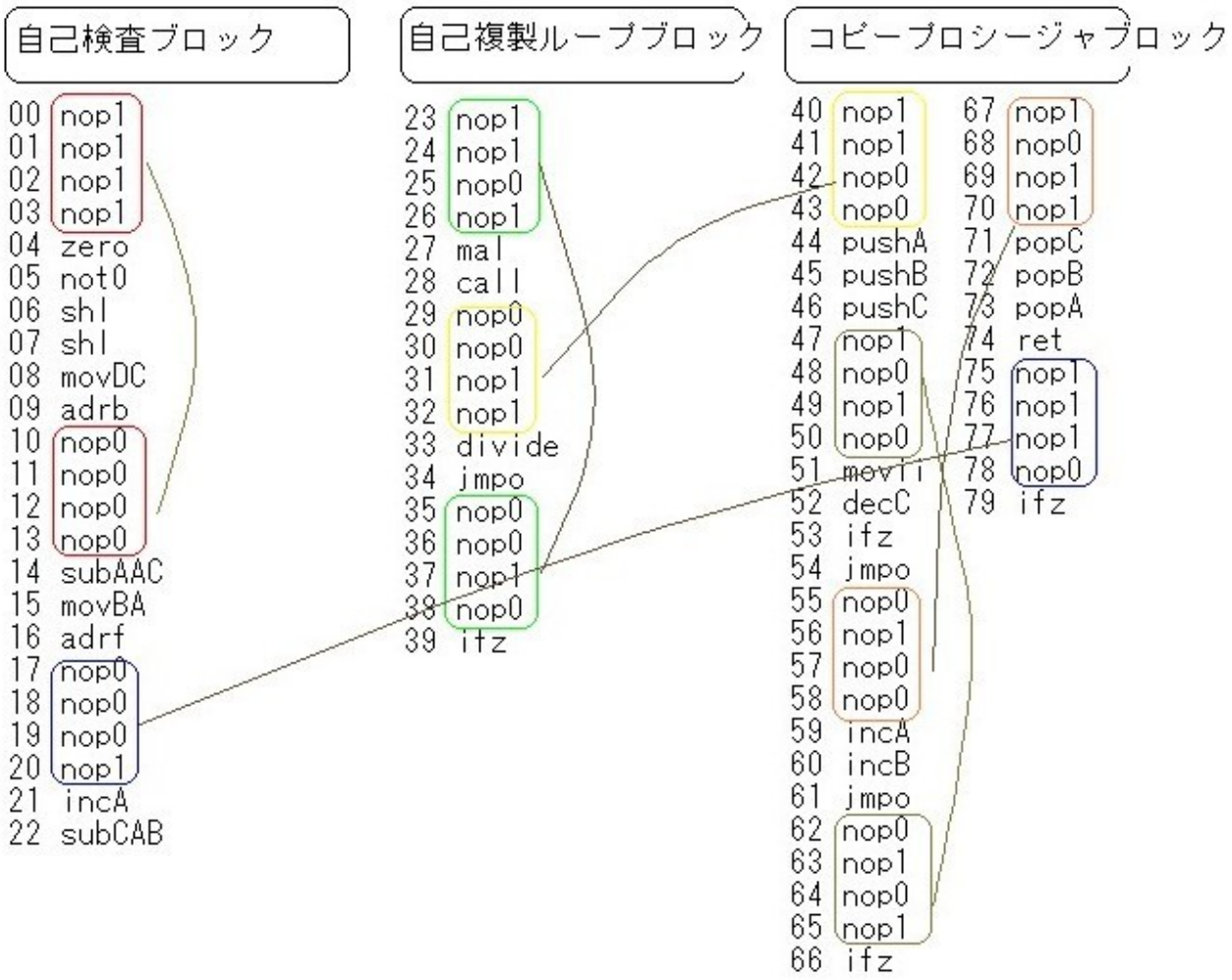

図 2.4: 先祖種 0080aaa のコード

コピープロシージャブロック (40-79)

コピープロシージャにおいて、コピー元になったデジタル生物の命令群は movii 命令によってコピー先にコピーされる。

コピーが終了すると、自己複製ループに戻り divide 命令を実行する。この命令によってコピー元のデジタル生物はコピー先への書きこみ権を失い、コピー先は新しいデジタル生物として活動を始める。divide 命令終了後、コピー元のデジタル生物は、自己複製ループを繰り返して行うことになる。

これら 3 つのブロックから先祖種 0080aaa は成り立っており、命令を処理していくことで自己複製を達成するのである。

2.2.5　リーパー

先祖種から始まる自己複製は、時間が進むにつれてメモリ空間を占有していくことになる。しかし、メモリ空間がいっぱいになってもそれまでのデジタル生物が存在してしまうと、新しい複製を行うことができなくなってしまう。

そこで Tierra オペレーティングシステムには古くなった生物を消去する「リーパー」というシステムが存在する。Tierra 内部のデジタル生物は、長い間メモリ空間にいたり、補完するテンプレートが見つからないなどの理由で命令の実行に失敗すると、リーパーの対象になりやすくなる。このシステムにより、よりすぐれたデジタル生物はメモリ空間に広がりやすくなり、逆にあまり優れないデジタル生物はメモリ空間から消去されやすくなるのである。

2.2.6　突然変異とビット反転による進化

メモリ空間に存在する命令は、主に 2 つの理由から変化する。

ひとつは、メモリ空間内にランダムに起こるビット反転である。仮想マシン語は 32 種類なので 5 ビットで内部的に表現され、そのなかの 1 ビットが反転することで別の命令になるのである。例えば、nop0 は 5 ビットであらわすと 00000 であり、最下位のビットが反転して 00001 になると nop1 になる。これは、 DNA などが宇宙からの紫外線によって傷つくことをモデル化したものである。このことによって、実行しようとしていた命令や、コピーの対象になっていた命令が変化をすることになる。

もうひとつは、 movii 命令の際に起こるビット反転である。movii 命令は、ある一定の確率でコピーしようとしていた命令の一部分を変化させ、コピー元とコピー先とで違う命令になるように設計されている。

これら 2 つの理由から起こるランダムな命令の変化によって、自己複製でできる次の世代が、元の世代とは異なる命令を持つようになる。つまり突然変異が起こるのである。

例えば、先祖種 0080aaa において、42 番目の命令の最下位ビットが反転すると 40-43 番目の命令は

nop1 , nop1 , nop1 , nop 1

になる。これは、終了コード位置を特定する

adrb , nop0 , nop0 , nop0 , nop 0

と補完の関係にあるために、43 番目までを自分のコードだと判断し (図 2.5)、この部分までのコピーのみを行うことになる (参照 2.2.4 小節)。このことによって、新しく長さ 45 のデジタル生物が存在するようになる。

このように、ビット反転によってさまざまなデジタル生物が誕生し、突然変異、遺伝、自然選択の 3 要素がそろい進化が促されるのである。

2.2.7　進化によって現れる種

Tierra 内部のデジタル生物は、時間がたつにつれてさまざまな種へと進化をする。ここでは、その中でも典型的かつ特徴的なものを数種類あげることにする。

図 2.5: ビット反転による進化

パラサイト

トム・レイの実験に現れた 0045aaa などのパラサイトと呼ばれる種はコピープロシージャを持っていない種である。そのために、単独では自己複製できない。しかし自分の近くのメモリ空間に先祖種があると、そのコピープロシージャを利用して自己複製を行う。この種は命令の数が先祖種よりも少ないために、高速に自己複製を行うことができ、次第に先祖種を圧倒するようになる。(図 2.6)

パラサイトに対する免疫種

これはコピープロシージャを呼ぶ際のアドレスのテンプレートを変化させたことによって、パラサイトがこの種のコピープロシージャを使うことを防ぐことを可能にした種である。

ハイパーパラサイト

パラサイトの攻撃を逆に利用することから、ハイパーパラサイトと呼ばれる種も存在する。ハイパーパラサイトは、寄生してきたパラサイトの CPU に自分の自己検査ブロックを実行させてコードの開始アドレスと終了アドレスを自分のものにセットしなおすことによって、パラサイトにハイパーパラサイトのコードをコピーさせるのである。このことで、ハイパーパラサイトは 2 倍の自己複製能力を持ちパラサイトは自己複製ができなくなってしまう (図 2.6)。

社会的ハイパーパラサイト

社会的ハイパーパラサイトは、ハイパーパラサイトの中でもさらに特徴を持った種である。この種はお互いがメモリ空間の近くにいることで、テンプレートを共有することが可能になっている。このことによってコードのコピーの時間が短縮され、より多くの自己複製を行うことができるのである。

ここに上げたほかにも、より高度に進化をした種が存在する。また共生種が出現する可能性もある。

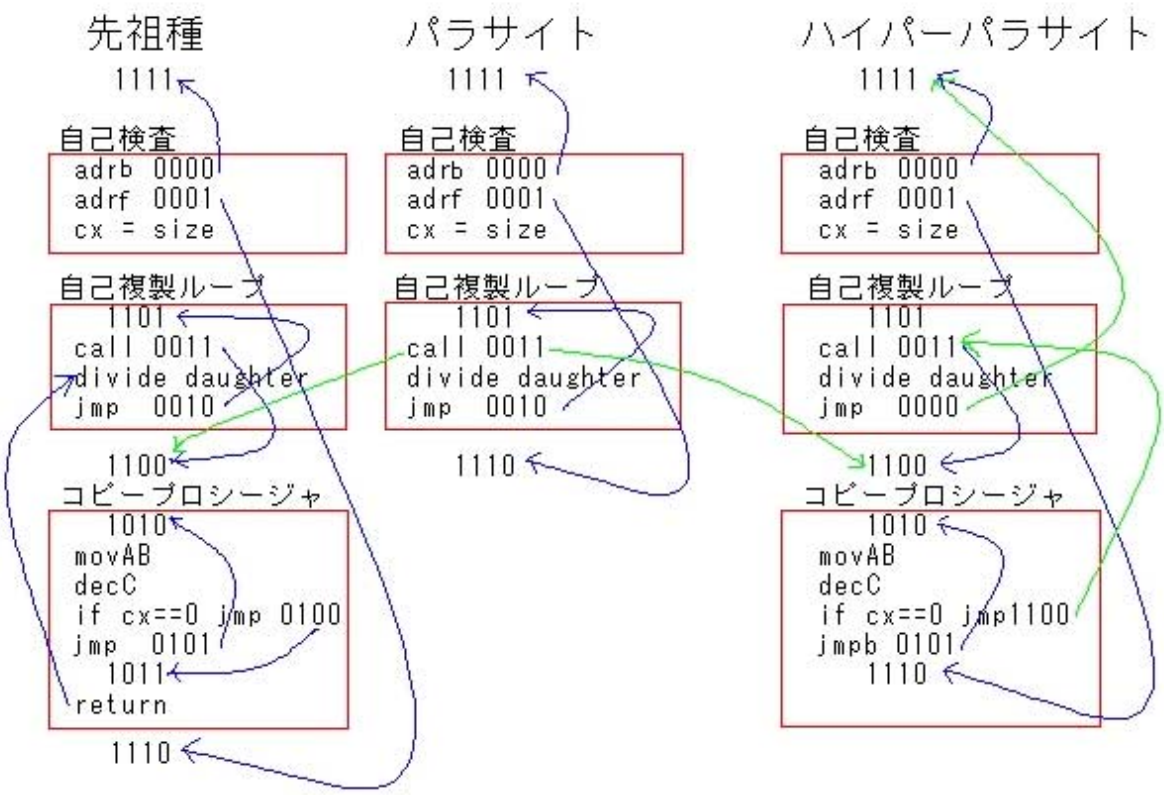

図 2.6: 先祖種とパラサイトとハイパーパラサイトの相互関係

2.3　Tierra 的手法とは何か

Tierra の重要な側面として、自己複製能力をもったアルゴリズムの進化がある。このアルゴリズムの進化を利用するのが Tierra 的手法である。したがって Tierra 的手法において最も核となる存在は、仮想マシン語でできた遺伝コードを持ち自己複製能力をそなえたアルゴリズムである。

Tierra 的手法で、進化を促すために必要な主なものとして、

1. アルゴリズムを表現するための仮想マシン語セット
2. アルゴリズムの進化の舞台となる仮想空間
3. アルゴリズムを自然選択する手段
4. それぞれのアルゴリズムを管理するオペレーティングシステムの役割をする存在

が考えられる。

本論文では、Tierra 的手法を用いるにあたって、これらのものを次章で定義をし、植物系の進化のシミュレーションとその解析を行う。

第3章　植物系のモデル化

3.1　基本概念

本論文では Tierra 的手法を用いて、植物系がどのような方向に進化するかをシミュレーションし、その解析を行い、意図的に多様な種の存在する環境を作り出すのが目的である。ここでは、植物系をどのようにモデル化して扱うかを考えていくことにする。

本章では、はじめに本論文で扱う生命の定義を用意し、これをベースとして植物を枝、葉、花という大きく 3 つのパーツとして表現する。3 次元フィールドに配置された植物は、日光を受け取りながら、遺伝子として保存している Tierra 言語 Like なマシン語を実行することで、成長をしたり、たねを作ったりし、次第にフィールドを被い尽くすようになる。

ここでたねを作るときの遺伝子のコピーに突然変異を起こさせると、新しい種が現れ進化が促される。ここで現れた種が、既存の種よりも効率的に日光を受け取ったり、たねを作ることができれば新しい種は次第にフィールドに広がることになり、新しい環境が生まれることになる。なお、ここから先では植物の「たね」はひらがなで表現に、そして種類をあらわす「種 (しゅ)」は漢字での表現に統一する。本論文で扱う種については、3.4 節で定義を行なう。

4.2 節ではモデル化された植物を使って、フィールドの単位面積あたりの日光量と、枝、葉、花という植物のパーツにどれだけの維持カロリーが必要か、というパラメータをさまざまに変えてシミュレーションすることによって、どれだけ進化の方向が変わるかを、さまざまな角度から観測する。このシミュレーションは、意図的に多様な種の存在する環境を作り出すために必要な情報を収集するために行われる。

3.2　生命の定義としての 3 つの項目

生命は、さまざまな定義があり、必ずしも一意ではない。ここでは、Tierra 的手法を用いてシミュレーションを行うのに十分であると考えられる定義として 3 つの項目 (表 3.1) を用意し、これを満たすものを生命と呼ぶことにする。

表 3.1: 生命の定義

1. 自己とそれ以外の区別があること
2. 活動のためのエネルギーを外部から獲得すること
3. 自己複製子であること

これらの定義のうち、”1. 自己とそれ以外の区別があること” はこの定義がないとどの部分を生命と呼ぶべきなのか、どの部分は生命と呼ぶべきではないかが不明になり、生命についての情報を収集できなくなるからであ

る。”2. 活動のためのエネルギーを外部から獲得すること” は 1 の定義で定義される外部と何らかの形で依存関係になければならないことを意味していて、生命内部だけの閉じた環境というものをなくすためである。

ただし、”3. 自己複製子であること”という条件は 1,2 のようにほとんど自明のような条件とは少し違う。この条件は、利己的な遺伝子についての考えをベースにしてあり、進化を促すためのものである。したがって、生命と進化を別々に考えるならば不要となりえる項目であるが、本論文では進化を扱うために必要と考えた項目である。

3.3　植物の表現方法と中間表現の定義

前節 3.2 を受けて、植物の表現方法をここで扱う。

まず、植物の主要なパーツとして前節 3.2 の表 3.1 に添った形で表 3.2 の 3 つを定義する。

表 3.2: 植物の主要なパーツの定義

1.	枝 (E)	自己とそれ以外の区別があるもの
2.	葉 (L)	活動のためのエネルギーを外部から獲得すること
3.	花 (F)	自己複製子であること

さらに、これらのパーツをサポートする 2 つのパーツを定義する。(表 3.3)

表 3.3: 植物のサポートするパーツの定義

1.	ノード (N)	枝と枝をつなぐ役割
2.	シード (S)	個体を統括する役割

シードの個体を統括する役割というのは、枝、葉、花、ノードの他の 4 つのパーツはシードに属していると考えているためである。つまり、ここではシードという言葉と個体という言葉を同意義に用いており、同じシードに属する枝や葉は同じ個体の 1 部分であると考えているのである。また、シードと個体という言葉が同意義であるため、個体の現在のエネルギーと、シードに貯えられている現在のエネルギーは一致する。

これら 5 つのパーツを用いて、表 3.4 にある Context Free Like Language から生成される文字列によって植物は中間表現される。表 3.4 のなかの 3 については、図 3.1 にも示してある。なお、表 3.4 の 4 のところに出てく

表 3.4: 植物を中間表現する Context Free Like Language

0.	初期状態	N
1.	ノードの追加	$N \rightarrow (NE < x,y,z > N)$
2.	葉の追加	$N \rightarrow (NE < x,y,z > L < radius >)$
3.	花の追加	$N \rightarrow (NE < x,y,z > F < radius >)$ (図 3.1)
4.	枝の削除	$(NE < x,y,z > A) \rightarrow N$

る A は、削除される E の先についているノードや花などを表している。また $<x,y,z>$ というのは、もとのノードから次のノードや葉までの 3 次元的な相対位置であり、 *radius* は葉や花の大きさを表現している。

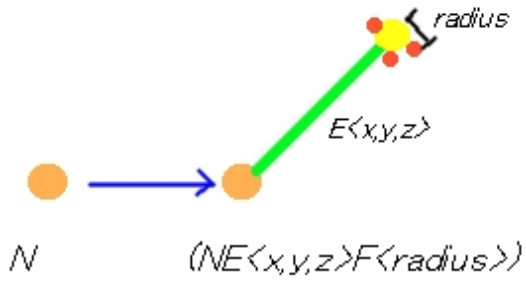

図 3.1: Context Free Like Language による花の追加

表 3.4 の Context Free Like Language で表される植物の具体例を見ていくことにする。

初期状態

N　　(図 3.2)

→　ノード 0 に相対位置 $<0,0,1>$ のところにノードを作る

$(NE<0,0,1>N)$　　(図 3.3)

→　ノード 1 に相対位置 $<1,0,1>$ のところに半径 1 の葉を作る

$(NE<0,0,1>(NE<1,0,1>L<1>))$　　(図 3.4)

→　ノード 1 に相対位置 $<-1,0,1>$ のところに半径 1 の花を作る

$(NE<0,0,1>((NE<-1,0,1>F<1>)E<1,0,1>L<1>))$　　(図 3.5)

→　枝 2 を削除

$(NE<0,0,1>(NE<-1,0,1>F<1>))$　　(図 3.6)

なお、ノードや葉、花、枝の番号は操作が行われるときに何番目にあるかということで決定される。つまり、ノード 1 というのは 1 番目の N に対応するノードのことである (数える順番は 0 番から始まる)。

図 3.2: 初期状態

図 3.3: ノードの追加

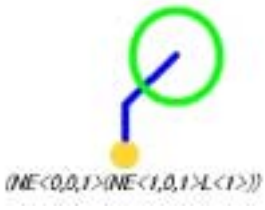

図 3.4: 葉の追加

この中間表現が変化していくことで、植物は成長をする。中間表現は 3.5 節で取り上げる CPU とそれによって実行されるマシン語によって変化をする。

図 3.5: 花の追加

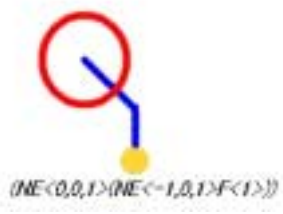

図 3.6: 枝の削除

3.4　植物の構成と種の定義

それぞれの植物は、ジーン、CPU、ABuffer と呼ばれる部分を持っている (図 3.7)。ジーンはマシン語のシークエンスであり、例えば表 3.5 のようなシークエンスである。このマシン語のシークエンスを CPU が解釈、実

表 3.5: マシン語のシークエンスの例

00	set_leaf_energy
01	zero
02	increment
03	shift_left
04	shift_left
⋮	⋮

行を行なう。その実行の結果、中間表現を格納するバッファ ABuffer が変化をし、それに対応して植物自体の形態も変化する。

植物は葉からの日光吸収によって得られるエネルギーをもとに成長し、花を咲かせ、花から新しいたねをつくることで自己複製をする (図 3.8)。より効率良くエネルギーを得られ、新しいたねをつくることができる植物ほど、フィールドに広がりやすい性質を持っていると言うことができる。

図 3.7: 植物の構成

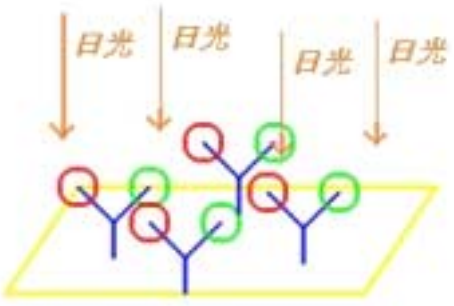

図 3.8: 日光を受ける植物

植物の種は、ジーンの等値性によって定義される。つまり、2 つの個体があったとき、その 2 つの個体の表 3.5 のようなジーン (マシン語のシークエンス) が完全に一致したときに限り、同じ種であるとする。ここで定義さ

れる種は、表現型によるものではなく、遺伝子型によって決定される。このことは、同じように葉や花をつけたとしても、同じ種であるとは限らないということである。実際の生物では全ての個体で遺伝子の並びが異なるため、ここでの種の定義と実際の生物の種の定義は異なるものとなるが、本論文でのシミュレーションでは、遺伝子型で種を特定することにしている。このことによって、明確に種を区別することが可能になっている。

3.5　CPU とマシン語セット

CPU は、各シードに対してそれぞれ独立に存在する。その内部は、概念的に 表 3.6 のようになっており、この CPU が後で示すマシン語セットの命令を実行することでそれぞれのシードが変化をする。

表 3.6: CPU 内部の構造

1.	ABuffer	中間表現を格納
2.	PartsList	枝、葉、花、ノードの情報を格納
3.	Energy	現在貯えているエネルギーなどの情報
4.	Gene	マシン語のシーケンス
5.	IP	Gene の中の何番目の命令を実行するか示すもの
6.	Registers	内部レジスタ (参照:表 3.7)
7.	Stack	内部レジスタ用スタック
8.	RegisterIndex	使用するレジスタを特定するためのインデックス

この中で、内部レジスタとしては表 3.7 のようなものが用意されている。

表 3.7: CPU 内部の構造

	レジスタ名	
1.	r	座標計算用の値
2.	theta	座標計算用の値
3.	phy	座標計算用の値
4.	edge_number	中間表現の枝を特定するためのレジスタ
5.	leaf_number	中間表現の葉を特定するためのレジスタ
6.	flower_number	中間表現の花を特定するためのレジスタ
7.	node_number	中間表現のノードを特定するためのレジスタ
8.	leaf_energy	葉に送るエネルギーの値
9.	flower_energy	花に送るエネルギーの値
10.	counter	ループに使用するカウンタ

ここで、< r,theta,phy > の 3 つのレジストリは、極座標からデカルト座標 $< x,y,z >$ へと変換され、中間表現の相対座標に使用される。また、それぞれのレジスタは初期値として 0 が設定される。

このような概念を持った CPU で、次に示すようなマシン語のシークエンスを実行していくことになる。たと

えば、マシン語のシークエンスの例としては、表 3.8 のような感じである。仮に IP = 03 ならば、set_node_number が実行されその後で IP = 04 に設定される。

表 3.8: マシン語のシークエンスの例

00	set_r
01	increment
02	shift_left
03	set_node_number
04	zero
05	nop0
06	nop1
07	make_node
08	increment
09	jmpb
10	nop1
11	nop0
12	gene_separate

ここから、CPU に対して用意された 32 種類 (16 進数で 0x00-0x1F) からなるマシン語の命令を一つ一つ詳しく見ていくことにする。なおここで示すマシン語は 2.2.3 小節で示した考え方（少ない命令数とテンプレートによるアドレス方式）を取り込んだものである。また、ここから先はたくさんのパラメータがあるため、どのようなパラメータがあり、それらがどのようなデフォルト値になっているかは、この章の最後の表 3.14 にまとめてある。

00:nop0

テンプレート用の no-operation 命令。詳しくは、04:jmpf を参照。

01:nop1

テンプレート用の no-operation 命令。詳しくは、 04:jmpf を参照。

02:gene_separate

CPU では no-operation として扱う特殊な命令。3.8 節の自己複製を参照。

03:if_counter_zero

レジスタの counter が 0 に近いとき（具体的には -0.5 < counter < 0.5 のとき）、次の命令を実行し（IP が 1 増える）、そうでないときは次の命令を飛ばす（IP が 2 増える）。

04:jump_forward(jmpf)

この命令に続くテンプレートに補完するテンプレートが見つかるところまで IP を移動させる。たとえば、表 3.9 のように命令の列があって、IP = 01 とすると、jump_forward 命令を実行することになる。この場合は、その

表 3.9: jump_forward , jump_backward の例

00	increment
01	jump_forward
02	nop0
03	nop1
04	gene_sepate
05	decrement
06	nop1
07	nop0
08	nop1
09	if_counter_zero
10	jump_backward
11	nop1
12	nop0
13	gene_separate
14	if_counter_zero

後に続くテンプレートが 02:nop0 , 03:nop1 であり、このことから

jump_forward 01

として解釈される。ここで 01 に補完するテンプレートを IP の増加方向で考えてみると 06:nop1 , 07:nop0 から構成される 10 が見つかる。よって IP = 08 がセットされ、次に実行される命令は 08:nop1 になる。

なお、jump_forward 命令はテンプレートを伴わなかった場合や補完するテンプレートを IP の増加方向で発見できなかった場合は失敗し、 IP が単純に 1 増えることになる。

05:jump_backward

この命令は、jump_forward 命令に似ているが、補完するテンプレートを IP の減少方向で探すところが異なっている。たとえば、表 3.9 で、 IP = 10 なら jump_backward 命令によって次は IP = 09 になる。

06:call

この命令は、次の ret 命令と組み合わて利用することによってプロシージャを構成するためのものである。この命令も、jump_forward , jump_backward と同様にテンプレートによる jump を行い、さらに現在の IP をスタック上に push しておく。このことによって、ret 命令は、スタック上から IP を pop することで call 命令の次の命令に復帰することができるようになっている (図 3.9)。

なお、call におけるテンプレート探索は先に IP 増加方向に探索し、それに失敗したときは減少方向に探索を行う。両方ともに失敗したときには no-operation と同じ扱いになり、IP のスタック上への push も行わない。

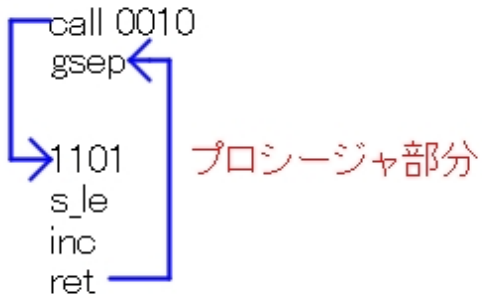

図 3.9: call 命令と ret 命令

07:ret

call によって push された IP を pop することで、call 命令の次の命令へと復帰する。また、スタック上に IP が push されていないときは、no-operation と同じ扱いである。

08:push

RegisterIndex で指定されているレジスタの内容をスタック上に push しておく。なお、ここで使うスタックは call,ret 命令とは別に用意されているものであり、RegisterIndex で指定されるレジスタは表 3.7 のうちの一つである。

09:pop

push 命令でスタック上に push された内容を、RegisterIndex で指定されているレジスタの内容に pop する命令である。また、スタック上に push されていないときは、no-operation と同じ扱いである。

0A:shift_left

RegisterIndex で指定されているレジスタの内容を 2 倍にする。

0B:shift_right

RegisterIndex で指定されているレジスタの内容を 0.5 倍にする。

0C:increment

RegisterIndex で指定されているレジスタの内容を 1 増やす。

0D:decrement

RegisterIndex で指定されているレジスタの内容を 1 減らす。

0E:zero

RegisterIndex で指定されているレジスタの内容を 0 にする。

0F:cut_edge

edge_number レジスタで指定された枝を ABuffer から探し、その枝から派生するすべてのパーツを取り除き、それにあわせて ABuffer を調整する。（表 3.4(4)）

10:set_flower_number

RegisterIndex によって指定されるレジスタを flower_number にする。

11:set_leaf_number

RegisterIndex によって指定されるレジスタを leaf_number にする。

12:set_flower_energy

RegisterIndex によって指定されるレジスタを flower_energy にする。

13:set_leaf_energy

RegisterIndex によって指定されるレジスタを leaf_energy にする。

14:set_node_number

RegisterIndex によって指定されるレジスタを node_number にする。

15:set_edge_number

RegisterIndex によって指定されるレジスタを edge_number にする。

16:set_phy

RegisterIndex によって指定されるレジスタを phy にする。

17:set_theta

RegisterIndex によって指定されるレジスタを theta にする。

18:set_r

RegisterIndex によって指定されるレジスタを r にする。

19:set_counter

RegisterIndex によって指定されるレジスタを counter にする。

1A:make_flower

表 3.4(3) をもとにして、花を追加する。

具体的にはまず、node_number で指定されるノードの中心座標 $< x_0, y_0, z_0 >$ を得る。node_number の数が大きすぎるなどの理由からノードを特定できない場合は、この時点で失敗する。次に、3 つのレジスタ、 r, theta, phy から 、極座標 $< \mathrm{r}, \mathrm{theta}, \mathrm{phy} >$ をデカルト座標 $< x_d, y_d, z_d >$ に変換する。ただし、r がパラメータ *mSeedList_min_r* よりも小さいときには失敗する。これは、 r = 0 、つまり長さ 0 の枝を作らせないためである。また、デカルト座標に変換する際には、theta , phy はそれぞれパラメータ *mCPU_ad just_theta* , *mCPU_ad just_phy* を掛け合わせた値が使われる。つまり、

$$\begin{cases} x_d &= \mathrm{r} \times \sin(\mathrm{theta} \times \mathit{mCPU\_ad\,just\_theta}) \times \cos(\mathrm{phy} \times \mathit{mCPU\_ad\,just\_phy}) \\ y_d &= \mathrm{r} \times \sin(\mathrm{theta} \times \mathit{mCPU\_ad\,just\_theta}) \times \sin(\mathrm{phy} \times \mathit{mCPU\_ad\,just\_phy}) \\ z_d &= \mathrm{r} \times \cos(\mathrm{theta} \times \mathit{mCPU\_ad\,just\_theta}) \end{cases}$$

によって変換される。そして新しい花の中心を $< x_1, y_1, z_1 >=< x_0 + x_d, y_0 + y_d, z_0 + z_d >$ とする。ここでも、$< x_1, y_1, z_1 >$ がフィールドの外に出てしまった場合は失敗する。

次に、$< x_0, y_0, z_0 > - < x_1, y_1, z_1 >$ の間の枝を考える。枝は、パラメータ *mEdge_radius* で指定される半径を持つ円柱として考えられる。ここでは、新しい枝が既存の葉や花と 3 次元的に衝突する場合は、失敗に終わる。ただし、枝と枝の衝突は計算が複雑になるために行っていない。さらに、この枝の体積にパラメータ *mEdge_calorie* を掛け合わせた分のエネルギーを枝を作るのに消費する。

ここまでで、$< x_1, y_1, z_1 >$ を中心とする半径 0 の花と $< x_0, y_0, z_0 > - < x_1, y_1, z_1 >$ の間の枝の枝ができ、これにあわせて ABuffer の方も変更する。

1B:make_leaf

表 3.4(2) をもとにして、make_flower と同様に、葉を追加する。

ただし、make_leaf で追加される葉は、半径が 0 ではない。半径 *radius* は、パラメータ *mLeaf_calorie* とレジスタ leaf_energy によって、

$$(\frac{4\pi}{3}(radius)^3) \times mLeaf\_calorie = \text{leaf\_energy}$$

を満たすように計算される。つまり、単位体積あたりのカロリー *mLeaf_calorie* と体積とを掛け合わせた積が、leaf_energy に一致するようにするのである。また、この葉をつくるのに枝と同じようにエネルギーを消費し、その消費する値は leaf_energy と同じであり、同時に枝を作るエネルギーも消費する。さらに、半径が 0 でないので、この葉が既存の枝や葉、花と衝突しないかも調べ、もし衝突するようならその時点で失敗する。

1C:make_node

表 3.4(1) をもとにして、make_flower と同様に、ノードを追加する。これは、追加されるのが花ではなくノードであるということ以外は、make_flower と同様である。

1D:grow_flower

すでに ABuffer に存在している花を大きくする。

具体的には、まずレジスタ flower_number で定義される花を ABuffer 上から探す。このときに、flower_number の数が大きすぎるなどの理由で花を特定できないときは失敗する。次に、特定された花の現在の大きさ $radius_C$ とパラメータ *mFlower_calorie* とレジスタ flower_energy から、次の式を満たすように新しい大きさ $radius_N$ を計算する。

$$(\frac{4\pi}{3}(radius_N)^3) \times mFlower\_calorie = \text{flower\_energy} + (\frac{4\pi}{3}(radius_C)^3) \times mFlower\_calorie$$

つまり、現在花に蓄えられているエネルギーに flower_energy の値を加え、そこから新しい半径 $radius_N$ を計算するのである。なお、ここで花を大きくするときも、すでにある枝や、葉、花と衝突する場合は失敗し、花の大きさは変わらない。また、花を大きくすることが可能な場合には、flower_energy の分のエネルギーを消費し花を大きくし、それに合わせて ABuffer の更新を行なう。

1E:grow_leaf

この命令は、花が葉に変わることを除けば、grow_flower と全く同じである。

1F:breed_flower

この命令は、現在のシードをベースにして新しいシードを作るためのものである。詳しくは、3.8 節を参照すること。なお、この命令では flower_number で指定される花から新しいシードを作るが、そのときにその花のエネルギーを新しいシードに渡してしまうため、花に蓄えられているエネルギーは 0 になり、それに合わせて ABuffer も更新される。

3.6　フィールドと日光の与え方と受け取り方

今回利用するフィールドは、*xyz* 座標を持つ 3 次元空間のうち *xy* 平面を地面として、*z* 軸の正の方向を空のある方向として考えられたフィールドである。具体的には、4 つのパラメータ *mField_min_x, mField_min_y, mField_N, mField_haba* によって計算される *xy* 平面上の正方形とその上に広がる 3 次元空間である。*xy* 平面上の正方形は、*x* 軸方向が

$$[mField\_min\_x, mField\_min\_x + mField\_N \times mField\_haba]$$

で決定され、*y* 軸方向が

$$[mField\_min\_y, mField\_min\_y + mField\_N \times mField\_haba]$$

で決定される。付け加えていえば *z* 軸方向は

$$[0, \infty]$$

である。

make_node, make_flower, make_leaf などでつくられる新しいパーツは、かならずこのフィールド上に存在しなければならず、このフィールドの外にパーツをつくることはできない。ただし、葉と花はその中心点がフィールド内にあれば良く、その一部分が *xy* 平面上の正方形から外れても構わないが、*z* 軸で負になる部分には一部分でもはみ出してしまったら新しいパーツをつくれない。つまり地面にもぐり込むようなパーツはつくれないということである。

フィールドは、*x* 軸方向、*y* 軸方向ともにパラメータ *mField_haba* ごとに区切られ、*xy* 平面は、*mField_N* の自乗個の格子に区切られる。日光は、*z* 軸の無限大の方向から *z* 軸に並行にフィールドの垂直方向に、格子あたり *mField_daylight* の量で照射している。

次に、それぞれのシードがどれだけ日光を受けているのかを考えることにする。まず、フィールド内の全ての葉と花を高さでソートし、高い順に (*z* 軸の値の大きい順に)、どれだけの日光を受けているかを計算する。例えば、一番上にあるのが葉のときには、葉の中心座標 $< l_x, l_y, l_z >$ と半径 *radius* を取得し、フィールド上のどの格子と重なるかを計算する。このとき、正方形である格子の中心座標を $< b_x, b_y, 0 >$ として、

$$\sqrt{(b_x - l_x)^2 + (b_y - l_y)^2} \leq radius$$

を満たすとき、葉と格子は重なっていると判断する。重なった格子のフィールド上の日光のうち *mField_downsize* で決定される割合の分、つまり

$$mField\_daylight \times mField\_downsize$$

を葉が吸収し、余りはフィールド上に残される形になる。このフィールド上に残された日光は、その格子の低い高さのところにあるパーツの計算のときに再び利用される。したがって、あるパーツについての日光の量を計算しているときに、上のパーツと重なる格子がある場合には、その格子については、

$$mField\_daylight \times mField\_downsize \times (1 - mField\_downsize)$$

の分の日光を受けることになる (図 3.10)。

これを全ての葉と花で計算し終わったところで、それぞれのシード別に、パーツとして持っている葉が吸収したエネルギーを計算し、その和をシードの得たエネルギーとして格納する。

また、植物によって吸収された日光の量が、フィールド全体に与えた日光の量のどれだけを占めるかという割り合いを計算し、この値をフィールドキャパシティーと呼ぶことにする。フィールドキャパシティーがあるパラメータ *mField_capacity* を超えるときには、フィールドが過密になったと判断し、シードの選択 (3.9 節) の段階で削除されるシードの数が増えることになる。

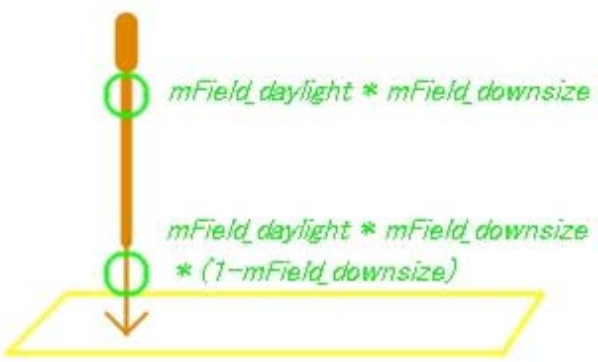

図 3.10: フィールド上の日光の受け取り方

3.7　先祖種

今回フィールド上に最初に置かれるシード、つまり先祖種は、ID:61-0-1 である。なお ID は、[ジーンの長さ-シードが誕生した時間-特殊ビット] で表現され、特殊ビットは常に 1 である。ジーンの長さとは、そのシードが持っているマシン語のシークエンスの長さのことであり、シードが誕生した時間とは、そのシードが誕生するまでに全てのシード全体で何回のマシン語が処理されたかということである。

先祖種 (ID:61-0-1) のジーン (マシン語シークエンス) は、表 3.10、表 3.11 に示しているものである。先祖種のジーンは要約すると、葉と花をひとつずつ作り、葉でエネルギーを吸収して花を大きくし、ある程度花が大きくなったら新しいシードを作るというもの (図 3.11-3.18) である。また、花を大きくするところとシードを作るところは無限ループなので、先祖種が生きている限り何度でもシードが作られる。また、シードの最初のエネルギーは 1000 に設定されているので、新しいシードにこれ以上のエネルギーを送れる大きさに花がなってから breed_flower 命令を実行し次の世代のシードを作るようなジーンにしてある。

なお、デフォルトではパラメータ $mCPU\_adjust\_theta, mCPU\_adjust\_phy$ はともに $\frac{\pi}{6}$ なので、theta, phy に対しての increment 命令ではその値が $\frac{\pi}{6}$ 増える計算になる。

3.8　自己複製と突然変異

シードは、 breed_flower 命令を実行することで新しいシードを作り自己複製をする。ここでは、その自己複製の過程を詳しく見てゆくことにする。なお、自己複製の過程のひとつである Daughter Gene の作成のステップで突然変異が起こるようにしてあり、確率的に新しい種が誕生する。

花の特定

まず、breed_flower を実行したシードを Mother Seed 、新しくできるシードを Daughter Seed とする。breed_flower のときのレジスタ flower_number で指定される花を Mother Seed の ABuffer から特定する。もしこの花の特定に失敗したら、その時点で breed_flower 命令が失敗する。

表 3.10: 先祖種 (ID:61-0-1) 前半

00	set_leaf_energy	レジスタを leaf_energy にセット (図 3.11,3.12)
01	zero	leaf_energy = 0
02	increment	leaf_energy = 1
03	shift_left	leaf_energy = 2
04	shift_left	leaf_energy = 4
05	gene_separate	
06	shift_left	leaf_energy = 8
07	shift_left	leaf_energy = 16
08	set_flower_energy	レジスタを flower_energy にセット
09	zero	flower_energy = 0
10	increment	flower_energy = 1
11	shift_left	flower_energy = 2
12	shift_left	flower_energy = 4
13	shift_left	flower_energy = 8
14	shift_left	flower_energy = 16
15	shift_left	flower_energy = 32
16	gene_separate	
17	shift_left	flower_energy = 64
18	shift_left	flower_energy = 128
19	shift_left	flower_energy = 256
20	shift_left	flower_energy = 512
21	set_r	レジスタを r にセット
22	zero	r = 0
23	increment	r = 1
24	shift_left	r = 2
25	shift_left	r = 4
26	shift_left	r = 8
27	gene_separate	
28	make_node	r = 8 , theta = 0 , phy = 0 にノード 1 を作成 (図 3.13,3.14)
29	set_node_number	レジスタを node_number にセット
30	increment	node_number = 1

表 3.11: 先祖種 (ID:61-0-1) 後半

31	gene_separate	
32	set_theta	レジスタを theta にセット
33	increment	theta = 1
34	shift_left	theta = 2
35	gene_separate	
36	make_leaf	ノード 1 から r = 8 , theta = $\frac{\pi}{3}$, phy = 0 の相対位置に葉 0 を作成 (エネルギーが 16)　　(図 3.15,3.16)
37	gene_separate	
38	set_leaf_number	レジスタを leaf_number にセット
39	grow_leaf	葉 0 を大きく (葉 0 のエネルギーが 32 に)
40	grow_leaf	葉 0 を大きく (葉 0 のエネルギーが 64 に)
41	gene_separate	
42	set_phy	レジスタを phy にセット
43	increment	phy = 1
44	shift_left	phy = 2
45	increment	phy = 3
46	shift_left	phy = 6
47	gene_separate	
48	make_flower	ノード 1 から r = 8 , theta = $\frac{\pi}{3}$, phy = π の相対位置に花 0 を作成
49	gene_separate	
50	nop1	補完用テンプレート no-operation
51	nop1	補完用テンプレート no-operation
52	set_flower_number	レジスタを flower_number にセット
53	grow_flower	花 0 を大きく (花 0 のエネルギーが 512 に)
54	grow_flower	花 0 を大きく (花 0 のエネルギーが 1024 に)　　(図 3.17,3.18)
55	breed_flower	花 0 から新しいシードを作る (花 0 のエネルギーは 0 になり、新しいシードのエネルギーが 1024 になる)
56	gene_separate	
57	jump_backward	後方ジャンプ (次の命令は 52:set_flower_number になる)
58	nop0	テンプレート no-operation
59	nop0	テンプレート no-operation
60	gene_separate	

図 3.11: 先祖種の成長 (0:初期状態)

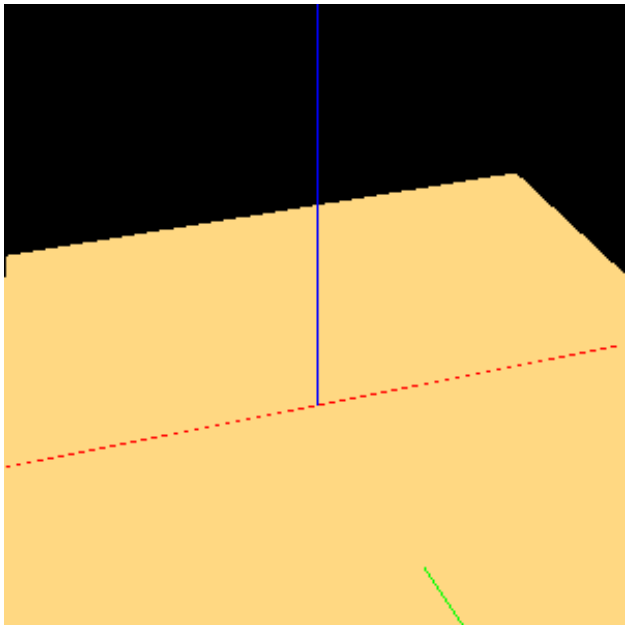

図 3.12: 先祖種の成長 (0:初期状態) 3D

図 3.13: 先祖種の成長 (28:ノードの作成)

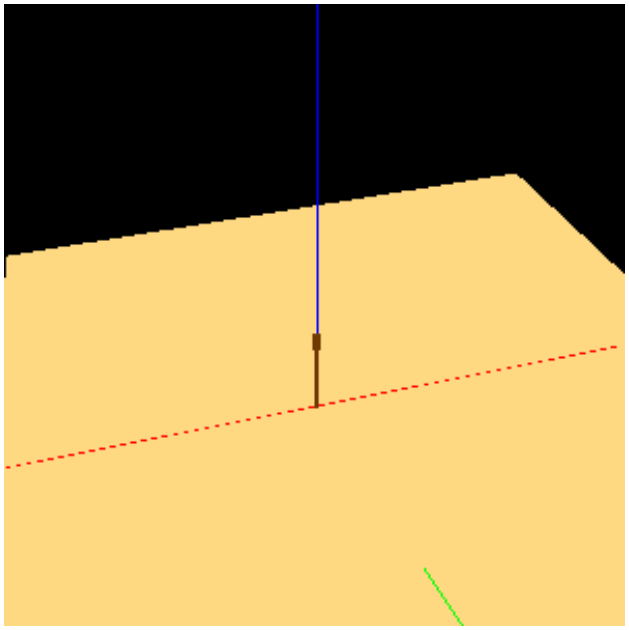

図 3.14: 先祖種の成長 (28:ノードの作成) 3D

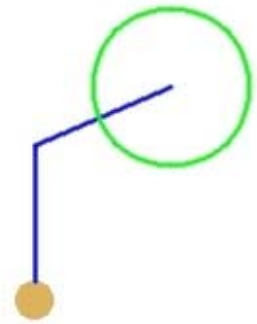

図 3.15: 先祖種の成長 (36:葉の作成)

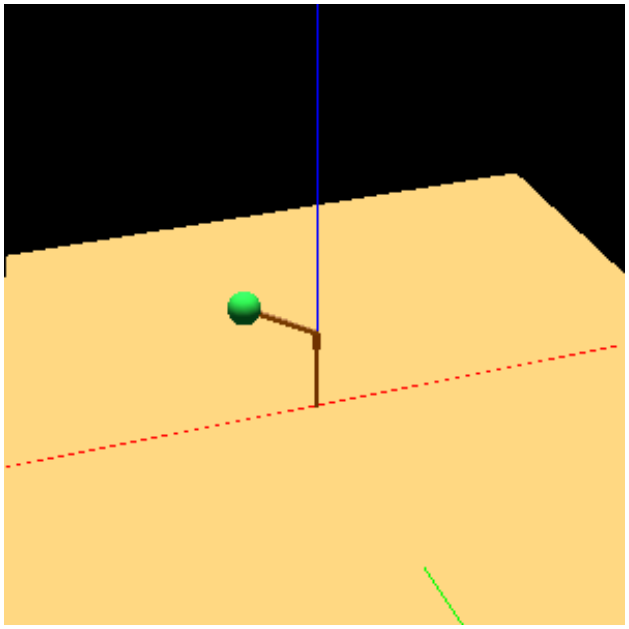

図 3.16: 先祖種の成長 (36:葉の作成) 3D

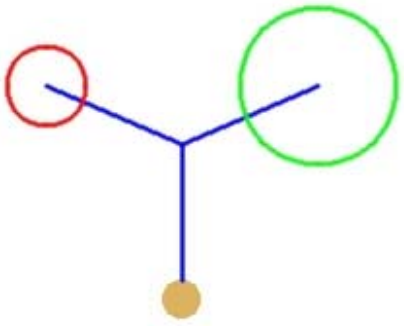

図 3.17: 先祖種の成長 (54: breed_flower 直前)

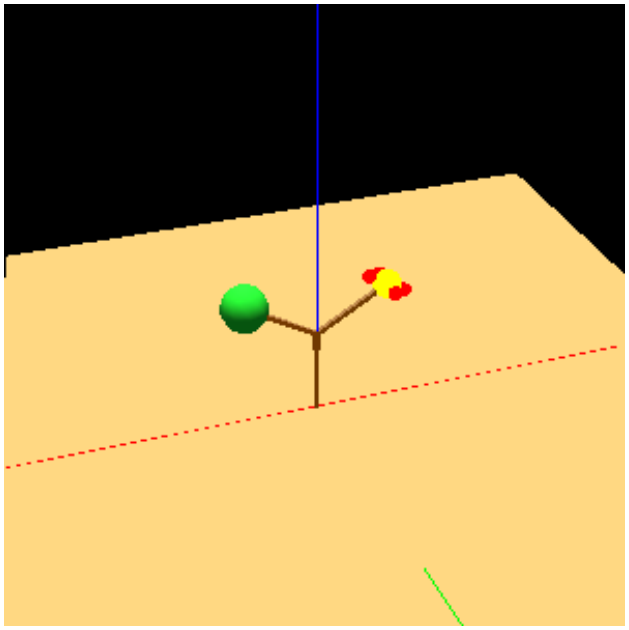

図 3.18: 先祖種の成長 (54: breed_flower 直前) 3D

Daughter Seed の中心位置の決定

つぎに、この花の中心位置から Daughter Seed の中心位置を決定する。まず、花の中心座標を $< x_0, y_0, height >$ とする。ここで、パラメータ $mCPU\_adjust\_height, mCPU\_height\_variance$ を用いて

$$adjust\_height = height \times mCPU\_adjust\_height$$

$$adjust\_variance = adjust\_height \times mCPU\_height\_variance$$

を計算し、平均 $adjust\_height$ 、分散 $adjust\_variance$ の正規分布によって半径 r_d を求める。さらに、$[-\pi, \pi]$ の一様乱数から角度 θ_d を求める。これらから、Daughter Seed の中心位置を、$< x_0 + r_d \times \cos\theta_d, y_0 + r_d \times \sin\theta_d, 0 >$ によって求める。なお、この新しい位置がフィールドの外にでてしまった場合は、breed_flower 命令は失敗であり、新しいシードは誕生しない。

Father Gene の特定

さらに、Mother Seed のジーン Mother Gene と掛け合わせて、Daughter Seed のジーンをつくることになる Father Gene を決定する。まず、パラメータ $mSeedList\_maxFatherDistance$ の平方根を距離の許容値として、breed_flower を行なった花から、許容値以内の距離にある花までの距離を $r_1, r_2, r_3, \ldots, r_N$ とする (図 3.19)。ここで、breed_flower を行なった花と同じシードに属する花は Father の選択対象から外す。残った選択対象から距離の最も小さい花のシードを Father Seed とし、そのジーンを Father Gene とする。またその花までの距離を r_{min} として、あとで利用する Father の寄与率 $father\_per$ を、パラメータ $mGene\_maxFatherEffect$ を使って

$$father\_per = \frac{\frac{1}{r_{min}^2}}{\sum_{i=1}^{N} \frac{1}{r_i^2}} \times mGene\_maxFatherEffect$$

によって計算しておく。

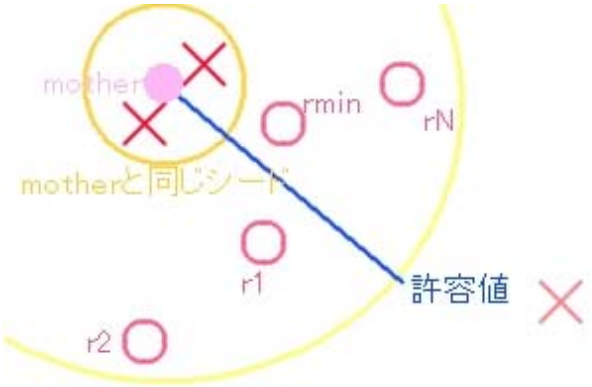

図 3.19: Father Gene の特定

Daughter Gene の作成

新しいシードのジーンである Daughter Gene を、Mother Gene と Father Gene から作成する。この作成の時点で確率的な要素を取り込むことで、突然変異を起こすようにしてある。

まず、Mother Gene と Father Gene のマシン語命令シークエンスを、gene_separate 命令を区切りに分割し、最初に Daughter Gene に Mother Gene の内容を一通りコピーしておく (図 3.20)。次に、gene_separate の区切りの単位ごとに *father_per* の確率で、Mother Gene の内容から、Father Gene の内容へと変更しておく (図 3.21)。

さらに、gene_separate の単位ごとに追加、あるいは削除をパラメータ *mGene_vectorVariation* の確率で行なう。ここで追加するときは、Mother Gene か Fataher Gene のなかからランダムに選ばれた gene_separate の単位が Daughter Gene のランダムな場所に追加され、逆に削除されるときには Daughter Gene のランダムに選択された gene_separate の単位が削除される (図 3.22,3.23)。

その上で、一つの命令を *mGene_sizeVariation* の確率で追加、あるいは削除する。ここで追加するときは、Daughter Gene のランダムに選択された場所にランダムな命令が追加され、逆に削除されるときは Daughter Gene のランダムに選択された命令が削除される (図 3.24,3.25)。

最後に、*mGene_operatorVariation* の確率で、ランダムに選択された一つの命令をランダムな命令に入れ替える (図 3.26)。

このような過程を経て、Daughter Gene は決定される。Daughter Gene は確率的な要素を多分に含むので、Mother Gene と大きくことなることもあり、突然変異となる。

図 3.20: Mother Gene からのコピー

図 3.21: Father Gene からのコピー

ID の作成

ジーンが決まったところで、Daughter Seed と Daughter Gene に ID を付ける。これは、他のシードと一致するかを判定したり、同じジーンの情報を簡単に得られるようにするためのものである。

まず、Daughter Seed の ID は、例えば 61-100-1 のように決定される。具体的には [ジーン (マシン語シーケンス) の長さ - シードが誕生するまでにシステム全体で実行されたマシン語の数 - 特殊ビット] であり、特殊ビットは現在は 1 に固定されている。

Daughter Gene の ID は、Daughter Seed の ID とほぼ同様に決定される。しかし、このシードが誕生した時点ですでに同じマシン語シーケンスを持つシードがフィールド上に存在していたときには、そちらのジーンの ID がこのシードのジーンの ID にも設定される。ただし、過去に同じマシン語シーケンスを持つシードが存在していたとしても、このシードが誕生した時点でフィールド上から失われていると、新しい ID が付けられる。つま

図 3.22: gene_separate 単位の追加

図 3.23: gene_separate 単位の削除

図 3.24: 命令単位の追加

図 3.25: 命令単位の削除

図 3.26: 命令単位の入れ替え

り、Daugher Seed の ID と Daughter Gene の ID が一致するのは、Daughter Seed がその誕生時点でフィールド上に存在しないジーンをもって誕生したときである。

Daughter Seed の作成

ここまでで、ジーンや中心位置、ID などが決まったのでこれらの情報をもとに Daughter Seed を作成する (図 3.27)。Daugther Seed の最初に持つエネルギーは、breed_flower を行なった花に蓄えられていたエネルギーであり、この結果もとの花はエネルギーを失いその半径が 0 になる。これ以降、Daughter Seed は、Mother Seed と同じように CPU の実行を繰り返し、自己複製をしていくことになる。

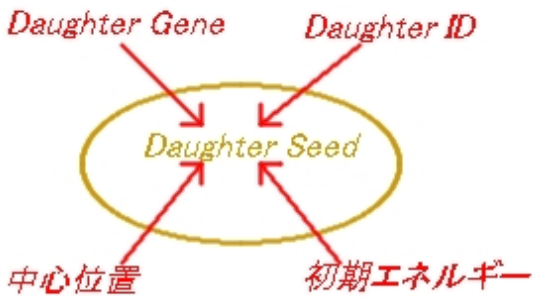

図 3.27: Daughter Seed の作成のための要素

3.9　評価値とシードの選択

シードそれぞれに計算される評価値は、すべてのシードの CPU の実行が終了した時点で計算される。まず、評価値自体は表 3.12 の計算式によって決定される。

表 3.12: シードの評価値の決め方

$$
\text{評価値} = \begin{cases} \text{breed\_flower に成功した回数 } +1 & \text{(標準)} \\ -1 & \text{(エネルギーを使い果たしたとき)} \\ -1 & \text{(breed\_flower に成功しないで ある一定の命令数を実行したとき)} \end{cases}
$$

このように計算された評価値によって、次のシードの CPU の実行順番が決定される。つまり、評価値の高いシードほど次の CPU 実行が早く回ってくるのである。なお、３番目の計算方法である一定の命令数に達したときとは、シードのジーンの長さにパラメータ $mSeed\_turn\_effect$ をかけた値の回数 CPU を実行するまでに breed_flower

命令に成功しなかった場合に、評価値を −1 にするということである。たとえば、 ID:61-0-1 の場合は、仮に $mSeed\_turn\_effect = 1.5$ とすれば、$61 \times 1.5 = 91.5$ 回の CPU の実行回数で breed_flower 命令を成功させないと評価値は 92 回目の実行の後に −1 になる。

また、フィールド上にあまりにたくさんのシードが存在するようになると、新しいシードの成長が行われなくなってしまうため、途中からシードの選択を行う。ここで選ばれなかったシードは、フィールド上から削除されることになる。

まず、シードの選択が行われるケースとして考えられるのが、フィールドが満杯に近くなってきたときである。フィールドのところでも説明したように、葉や花に日光が吸収されると、地上に到達する日光の量が減ることになる。ここで、上空から降り注いでいる日光と途中で吸収される日光との比が パラメータ *mField_capacity* を超えるようになったところでシードの選択を行うことにする。仮に上空から降り注いでいる日光の量を 100 、途中で吸収される日光が 60 、 パラメータ $mField\_capacity = 0.50$ 、 シードの数を 1000 とおくと、$\frac{60}{100} > 0.50$ より、シードの選択が行われることになり、$(\frac{60}{100} - 0.50) \times 1000 = 100$ 個のシードが削除されることになる。ここでは確率的に 100 個のシードを決定する。それぞれのシードに評価値の高さによって決定される閾値に対して、それぞれのシードで乱数を計算し、この閾値を下回った場合にシードが削除される。たとえば、閾値が 0.2 のシードに対して $[0,1]$ 区間の一様乱数で 0.25 という値なら削除されないが 0.18 なら削除されるという具合である。もちろんそれぞれのシードにおいて一様乱数は独立に計算される。それぞれのシードの閾値は評価値から算出されるが、これは評価値が高ければ閾値が低く、評価値が低ければ高く計算され、もっとも評価値の低いものの閾値がもっとも評価の高いものの閾値の、パラメータ *mSeedList_delete_fractional* 倍になるように計算される。これらのランダムな削除によって確率的な平均として 100 個のシードが削除されることになる。

シードが削除されるケースとして次に考えられるのは、すべてのシードが CPU の計算を終了して評価値を計算し終わったときである。このときに評価値が 0 未満のシードは自動的に削除される。このことと評価値の決め方から、エネルギーを使い果たしたシードやいつまでたっても breed_flower 命令を成功できないシードが削除されることになる。

シードが削除されるもう一つのケースとしてランダムな削除がある。これも、すべての CPU が実行を終えたときに行われる。このケースはフィールド上に パラメータ *mSeedList_min_random_delete_number* を超えるシードが存在するときにときにだけ実行され、それぞれのシードが パラメータ *mSeedList_random_delete_probability* の確率で削除される。

このようにシードの選択にはいろいろなケースがあるのだが、それらのケースがどのような順番で行われるかというのは選択に大きな意味を持っている。実際の計算では、すべてのシードの CPU が実行されると評価値が決定され、上で述べたケースの順でシードの選択が実行される。なお、新しく breed_flower 命令で誕生してすぐの CPU を一度も実行していないシードは、シードの選択にはまったく関与せず、削除の対象にはならない。

3.10　オペレーティングシステム

オペレーティングシステムのもっとも大きな役割は、フィールドとシード全体の管理である (図 3.28)。ここでは、オペレーティングシステムがどのような動きをしているのかを見ることにする。オペレーティングシステムは基本的に表 3.13 のようなサイクルを繰り返している。

これらのサイクルのうち、SUN_LIGHT フェーズから次の SUN_LIGHT フェーズまでを 1 ターンとして考えることができる。

表 3.13: オペレーティングシステムの行うサイクル

SUN_LIGHT	：	日光の計算
	↓	
EXECUTE	：	シードの CPU の実行
	↓	
AFTER_EXECUTE	：	情報収集とシードの選択
	↓	
SUN_LIGHT	：	日光の計算

まず、SUN_LIGHT のフェーズでは、フィールド上の日光についての計算と、それに伴うシードに貯えられているエネルギーの再計算をし、フィールドキャパシティーなどのフィールドについての情報収集が行われる。

つぎに、EXECUTE のフェーズでは、それぞれのシードの CPU が実行される。ここでは、評価値の高いシードから実行され、それぞれのシードの CPU は 1 ターンに 1 回のみ実行される。

AFTER_EXECUTE のフェーズでは、まずシードの維持エネルギーが計算される。シードはこのターンでの ABuffer に合わせて維持エネルギーを計算し、現在のエネルギーを減少させる。さらに、シードの評価値が計算され、情報収集を行いログファイルが生成される。ここで収集される情報としてはたとえば評価値の平均値だったり、そのターンでのシードの数やジーンの数である。また、シードの評価値の計算に伴ってシードの選択が行われる。その後にこのターンで breed_flower 命令で生まれたシードがシードのリストへと登録されるのである。

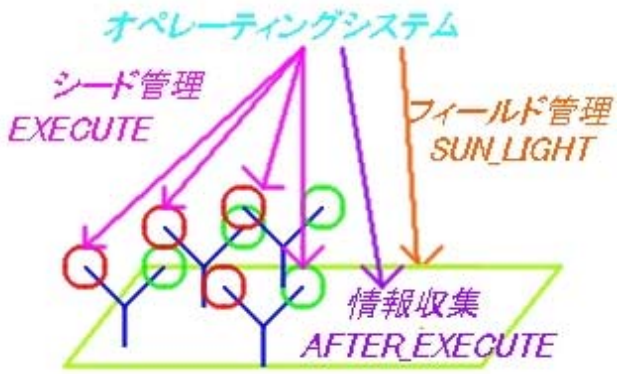

図 3.28: オペレーティングシステムの概念

3.11　進化はどのようにして起こるか

新しい種が生まれるチャンスは、breed_flower 命令のときだけである。このときに確率的な突然変異によって Daughter Gene と Mother Gene とが異なると、新しい種は誕生する。

具体的に先祖種 ID:61-0-1 からどのような突然変異があるかを考えてみることにする。 ID:61-0-1 のジーンは表 3.10, 3.11 に載せてあるのだが、ここで突然変異としてのランダムな命令の入れ替えで (図 3.26)、25 番目の shift_left が increment になったとすると、ノードの作成などに使われているレジスタ r が 8 から 6 に 変化するこ

とになる。つまり、このことによって新しい種ではシードとノードとの距離や、ノードから葉や花への距離が小さくなり (つなぐ枝が短くなっている)、全体的にコンパクトになる (図 3.29)。

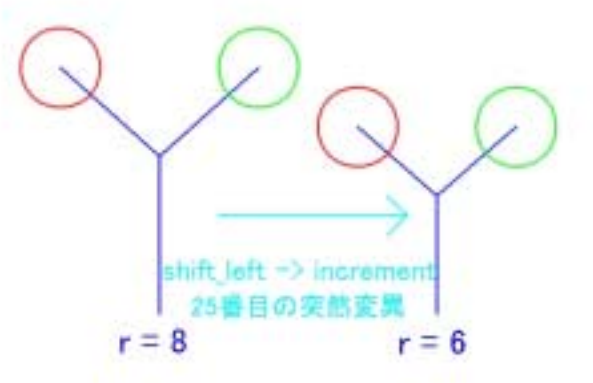

図 3.29: 突然変化による進化

ほかにも、make_flower 命令の重複などによって複数の花を持ったりする種など、いろいろな種が進化によって誕生する。

3.12　結果として得られる情報

ここでは、オペレーティングシステムの方が情報収集し、ログファイルとして結果を出力している情報について、重要なものをログファイルごとに見ていくことにする。

seed_num.log

それぞれのターンで、どれだけのシード数があったかを記録するログファイル。

gene_num.log

それぞれのターンで、どれだけのジーン数があったかを記録するログファイル。

ave_eva.log

平均評価値を記録するログファイル。

ave_size.log

ジーン (マシン語シークエンス) の長さの平均を記録するログファイル。

ave_turn.log

それぞれのシードが誕生してから何ターン、フィールド上にいるかの平均 (平均寿命) を記録するログファイル。

fie_cap.log

それぞれのターンにおける、フィールドのキャパシティ (途中で葉や花に吸収される日光の量が上空から降り注ぐ日光の量に占める割り合い) を記録するログファイル。

edge.log

そのターンでフィールド上にいるシードが枝のために使用したエネルギーの量を記録しておくログファイル。ここで計算される量は、シードが誕生してからこのターンまでに、set_edge_number(枝の維持を含む), cut_edge, make_node の命令にどれだけのエネルギーを使ったかである。

leaf.log

そのターンでフィールド上にいるシードが葉のために使用したエネルギーの量を記録しておくログファイル。ここで計算される量は、シードが誕生してからこのターンまでに、set_leaf_number(葉の維持を含む), set_leaf_energy, make_leaf, grow_leaf の命令にどれだけのエネルギーを使ったかである。

flower.log

そのターンでフィールド上にいるシードが花のために使用したエネルギーの量を記録しておくログファイル。ここで計算される量は、シードが誕生してからこのターンまでに、set_flower_number(花の維持を含む), set_flower_energy, make_flower, grow_flower, breed_flower の命令にどれだけのエネルギーを使ったかである。

per_edge.log

edge.log, leaf.log, flower.log の３つのパーツに対してのエネルギーの消費のうち edge.log の値が何パーセントを示すかを記録するログファイル。

per_leaf.log

edge.log, leaf.log, flower.log の３つのパーツに対してのエネルギーの消費のうち leaf.log の値が何パーセントを示すかを記録するログファイル。

per_flower.log

edge.log, leaf.log, flower.log の 3 つのパーツに対してのエネルギーの消費のうち flower.log の値が何パーセントを示すかを記録するログファイル。なお、per_edge.log, per_leaf.log, per_flower.log の 3 つのファイルの値を加えると、 100 になる。

あと、この章の参考として、設定可能なパラメータのデフォルト値を表 3.14 にまとめておく。

表 3.14: パラメータとそのデフォルト値

パラメータ名	デフォルト値	簡易説明
mSeed_volume	10.0	シードの体積
mSeed_calorie	1.0	シードのカロリー
mSeed_keep_calorie	1.0	シードのターンごとの維持カロリー
mEdge_calorie	1.0	枝の単位体積あたりのカロリー
mEdge_keep_calorie	2.0	枝の単位カロリーあたりのターンごとの維持カロリー
mEdge_radius	0.5	枝の半径
mLeaf_calorie	1.0	葉の単位体積あたりのカロリー
mLeaf_keep_calorie	1.0	葉の単位カロリーあたりのターンごとの維持カロリー
mFlower_calorie	100	花の単位体積あたりのカロリー
mFlower_keep_calorie	0.01	花の単位カロリーあたりのターンごとの維持カロリー
mField_min_x	-40.0	フィールドの x 軸方向の最小値
mField_min_y	-40.0	フィールドの y 軸方向の最小値
mField_N	1000	フィールド内部の格子の数
mField_haba	0.08	フィールド内部の格子の幅
mField_daylight	0.1286	フィールドの単位格子あたりの日光の量
mField_downsize	0.7	一つの要素で吸収される日光の割合
mField_capacity	0.5	シード選択を行うのか判定をするための フィールドのキャパシティ
mCPU_energy_base	1.0	命令をひとつ実行したときに使用する基本エネルギー
mCPU_adjust_phy	0.5236 ($\frac{\pi}{6}$)	レジスタ phy を使う際の補正値
mCPU_adjust_theta	0.5236 ($\frac{\pi}{6}$)	レジスタ theta を使う際の補正値
mCPU_adjust_height	1.5	次のシードの飛ぶ範囲の平均値を決定する要因
mCPU_height_variance	1.0	次のシードの飛ぶ範囲の分散を決定する要因
mGene_vectorVariation	0.02	gene_separate 単位の命令の挿入削除が起こる確率
mGene_sizeVariation	0.02	1 つの命令の挿入削除が起こる確率
mGene_operatorVariation	0.05	1 つの命令の突然変異が起こる確率
mSeedList_delete_factorial	2.0	シードリストがいっぱいになったときに 削除する確率を決定する要因
mSeedList_random_delete_probability	0.003	シードリストの中のランダムに削除される個体の割合
mSeedList_min_random_delete_number	100	シード数がこの数を超えるとランダムな削除が行われる
mGene_maxFatherEffect	0.5	Father Gene の Daughter Gene への最大寄与率
mSeedList_max FatherDistance	100.0	Father Gene の候補になるための距離の限界値の 2 乗の値
mSeedList_min_r	0.001	利用できるレジスタ r の最小値 (この値よりも r が小さいと make_node, make_leaf, make_flower を行わない)
mPartsList_tolerance_energy	10.0	現在のエネルギーとこの値の積の値よりも 大きいエネルギーは一度には使えない

第4章　植物系の進化のシミュレーションとその解析

この章では、次の 3 段階にわたってシミュレーションとその解析を行い、設定した進化の方向に環境が変わっていくかどうかを調べる。

1. デフォルトパラメータによるシミュレーションの時間軸に沿った観測

   まず、設定するパラメータをデフォルトパラメータに固定し、時間が経つにつれて、どのような種が現れどのようにフィールドが変化していくのかを観測し、シミュレーションでどのような進化が起こるかを調べる。

2. 分散分析による複数回のシミュレーションの解析

   パラメータを変えながらシミュレーションを複数回行うことで情報収集をし、パラメータとそれに伴う進化の方向の変化を分散分析で解析する。

3. 進化の方向を設定した確認シミュレーション

   2. の分散分析で解析したパラメータと進化の方向の関係を利用することで、実際に設定した進化の方向 (多様な種の存在する環境) に近い環境を作れるのかを観測する。

なお、ここからシミュレーションとその解析では、表 4.1 に載せてあるパラメータのみを変動させることにし、その他はデフォルトのままで固定することにする。

表 4.1: 変動するパラメータとそのデフォルト値

パラメータ名	デフォルト値	
mField_daylight	0.1286	フィールドの単位格子あたりの日光の量
mEdge_keep_calorie	2.0	枝の単位カロリーあたりのターンごとの維持カロリー
mLeaf_keep_calorie	1.0	葉の単位カロリーあたりのターンごとの維持カロリー
mFlower_keep_calorie	0.01	花の単位カロリーあたりのターンごとの維持カロリー

また、本論文ではシミュレーションを行うにあたりコンピュータ上でプログラムを組んである。実装は Visual C++ で行っている。実装に関しては、 Tierra のコードを使わずにすべて新しく書き起こしたものである。このことは、今回の植物のモデル化のところで中間表現を利用していることに起因している。つまり、 Tierra では遺伝型と表現型が一致するのに対して、本論文のモデルでは遺伝型と表現型が異なるために、そのまま Tierra のプログラムを応用するよりも新しく書き直した方が容易であるためである。また、フィールドの様子やそれぞれの種の形状などを 3D グラフィックスで表現するために、OpenGL を用いている。

4.1　デフォルトパラメータによるシミュレーションの時間軸に沿った観測

4.1.1　代表的な種とフィールドの様子から見る推移

まず最初に、デフォルトのパラメータにおいて、時間経過とともにどのような種が生まれ、どのようにフィールドが変化していくのかを見て行くことにする。

ここでは、fie_cap.log、つまりフィールド上でどれだけの日光が葉や花に吸収されているか、の変化に合わせて見て行くことにする。まず、fie_cap.log のグラフとして、

1. 0 ターンから　2500 ターン (図 4.1)
2. 0 ターンから 10000 ターン (図 4.2)

の 2 つのスケールを考え、それぞれのグラフで比較的変動の少ない期間から特徴のあるターンを選択し、一覧にまとめたのが表 4.2 である。ターンの定義については、3.10 節に述べておいた。なお、 10000 ターン以上のところはパラメータ *mField_capacity*(デフォルト値は 0.5) とシードとの選択の関係 (参照:3.9 節) のために、fie_cap.log グラフ自体には変動があまり見られないが、特徴的な種が現れた所を抜粋してある。

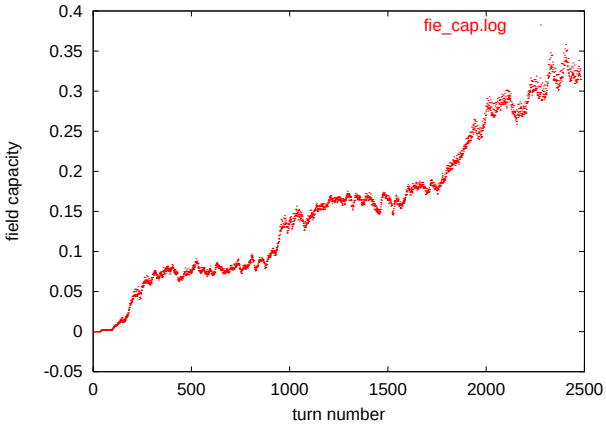

図 4.1: 0 ターンから 2500 ターンの fie_cap.log のグラフ

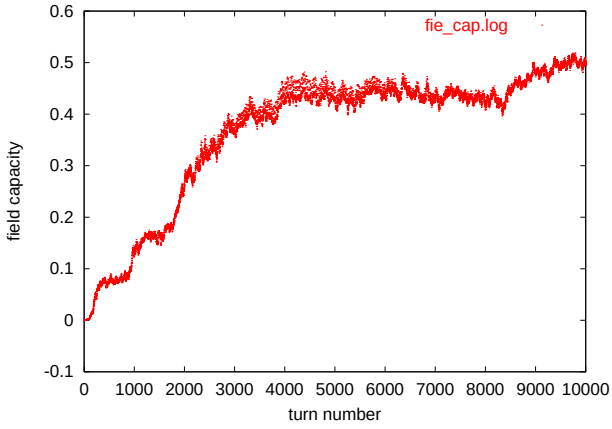

図 4.2:　0 ターンから 10000 ターンの fie_cap.log のグラフ

ここから先は、表 4.2 の分類に従って、そのターンでのフィールドの状態とどのような種が多かったかを見て行くことにする。

54 ターン : 最初の個体の成長

まず、最初の個体の成長が起こる。最初の個体は、先祖種 (61-0-1 , 図 4.4) であり、この個体のみがフィールドの中央に存在している状態である。(フィールドの様子は、図 4.12, 4.13)

426 ターン : 先祖種の広がり

最初の個体から breed_flower を繰り返していくことで、だんだんと先祖種がフィールド上に広がっていく過程を見て取ることができる。(フィールドの様子は、図 4.14, 4.15)

表 4.2: 特徴のあるターン

	ターン	特徴
1	54	最初の個体の成長
2	426	先祖種の広がり
3	1126	θ の変化
4	3067	背の高い個体の登場
5	4522	花の増加
6	9800	フィールドが限界に
7	65913	複雑な種へ
8	165624	より複雑な種へ

1126 ターン : θ の変化

先祖種がフィールド上にある程度広がると、個体数の多い先祖種との空間的な衝突をさけることでフィールド上に存在する種が現れるようになる。その始まりともいうべきなのが、61-127029-1 である (図 4.6)。先祖種が $\theta = 2$ 、つまり垂直方向から $\frac{\pi}{3}$ のところに葉と花をつけていた (パラメータ *mCPU_ad just_theta* による補正がある、参照 3.5 節 の 1A:make_flower の項目) のに対して、この種では $\theta = 1$ 、つまり垂直方向から $\frac{\pi}{6}$ のところに葉と花をつけることになる。

先祖種よりも高い葉と花をつける位置を変えることによって、先祖種との空間的衝突を避け、さらに先祖種以上に多くの日光を受けることができるようになる。61-127029-1 は次第に個体数を増やし、先祖種は受け取れる日光の減少から個体数を減らしていくことになる。(フィールドの様子は、図 4.16, 4.17)

3067 ターン : 背の高い個体の登場

65-127029-1 は θ を変化させることで広がってきたのだが、似たような種類だけではやはり空間的な衝突は時間とともに増えることになる。そのような状況をクリアすることができる種が 65-1061377-1 である (図 4.7)。

65-1061377-1 は 61-127029-1 と比べて、葉や花をつけるためのノードが高いところにある。つまり背が高いのである。ここでの 65-1061377-1 と 61-127029-1 の間の関係は、1126 ターンの 61-127029-1 と先祖種の間の関係と同じである。つまり、空間的な余裕のある 65-1061377-1 は個体数を増やし、その影となる 61-127029-1 は衰退することになる。(フィールドの様子は、図 4.18, 4.19)

4522 ターン : 花の増加

4522 ターンではもっとも個体数のある種 (64-872247-1) は、3067 ターンと同じような傾向を持つ (図 4.8)。

しかし、フィールド上の様子はまったく異なる。花が目立つのである。このことは、背が高くなった種がひととおり空間を埋め尽くしたことを意味している。3067 ターンの 65-106137-1 や、このターンの 64-872247-1 などがフィールド上を埋め尽くしたことによって、これらの背の高い種の花が目に付くのである。(フィールドの様子は、図 4.20, 4.21)

9800 ターン : フィールドが限界に

9800 ターンまでくると進化の方向は一変する。それまで、61-0-1 , 61-127029-1 , 65-1061377-1 , 64-872247-1 とだんだんと背を高くしていったのだが、ここに来てもっと個体数を増やした種は、65-10548967-1 という比較的背の低い個体である (図 4.9)。このことは、フィールドキャパシティーの限界を超えてシードの選択がより厳しくなったため、単純に背の高さだけでない進化が起こっていることを意味している。

65-10548967-1 には、今までにない特徴が 2 つある。まず一つ目は、コンパクトであるということである。65-1061377-1 (図 4.7) のような背の高い種が隣にあってもぶつかることはないし、自分と同じ高さの種が隣にきても水平的な空間もコンパクトなためにぶつかりにくいのである。とくに自分と同じ高さの種とぶつかりにくいということは重要であり、これによって種を変えることなく個体数を増やしていくことが可能になる。

もう一つの大きな特徴は、葉が 2 枚あるということである。このことによって、より多くの日光を受けられるということはもちろんであるが、葉が 2 枚あることの本当の効果は別のところにある。葉が 2 枚あるということは、たとえどちらか一方が空間的な衝突によって葉をつけられないという状態になっても、もう一枚でその状況を克服できるということである。(フィールドの様子は、図 4.22, 4.23)

65913 ターン : 複雑な種へ

65913 ターンでもっとも個体数を増やす種は 76-10534514-1 である (図 4.10)。

この種は、背の高い 65-1061377-1 と葉を 2 枚もつ 65-10548967-1 の両方の性質をあわせもっている。ただし、この種はその両方の性質を単純に足し合わせただけではない。たとえば 65-1061377-1 は葉や花がつくノードの高さを高くすることで全体の高さを高くしていたが、76-10534514-1 ではこれをさらに高くしてより高いところに葉を付けられるようにしている。また、65-10548967-1 は葉が 2 枚で花が 1 つという構成だったが、76-10534514-1 では花を 2 つにしている。この 2 つの花のうち実際にエネルギーを貯えて breed_flower を行うのはどちらか一方のみで、もう片方は breed_flower を行わない。つまり花についても保険的な位置を確保し、より確実に breed_flower が行えるようにしているのである。

また、このあたりまでくると個体数を増やす種のジーンの長さが増加してくる。先祖種では 61 であったのがここでは 76 にまでなっている。基本的にジーンが短い方が breed_flower を行うまでの命令数が少ないので短時間で複製を行うことができるのだが、フィールドの限界などから自然選択が厳しくなると、単純に複製の速さだけでなく、花の個数を増やすなど確実に複製を行うためのしくみが重要になってくるのである。(フィールドの様子は、図 4.24, 4.25)

165624 ターン : より複雑な種へ

今回のシミュレーションでは、コンピュータ上の計算時間の関係上、このターンまで行ってある。ここでもっとも個体数の多い種であったのは、88-23248225-1 であった (図 4.11)。

この種は、76-10534514-1 よりもさらに複雑になり、葉は 4 枚、花になる候補の部分は 2 つあり、より確実に breed_flower できるようになっている。また、今までの種よりも葉や花の位置が高いのでより効果的に他の種を影にすることが可能になっているのである。(フィールドの様子は、図 4.26, 4.27)

ここで、フィールド上に存在する種のジーンの長さとその個体数をグラフにしたのが、図 4.3 である。

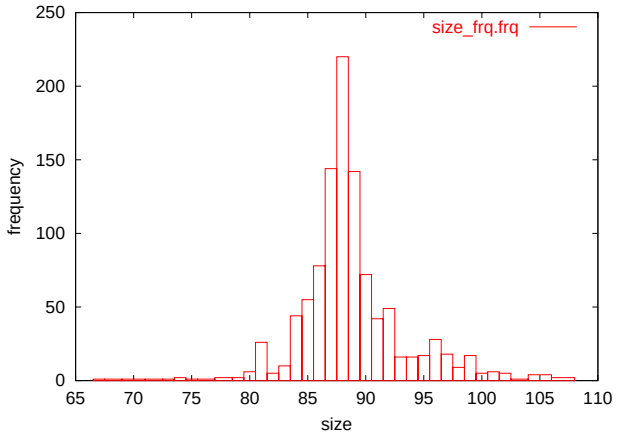

図 4.3: ジーンの長さと個体数

この図を見るとわかるように、先祖種のように 61 といった短いジーンを持った個体は既に存在してなく、88 といった長いジーンを持った個体が多い。さらに 88 よりも短いジーンのところよりも、88 より長いジーンのところに偏りがある。これは、このターンがより複雑に進化していく過程の途中であることを示唆している、と考えることができる。

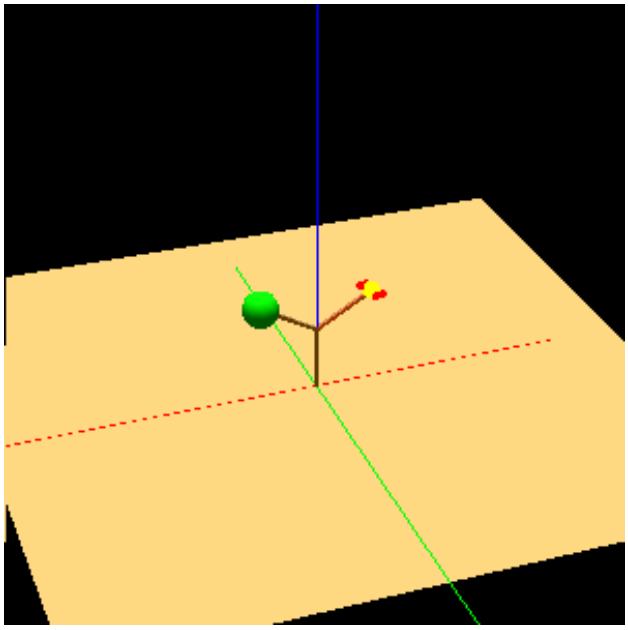

図 4.4: 54 ターン : 特徴的な種 (61-0-1)

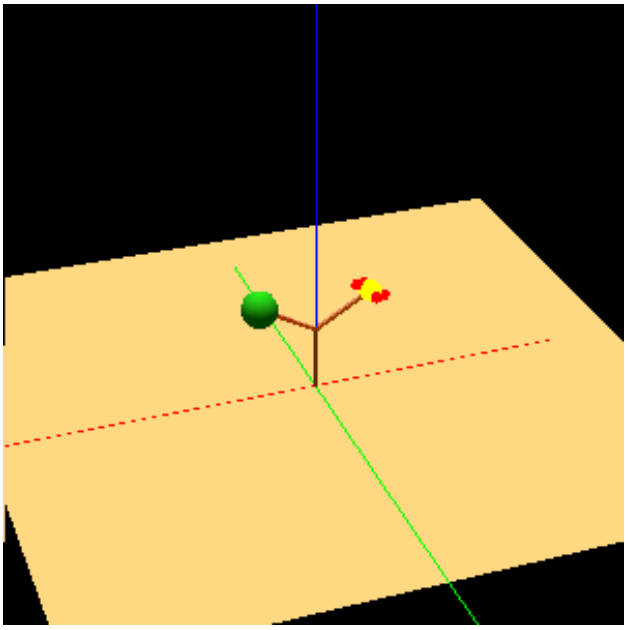

図 4.5: 426 ターン : 特徴的な種 (61-0-1)

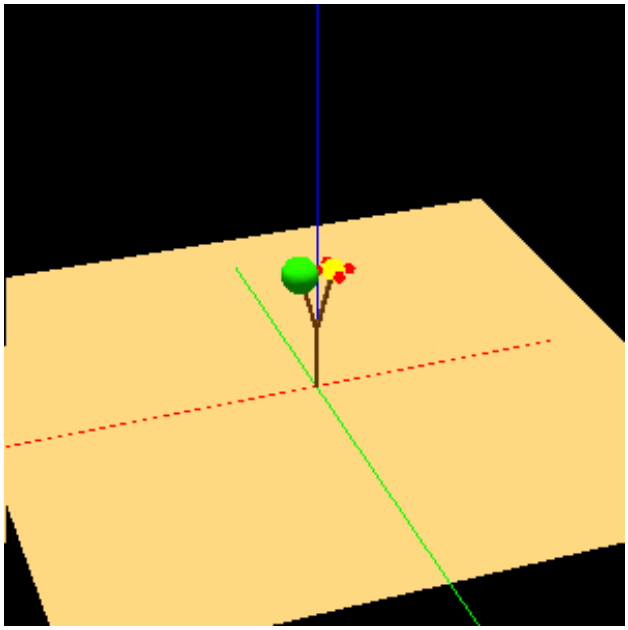

図 4.6: 1126 ターン : 特徴的な種 (61-127029-1)

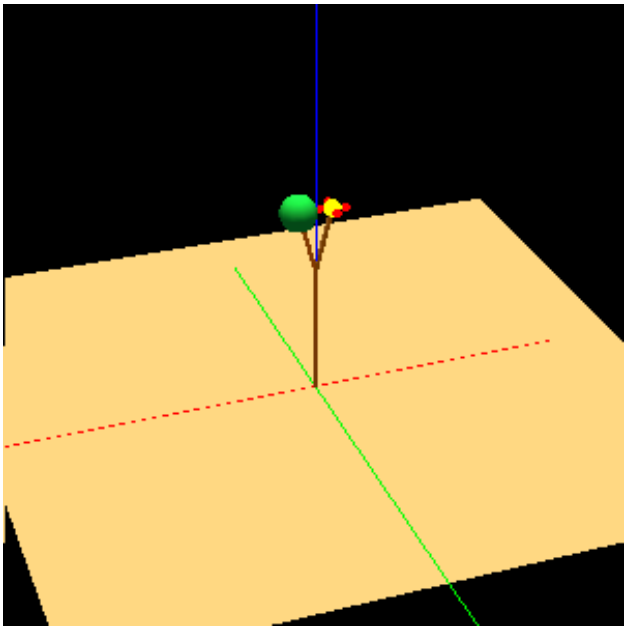

図 4.7: 3067 ターン : 特徴的な種 (65-1061377-1)

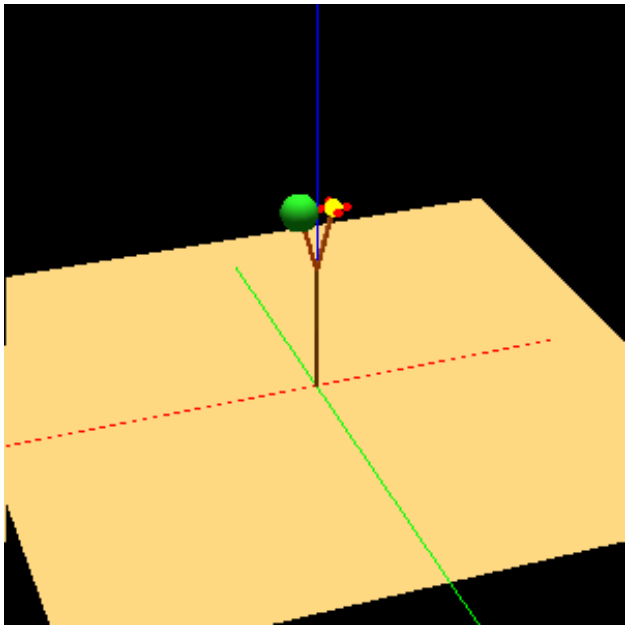

図 4.8: 4522 ターン : 特徴的な種 (64-872247-1)

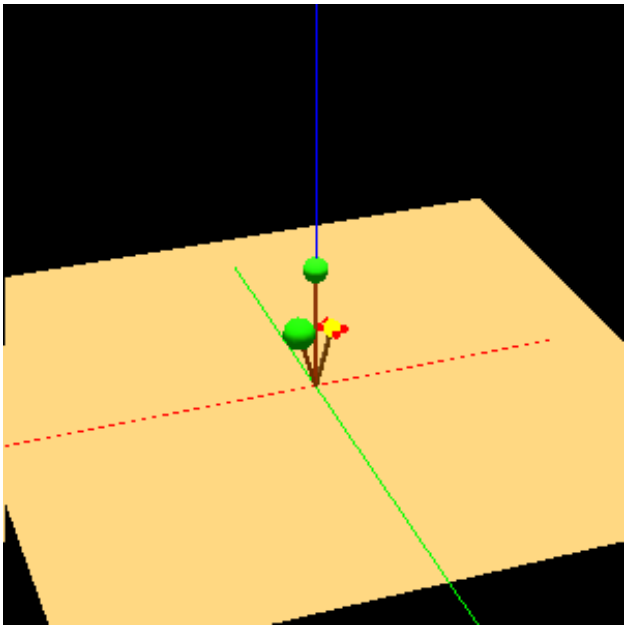

図 4.9: 9800 ターン : 特徴的な種 (65-10548967-1)

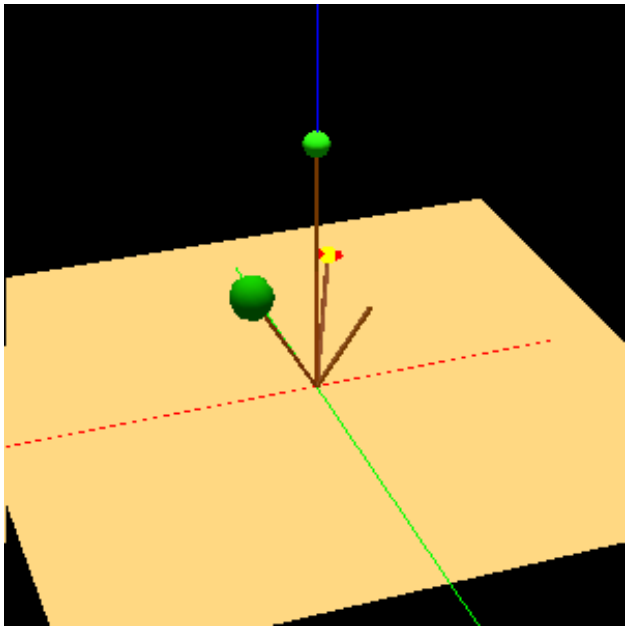

図 4.10: 65913 ターン : 特徴的な種 (76-10534514-1)

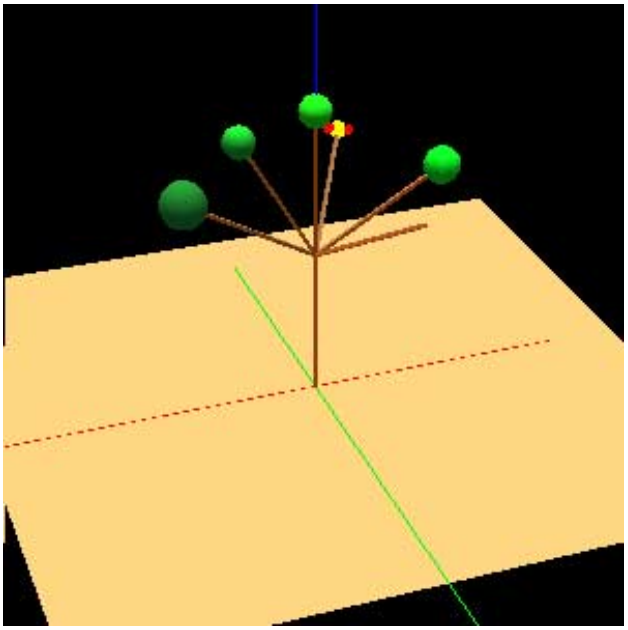

図 4.11: 165624 ターン : 特徴的な種 (88-23248225-1)

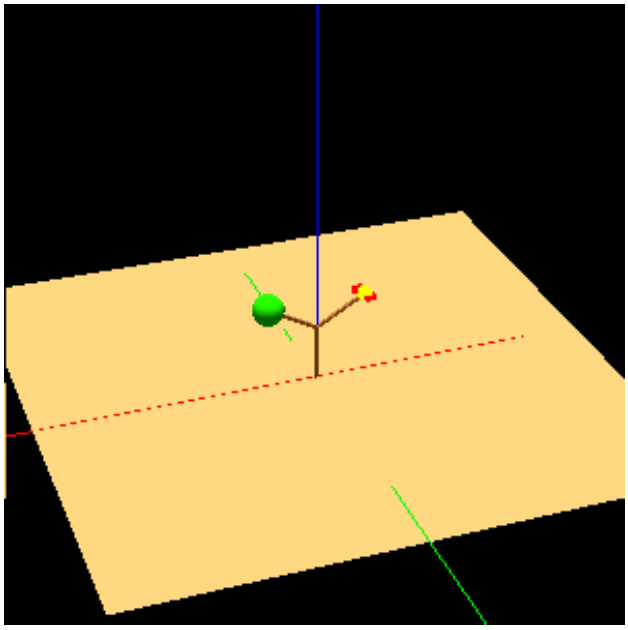

図 4.12: 54 ターン : フィールドの様子

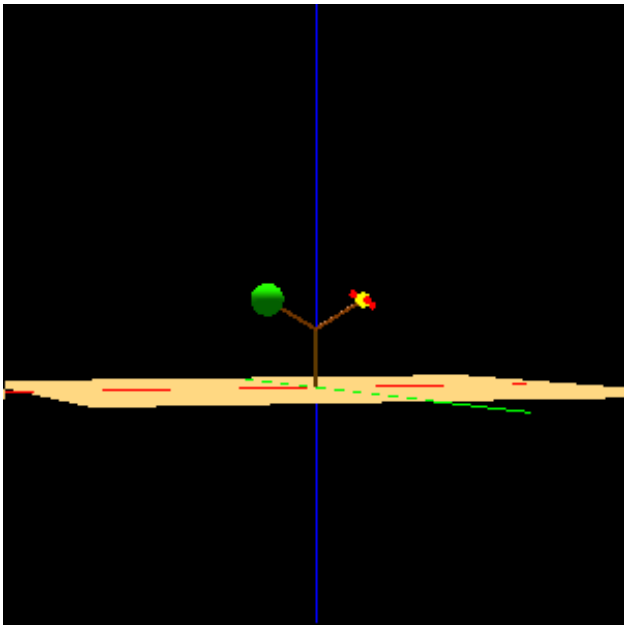

図 4.13: 54 ターン : フィールドの様子 (真横)

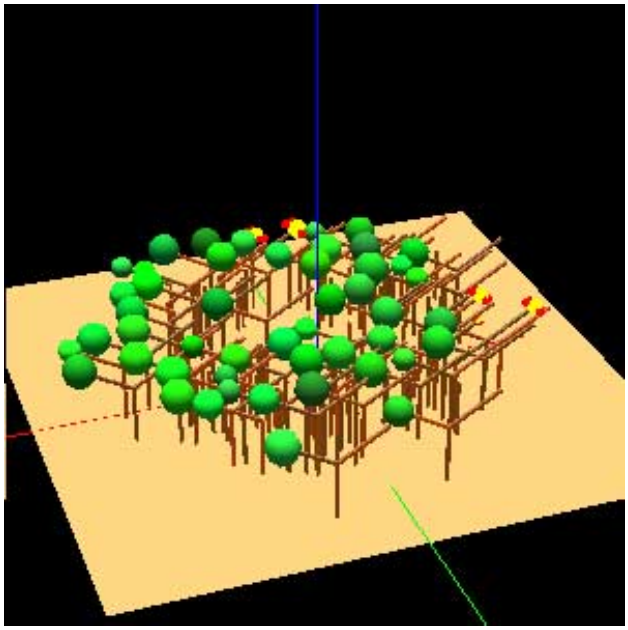

図 4.14: 426 ターン : フィールドの様子

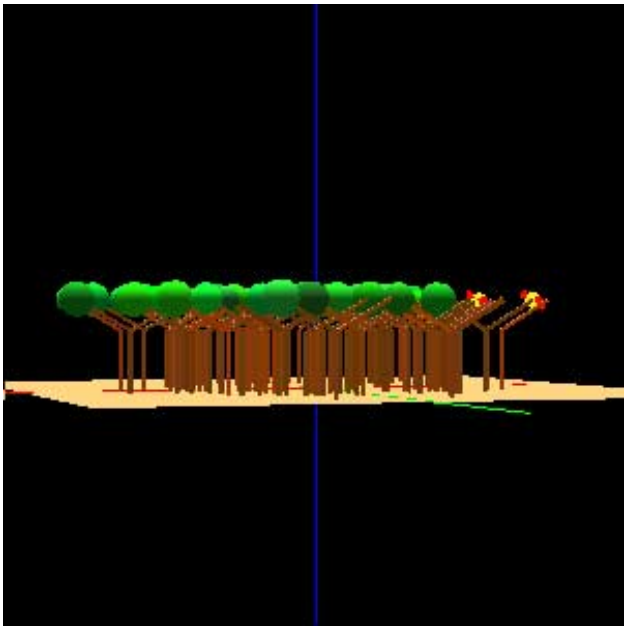

図 4.15: 426 ターン : フィールドの様子 (真横)

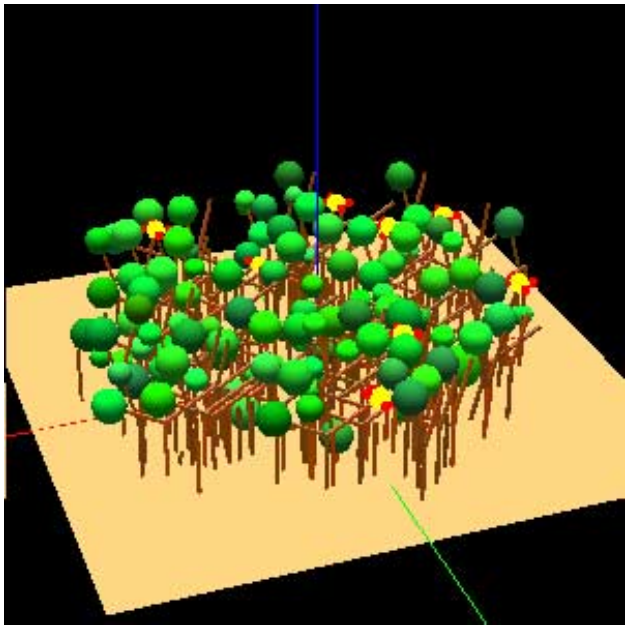

図 4.16: 1126 ターン：フィールドの様子

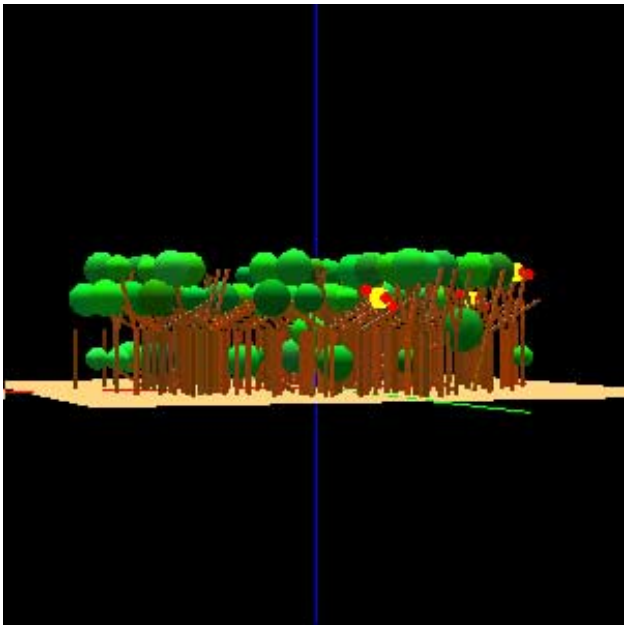

図 4.17: 1126 ターン：フィールドの様子 (真横)

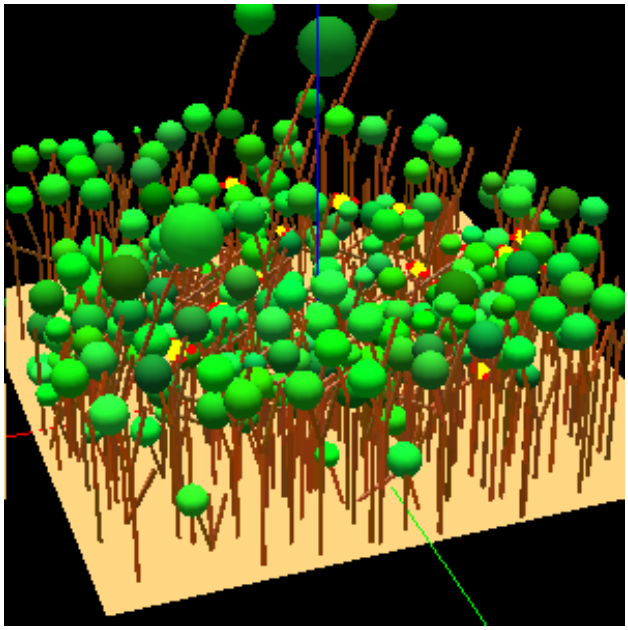

図 4.18: 3067 ターン：フィールドの様子

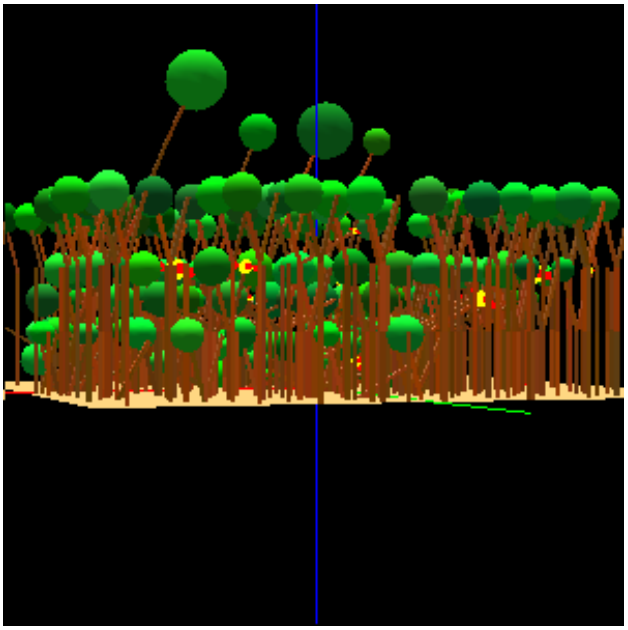

図 4.19: 3067 ターン：フィールドの様子 (真横)

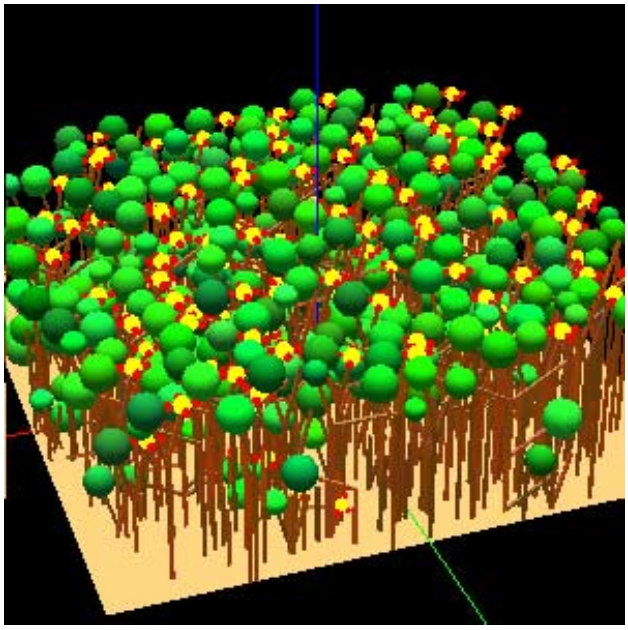

図 4.20: 4522 ターン：フィールドの様子

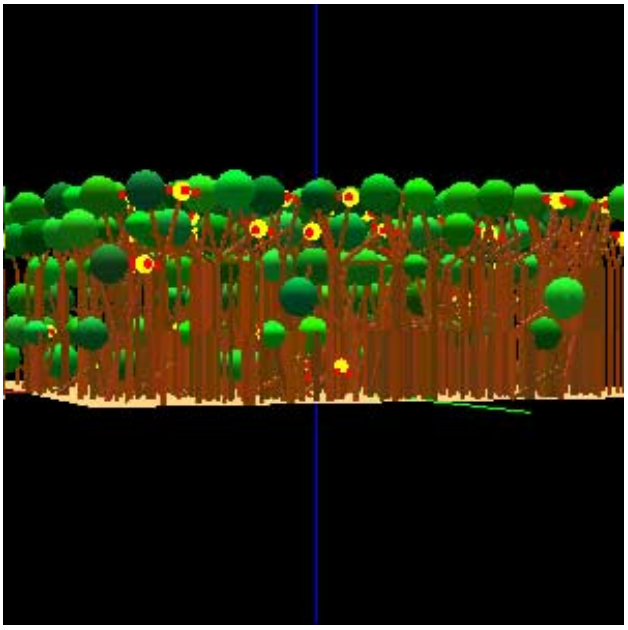

図 4.21: 4522 ターン：フィールドの様子 (真横)

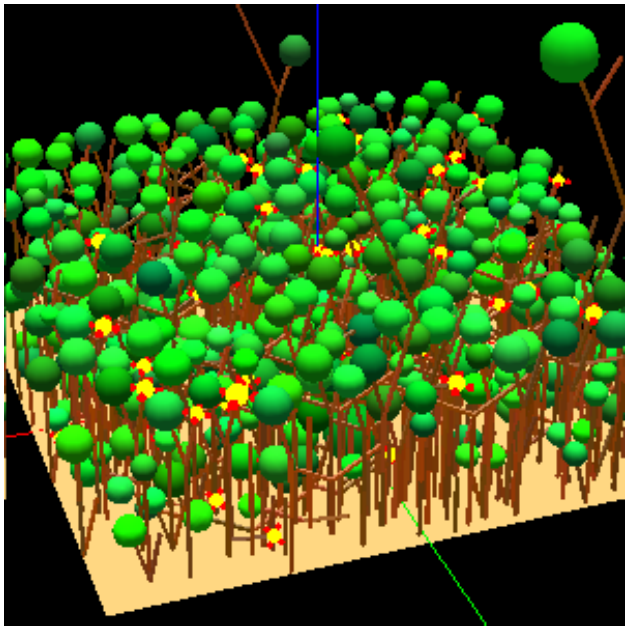

図 4.22: 9800 ターン：フィールドの様子

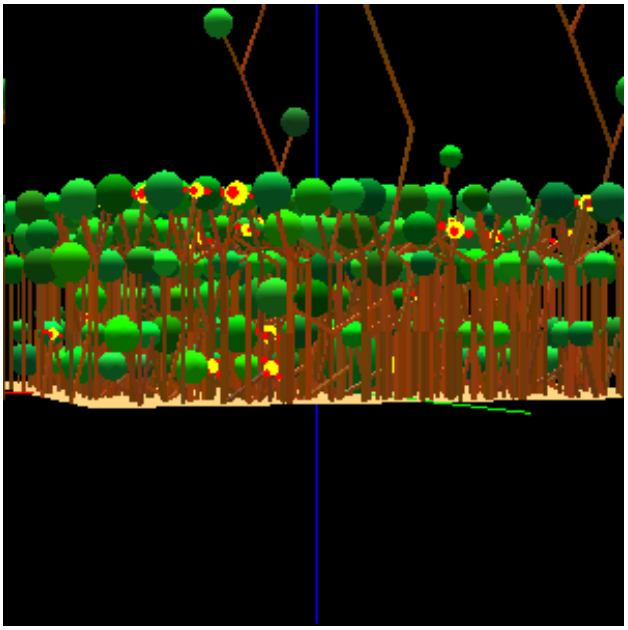

図 4.23: 9800 ターン：フィールドの様子 (真横)

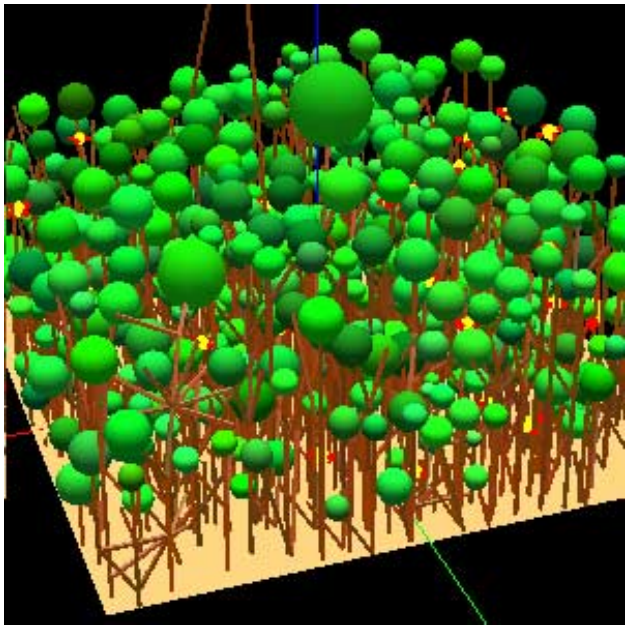

図 4.24: 65913 ターン：フィールドの様子

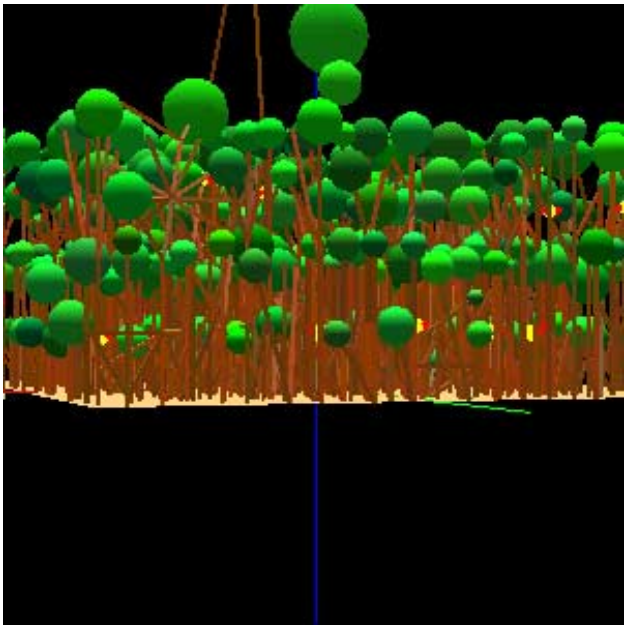

図 4.25: 65913 ターン：フィールドの様子 (真横)

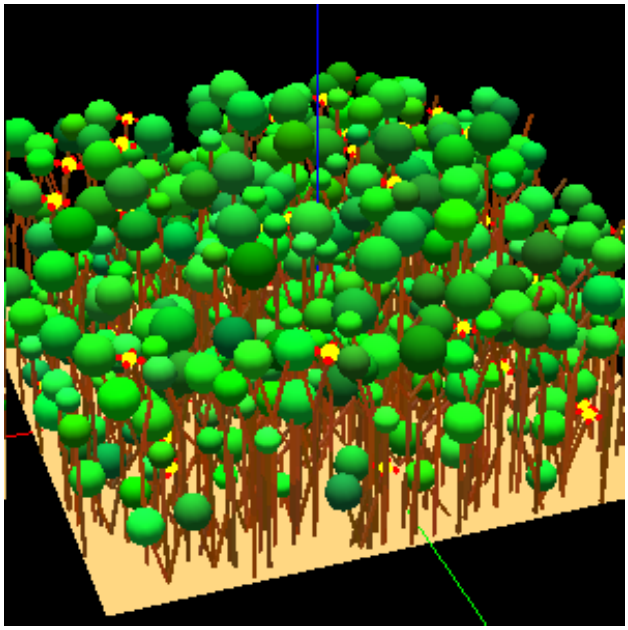

図 4.26: 165624 ターン：フィールドの様子

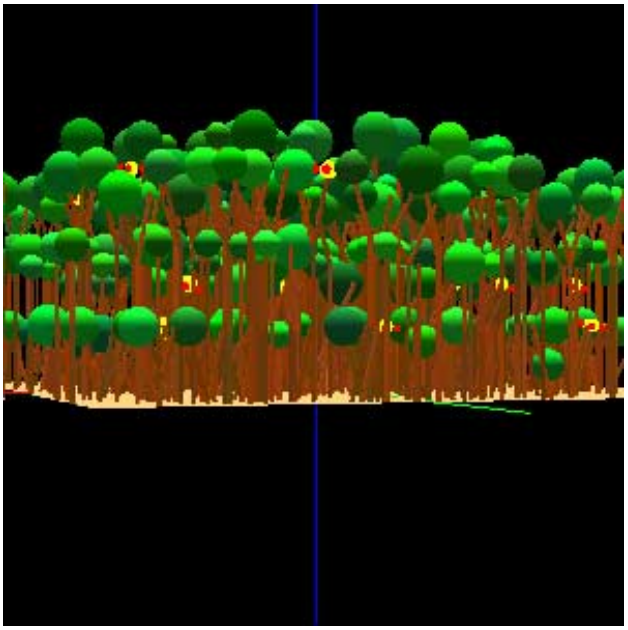

図 4.27: 165624 ターン：フィールドの様子 (真横)

4.1.2　10000 ターンまでのログファイルから見る推移

ここから先では、10000 ターンに達するまでのログファイルの推移を見ていくことにする。

まず、個体数とジーンの種類の数の推移を示したのが、それぞれ図 4.28, 4.29 である。基本的に個体数は、

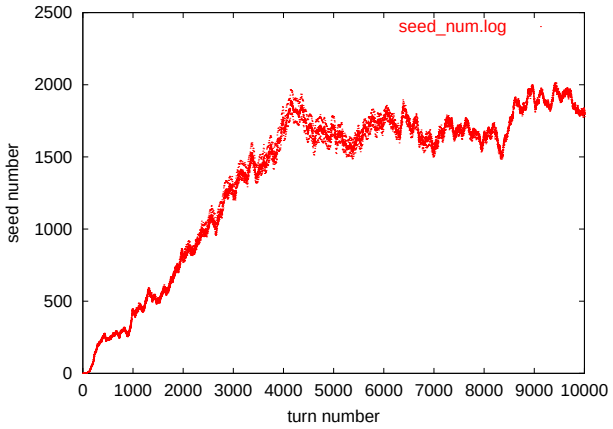

図 4.28: seed_num.log(個体数)

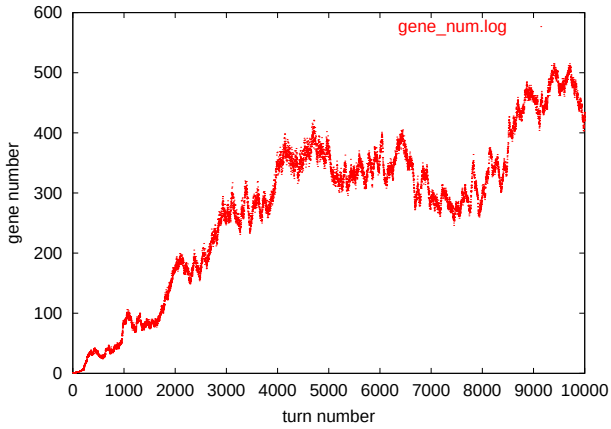

図 4.29: gene_num.log(ジーンの種類)

フィールドキャパシティー (fie_cap.log, 図 4.2) と似たような傾向を示すことが多い。このことは、フィールド上でどれだけの日光を受けられるかということが、個体数と直接関係を持っているということを示している。ジーンの種類も、比較的似ているが多少異なる部分がある。それは、個体数の増減よりもジーンの種類の数の増減の方が早く起こるということである。つまり新しい種が生まれてからフィールド上に広がるまでにはある程度の時間が掛かるということである。

つぎに、平均評価値、ジーンの長さの平均、平均生存ターン数 (平均寿命) を示したのが、それぞれ図 4.30, 4.31, 4.32 である。 まず、平均評価値は基本的に breed_flower に成功した平均回数に 1 加えたものであって、こ

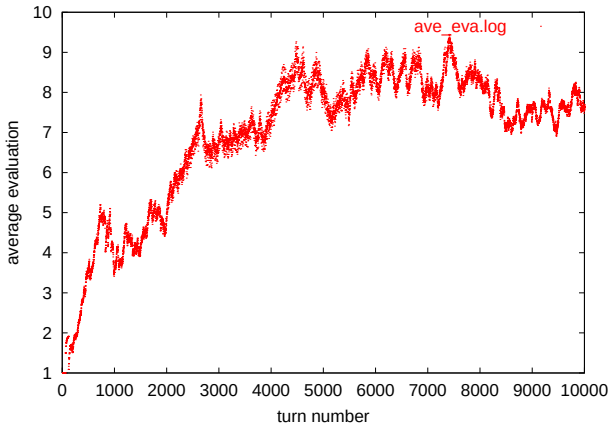

図 4.30: ave_eva.log(平均評価値)

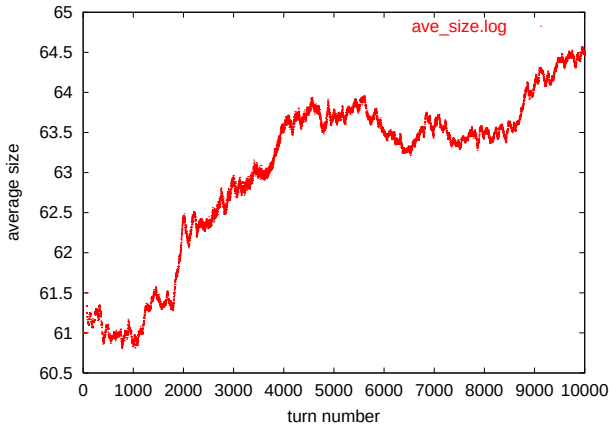

図 4.31: ave_size.log(ジーンの長さの平均)

れは個体数の増減との関係が強い。個体数が増えているにも関わらず平均評価値が下がるという部分があるのは、新しく誕生した個体が breed_flower を実行するまでにはある程度時間がかかるからである。

ジーンの長さの平均のグラフは少し今までとは違った様相を示している。このグラフは短期的に増加する部分となだらかに減少する部分を繰り返しているのである。短期的に増加する部分は、フィールドの様子が大きく変

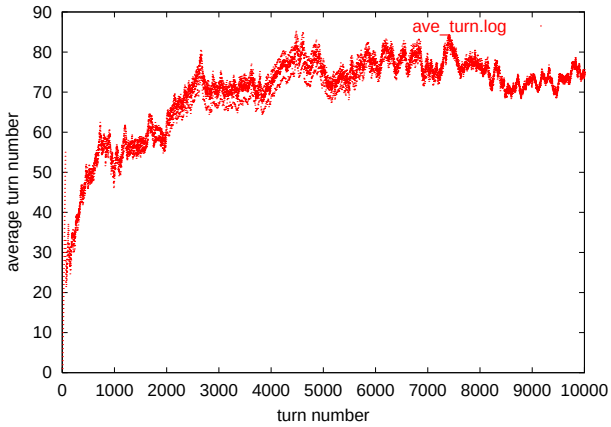

図 4.32: ave_turn.log(平均生存ターン数、平均寿命)

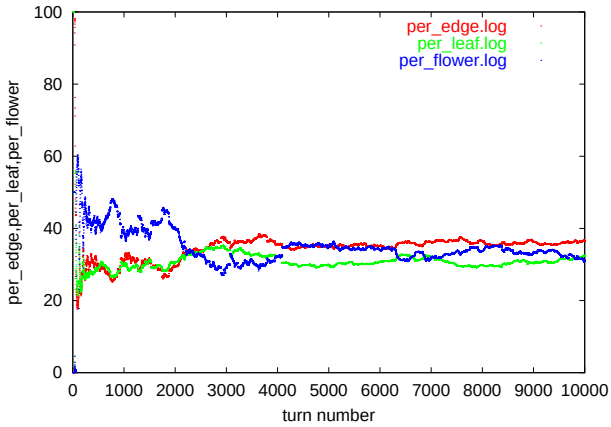

図 4.33: per_edge.log,per_leaf,per_flower(枝、葉、花に使用したエネルギーの割合)

化する時期と重なっており、ジーンが長くたくさんの命令を持ち複雑な表現をできるようになった種がフィールドを変えたということである。逆になだらかに減少する部分では、同じような表現型を持ちながらもジーンを短くして breed_flower までの時間を短くすることで、複製スピードを上昇させる進化が進むということである。このことから表現型が同じでありながら遺伝型を変えていくという、中立的な進化を見ることが出来る。このように、短期的な急激な変化と長期的ななだらかな変化を持っているという性質は進化論では重要な意味を持っている。

平均生存ターン数は、急激な上昇の後にあまり変化は見られない。平均生存ターン数が一定であるということは、古い個体と新しい個体の入れ替えが常に行われているということを意味していると考えられる。

枝、葉、花のそれぞれにどれだけのエネルギーを使用したかの割合を示すのが、図 4.33 である。デフォルトパラメータは、この 3 つのパーツにそれぞれ同じような割合でエネルギーを使用するように設定してある。これは、 3 つのパーツに使用するエネルギーの推移を比較しやすくするためである。

最初の頃は breed_flower の影響が大きいので花へのエネルギーが多いが、次第に枝や、葉に移ってくる。このことは花をつけることよりも、空間的な位置の方が重要であることを示している。さらに進むと、今度は葉や花に多くなる。これは複雑な種が複数の葉や花を持つようになったことが原因であると考えることが出来る。

4.2　分散分析による複数回のシミュレーションの解析

ここでは、パラメータを変えてのシミュレーションを複数回行うことによって得られる結果を、分散分析することで、パラメータとそれから得られる結果との関係を解析したい。分散分析については、[Taguchi1979] を参考にしてほしい。

まず、変化させるパラメータとして概念的に表 4.3 のようなものを考える。これに対応する具体的なパラメータとして表 4.4 のものを考える。ここから先のシミュレーションでは、表 4.4 にあるものだけを変化させ、他のパラメータはデフォルト値に固定する。また、変化させるパラメータもここから先では、$day, edge, leaf, flower$ という省略名を使用することにする。

次に、表 4.4 にあるパラメータにどのような水準を設定していくかを考えることにする。まず、先祖種では 36 ターン終了から光合成が始まり、60 ターン終了で 2 回の breed_flower を行うので、この 2 ヶ所のターンで貯え

表 4.3: 変化させるパラメータ (概念的なもの)

内的要因	枝、葉、花のそれぞれの 1 ターンの維持エネルギー
外的要因	単位面積あたりの日光の量

表 4.4: 変化させるパラメータ (具体的なもの)

パラメータ	省略名		デフォルト値
mField_daylight	*day*	フィールドの単位格子あたりの日光の量	0.1286
mEdge_keep_calorie	*edge*	枝の単位カロリーあたりのターンごとの維持カロリー	2.0
mLeaf_keep_calorie	*leaf*	葉の単位カロリーあたりのターンごとの維持カロリー	1.0
mFlower_keep_calorie	*flower*	花の単位カロリーあたりのターンごとの維持カロリー	0.01

ているエネルギーが 0 にならないようにする。このことを、維持エネルギーなどを考えそれぞれのターンで計算し、4 つのパラメータの制約式で表すと表 4.5 にようになる。ここで、日光の計算についてはフィールドを格子

表 4.5: パラメータによる制約式

ターン数	制約式
36	$102.6 + 62.83 \times edge + 16.00 \times leaf \quad \leq 1000$
60	$2237 + 447.0 \times edge + 1088 \times leaf$ $+3072 \times flower - (3.878 \times 10^4) \times day \quad \leq 1000$

状に分割して厳密に計算したものではなく、円の面積で近似計算したものである。また、右辺の 1000 という値は最初の個体に設定される初期エネルギーである。この 2 つの制約式のうち *day* の値をかなり大きく取らない限り 60 ターンの制約式のほうが基本的に強い制約になるので、60 ターンの制約式のみに注目することにする。

4 つのパラメータのうち、最初に単位面積あたりの日光の量の *day* についての水準を決定する。*day* についてはデフォルト値 0.1286 の近くで 4 水準取ることにして、

$$day = 0.10, 0.12, 0.14, 0.16$$

とする。

ここで、もっと小さい水準の $day = 0.10$ とデフォルト値 $edge = 2.0, leaf = 1.0$ に対して 60 ターンの制約式を満たすように $flower$ を取るようにすると、$flower \leq 0.215$ である。これに近い値とデフォルト値の $flower = 0.01$ の間で 4 水準取ると

$$flower = 0.01, 0.08, 0.15, 0.22$$

となる。

残る $edge, leaf$ については、デフォルト値と 0 との間を等分するように 4 水準取る。

ここまでをまとめると、それぞれのパラメータの取る 4 つの水準は、表 4.6 のようになる。

ここで定めたパラメータの 4 水準を、16 回行うシミュレーションのそれぞれに対して表 4.7 のようにしてわりつける。(シミュレーション No.1 で *day* が 1 というのは、No.1 のシミュレーションではパラメータ *day*

表 4.6: パラメータの水準

	day	*edge*	*leaf*	*flower*
第 1 水準	0.10	0.50	0.25	0.01
第 2 水準	0.12	1.0	0.50	0.08
第 3 水準	0.14	1.5	0.75	0.15
第 4 水準	0.16	2.0	1.0	0.22

が第 1 水準、つまり $day = 0.10$ ということである。) この表 4.7 の特徴は、 16 回のシミュレーションを行う

表 4.7: パラメータのわりつけ

	day	*edge*	*leaf*	*flower*
No.01	1	1	1	1
No.02	1	2	2	2
No.03	1	3	3	3
No.04	1	4	4	4
No.05	2	1	2	3
No.06	2	2	1	4
No.07	2	3	4	1
No.08	2	4	3	2
No.09	3	1	3	4
No.10	3	2	4	3
No.11	3	3	1	2
No.12	3	4	2	1
No.13	4	1	4	2
No.14	4	2	3	1
No.15	4	3	2	4
No.16	4	4	1	3

と、それぞれのパラメータでの 4 水準が 4 回ずつ現れていることと、どの 2 つのパラメータを組合せ (例えば $(day, edge)$,$(day, flower)$ などを組合せ) ても、

$$(1,1),(1,2),(1,3),(1,4),(2,1),(2,2),\ldots,(4,3),(4,4)$$

がそれぞれ 1 度ずつシミュレーションのなかに現れるということである。表 4.7 のような特徴を持っている表のことを、実験計画法では直交表という。

ここから先では、表 4.7 によってシミュレーションのパラメータを設定し、16 回シミュレーションを行い、10000 ターン経過したところで表 4.8 にあげた結果を数値として得る。なお、10000 ターンのところで結果を得るのは、前節のデフォルトパラメータによるシミュレーションで見たときに、このあたりでフィールドが限界に達していることと、ある程度の種が存在して、フィールド全体の安定期間が長くなることからである。16 回のシミュレーションで実際に得られるのは、表 4.11 に示してあるような値である。(表 4.11 には次節の参考のために、gene_num.log と ave_turn.log の 16 回のシミュレーションで得られた観測値を載せてある。)

表 4.8: 結果として得る項目

seed_num.log	シードの数 (個体数)
gene_num.log	ジーンの数 (遺伝子の種類)
ave_eva.log	平均評価値
ave_size.log	ジーンの長さの平均値
ave_turn.log	平均生存ターン数 (寿命)
per_edge.log	枝に利用したエネルギーの割合
per_leaf.log	葉に利用したエネルギーの割合
per_flower.log	花に利用したエネルギーの割合

次に、16 回のシミュレーションの結果の数値から分散分析し、まず 4 つのパラメータの 3 次元の関数の和で表す。例えば gene_num.log で得られる値 (ジーンの数、遺伝子の種類) を、

$$\begin{aligned}
\text{gene_num.log} \quad = \quad & 743.4 \\
& + \quad 7718 \times (day - 0.13) \\
& - \quad (1.468 \times 10^4) \times [(day - 0.13)^2 - 0.0005] \\
& + \quad (3.531 \times 10^6) \times [(day - 0.13)^3 - 0.00082 \times (day - 0.13)] \\
& - \quad 415.5 \times (edge - 1.25) \\
& + \quad 387.0 \times [(edge - 1.25)^2 - 0.3125] \\
& - \quad 480.0 \times [(edge - 1.25)^3 - 0.5125 \times (edge - 1.25)] \\
& - \quad 59.80 \times (leaf - 0.625) \\
& - \quad 376.0 \times [(leaf - 0.625)^2 - 0.07813] \\
& - \quad 522.7 \times [(leaf - 0.625)^3 - 0.1281 \times (leaf - 0.625)] \\
& - \quad 949.3 \times (flower - 0.115) \\
& - \quad 9234 \times [(flower - 0.115)^2 - 0.006125] \\
& + \quad (2.143 \times 10^5) \times [(flower - 0.115)^3 - 0.01005 \times (flower - 0.115)]
\end{aligned}$$

のように表す。そして、この関数のなかから効果の薄い項を消し、また誤差などから信頼限界を計算すると、

$$\begin{aligned}
\text{gene_num.log} \quad = \quad & 743.4 \\
& + \quad 7718 \times (day - 0.13) \\
& - \quad 415.5 \times (edge - 1.25) \\
& + \quad 387.0 \times [(edge - 1.25)^2 - 0.3125] \\
& \pm \quad 428.3 \quad (96.54\%)
\end{aligned}$$

のような式を得ることができる。ここで括弧の中の数字はこの関数で 16 回のシミュレーションの結果の数値をどれだけ表現できているかを示す値である。また $\pm$ の部分で $-$ を取ったときの値と $+$ を取ったときの値がそれぞれ信頼限界の下限値と上限値である。信頼限界は 5% 程度の有意水準を持っているので、95% 程度の確率でこの区間であると推定することができる。ただし、この信頼限界の計算はもとのデータに依存しているので正確に 95% というわけではない。

このように、3 次元関数の和と信頼限界で表しているのは、実験計画法のモデルによる部分が大きい。実験計画法のモデルでは、そもそもパラメータの多項式でデータを表せるという仮定や前提条件を置くことはしない。得られた多項式でそのもとになったデータを何パーセント表現できるかということを示しているに過ぎないのである。

ここから、表 4.8 にあげた結果を 4 つのパラメータの 3 次元関数の和と信頼限界で表すことにする。

seed_num.log :: シードの数 (個体数)

$$\begin{aligned}\text{seed\_num.log} \quad = \quad & 2193 \\ & + \quad (1.596 \times 10^4) \times (day - 0.13) \\ & - \quad 1259 \times (edge - 1.25) \\ & + \quad 1554 \times [(edge - 1.25)^2 - 0.3125] \\ & \pm \quad 986.7 \quad (97.85\%)\end{aligned}$$

gene_num.log ::ジーンの数 (遺伝子の種類)

$$\begin{aligned}\text{gene\_num.log} \quad = \quad & 743.4 \\ & + \quad 7718 \times (day - 0.13) \\ & - \quad 415.5 \times (edge - 1.25) \\ & + \quad 387.0 \times [(edge - 1.25)^2 - 0.3125] \\ & \pm \quad 428.3 \quad (96.54\%)\end{aligned}$$

ave_eva.log :: 平均評価値

$$\begin{aligned}\text{ave\_eva.log} \quad = \quad & 8.368 \\ & + \quad 17.34 \times (day - 0.13) \\ & + \quad 1.558 \times [(edge - 1.25)^2 - 0.3125] \\ & + \quad 1.820 \times (leaf - 0.625) \\ & \pm \quad 1.824 \quad (99.41\%)\end{aligned}$$

ave_size.log :: ジーンの長さの平均値

$$\text{ave\_size.log} \quad = \quad 68.26$$

$$
\begin{aligned}
&+ \quad (3.129 \times 10^5) \times [(day - 0.13)^3 - 0.00082 \times (day - 0.13)] \\
&- \quad 271.0 \times [(flower - 0.115)^2 - 0.006125] \\
&\pm \quad 6.430 \quad (99.89\%)
\end{aligned}
$$

ave_turn.log :: 平均生存ターン数 (平均寿命)

$$
\begin{aligned}
\text{ave\_turn.log} \quad = \quad & 84.18 \\
&+ \quad 109.0 \times (day - 0.13) \\
&- \quad 4.349 \times (edge - 1.25) \\
&- \quad 21.26 \times [(edge - 1.25)^3 - 0.5125 \times (edge - 1.25)] \\
&- \quad 178.9 \times [(leaf - 0.625)^3 - 0.1281 \times (leaf - 0.625)] \\
&\pm \quad 8.946 \quad (99.86\%)
\end{aligned}
$$

per_edge.log :: 枝に利用したエネルギーの割合

$$
\begin{aligned}
\text{per\_edge.log} \quad = \quad & 33.96 \\
&+ \quad 7.821 \times (edge - 1.25) \\
&- \quad 43.46 \times (flower - 0.115) \\
&\pm \quad 15.40 \quad (97.55\%)
\end{aligned}
$$

per_leaf.log :: 葉に利用したエネルギーの割合

$$
\begin{aligned}
\text{per\_leaf.log} \quad = \quad & 26.41 \\
&- \quad 227.0 \times (day - 0.13) \\
&+ \quad 26.47 \times (leaf - 0.625) \\
&\pm \quad 21.62 \quad (92.92\%)
\end{aligned}
$$

per_flower.log :: 花に利用したエネルギーの割合

$$
\begin{aligned}
\text{per\_flower.log} \quad = \quad & 39.63 \\
&+ \quad 115.5 \times (day - 0.13)
\end{aligned}
$$

$$
\begin{aligned}
&- \quad 8.212 \times (edge - 1.25) \\
&+ \quad 12.86 \times [(edge - 1.25)^2 - 0.3125] \\
&- \quad 21.33 \times (leaf - 0.625) \\
&\pm \quad 10.69 \quad (99.14\%)
\end{aligned}
$$

ここまでの分散分析で得られた結果にデフォルトパラメータ (表 4.9) を代入して、信頼限界などから存在範囲を計算し、前節のシミュレーションのときの結果と比較したのが、表 4.10 である。

表 4.9: デフォルトパラメータ

パラメータ	デフォルト値
day	0.1286
$edge$	2.0
$leaf$	1.0
$flower$	0.01

表 4.10: デフォルトパラメータのときのシミュレーション結果と分散分析から得た存在範囲の比較

	デフォルト値の結果	存在範囲の下限値	存在範囲の中央値	存在範囲の上限値
seed_num.log	1779	627.4	1614	2601
gene_num.log	422	89.13	517.5	945.8
ave_eva.log	7.696	7.584	9.408	11.23
ave_size.log	64.48	60.86	67.29	73.72
ave_turn.log	75.59	70.18	79.12	88.07
per_edge.log	36.50	28.99	44.39	59.79
per_leaf.log	32.30	15.04	36.66	58.28
per_flower.log	31.19	17.83	28.52	39.20

なお、デフォルトパラメータのときの存在範囲は、gene_num.log を例にすると次のようにして計算している。

$$
\begin{aligned}
\text{gene\_num.log} \quad &= \quad 743.4 \\
&+ \quad 7718 \times (day - 0.13) \\
&- \quad 415.5 \times (edge - 1.25) \\
&+ \quad 387.0 \times [(edge - 1.25)^2 - 0.3125] \\
&\pm \quad 428.3 \\
&= \quad 743.4 \\
&+ \quad 7718 \times (0.1286 - 0.13) \\
&- \quad 415.5 \times (2.0 - 1.25) \\
&+ \quad 387.0 \times [(2.0 - 1.25)^2 - 0.3125] \\
&\pm \quad 428.3
\end{aligned}
$$

$$
\begin{aligned}
&= 517.5 \pm 428.3 \\
&= \begin{cases} 89.13 & \text{存在範囲の下限値} \\ 517.5 & \text{存在範囲の中央値} \\ 945.8 & \text{存在範囲の上限値} \end{cases}
\end{aligned}
$$

表 4.10 から解るようにデフォルトパラメータのときのシミュレーション結果は、分散分析によって求めた存在範囲に入っており、分散分析から得ることができた 4 つのパラメータの 3 次元関数の和と信頼限界でシミュレーション結果を表せるということの確認をすることができた。

次節では、さらに踏み込んで、これから行おうとしているシミュレーション結果を先に推測しておき、実際にシミュレーションをして、そこから得られた結果と推測値を比較することにする。

また、次節への参考として分散分析のもととなったシミュレーションの観測値のうち gene_num.log と ave_turn.log について表 4.11 にまとめておく。なお、表 4.11 の中のシミュレーションの No は表 4.7 に一致して

表 4.11: 16 回のシミュレーションで得られた gene_num.log,ave_turn.log の観測値

	gene_num.log	ave_turn.log
No.01	916	84.67
No.02	683	77.78
No.03	234	87.46
No.04	157	78.93
No.05	1157	85.35
No.06	443	76.18
No.07	599	79.12
No.08	591	82.99
No.09	1192	90.69
No.10	610	85.33
No.11	864	88.26
No.12	538	78.59
No.13	1414	94.21
No.14	1050	90.24
No.15	690	85.11
No.16	756	81.93

いる。この表からも、分散分析で得られた 3 次元関数の和と信頼限界によってシミュレーションの観測値を表現できているということを先程の gene_num.log の存在範囲の計算と同じ過程の計算によって確認することができる。

4.3　進化の方向を設定した確認シミュレーション

この節では、本論文の主題ともいうことができる進化の方向の推測について扱う。

前節のシミュレーションによる結果から、パラメータとそれに依存する進化の関係がわかるようになった。そ

こで、ここではパラメータとそれに依存する進化の方向の関係を利用して、設定された進化の方向に進化が起こるかどうかを確認シミュレーションを行うことで観測することにする。

ここで設定する進化の方向として、次の 2 つを考えることにする。

1. gene_num.log(ジーンの数, 遺伝子の種類) の最大化
2. ave_turn.log(平均生存ターン数, 平均寿命) の最大化

このうち前者はフィールド上にどれだけの多様な種が存在しているかということを最大にしようというものであり、後者はそれぞれの個体がどれだけ長く生存できるのかということを最大にしようというものである。特に注目したいのは前者の多様性の最大化であり、より多様な環境ほど外界からの変化に強くなると考えることができる。

まず、前節の結果から gene_num.log は 4 つのパラメータの関数と信頼限界で次のように表すことができる。

$$\begin{aligned}\text{gene\_num.log} \quad = \quad & 743.4 \\ & + \quad 7718 \times (day - 0.13) \\ & - \quad 415.5 \times (edge - 1.25) \\ & + \quad 387.0 \times [(edge - 1.25)^2 - 0.3125] \\ & \pm \quad 428.3 \quad (96.54\%)\end{aligned}$$

ここで、 4 つのパラメータを下限値を第 1 水準、上限値を第 4 水準とする実数区間で考えると、この関数を最大にするパラメータは、表 4.12 のように決定される。表 4.12 のなかで’——’ となっているパラメータは、

表 4.12: gene_num.log を最大化するパラメータ

	最大化する値	下限値	上限値
day	0.16	0.10	0.16
$edge$	0.5	0.5	2.0
$leaf$	——	0.25	1.0
$flower$	——	0.01	0.22

gene_num.log の値に余り影響しないということである。

同様に、前節の結果から ave_turn.log は 4 つのパラメータの関数と信頼限界で次のように表すことができる。

$$\begin{aligned}\text{ave\_turn.log} \quad = \quad & 84.18 \\ & + \quad 109.0 \times (day - 0.13) \\ & - \quad 4.349 \times (edge - 1.25) \\ & - \quad 21.26 \times [(edge - 1.25)^3 - 0.5125 \times (edge - 1.25)] \\ & - \quad 178.95 \times [(leaf - 0.625)^3 - 0.1281 \times (leaf - 0.625)] \\ & \pm \quad 8.946 \quad (99.86\%)\end{aligned}$$

表 4.13: ave_turn.log を最大化するパラメータ

	最大化する値	下限値	上限値
day	0.16	0.10	0.16
edge	0.5	0.5	2.0
leaf	0.8317	0.25	1.0
flower	——	0.01	0.22

ここで、4 つのパラメータを下限値を第 1 水準、上限値を第 4 水準とする実数区間で考えると、この関数を最大にするパラメータは、表 4.13 のように決定される。表 4.13 のなかで’——’ となっているパラメータは、ave_turn.log の値に余り影響しないということである。

表 4.12 と表 4.13 のパラメータを統合して、表 4.14 をパラメータとして設定する。*flower* に関しては分散分析で得られた式にあまり影響をおよぼさないということなので、デフォルト値で設定することにする。

表 4.14: gene_num.log,ave_turn.log を最大化する目的で設定するパラメータ

	設定するパラメータ	下限値	上限値
day	0.16	0.10	0.16
edge	0.5	0.5	2.0
leaf	0.8317	0.25	1.0
flower	0.01(デフォルト値)	0.01	0.22

表 4.14 のパラメータから、表 4.15 に示したように gene_num.log と ave_turn.log の最大値の存在範囲を推測することができる。

表 4.15: 推測される gene_num.log,ave_turn.log の最大値の存在範囲

	gene_num.log	ave_turn.log
存在範囲の下限値	954.9	85.72
存在範囲の中央値	1383	94.67
存在範囲の上限値	1812	103.6

ここで、表 4.15 で示された最大値の存在範囲の推測が正しいのかどうかを調べるため、表 4.14 のパラメータを設定し、10 回の確認シミュレーションを行い、10000 ターンのところでの gene_num.log と ave_turn.log の観測値をまとめたのが、表 4.16 である。

表 4.16 の結果に対しても分散分析する。ただし、パラメータがすべて同じであるので今回の分散分析では平均値と信頼限界で表すことになる。そのため、括弧の中の値は平均値で 10 回の確認シミュレーションの観測値の何パーセントを表現できるかを示している。

$$\begin{aligned} \text{gene\_num.log} &= 1222 \quad \pm 106.1 \quad (98.69\%) \\ \text{ave\_turn.log} &= 90.25 \quad \pm 3.378 \quad (99.75\%) \end{aligned}$$

表 4.16: gene_num.log,ave_turn.log を最大化するパラメータを設定した確認シミュレーションから得られた gene_num.log,ave_turn.log の観測値

	gene_num.log	ave_turn.log
No.01	1447	87.04
No.02	964	85.10
No.03	1175	92.77
No.04	1146	90.51
No.05	1161	90.20
No.06	1398	92.27
No.07	1334	88.02
No.08	1174	89.22
No.09	1248	100.9
No.10	1171	86.48

10 回の確認シミュレーションの結果と推測値を比較しやすいように表にまとめたのが、表 4.17 である。表 4.17

表 4.17: gene_num.log,ave_turn.log の推測値と確認シミュレーションから得られた観測値の存在範囲による比較

		下限値	中央値	上限値
gene_num.log	(推測値)	954.9	1383	1812
gene_num.log	(確認シミュレーションの観測値)	1116	1222	1328
ave_turn.log	(推測値)	85.72	94.67	103.6
ave_turn.log	(確認シミュレーションの観測値)	86.87	90.25	93.63

を見ると解るように、確認シミュレーションから得られた値の存在範囲は、推測値の存在範囲に完全に覆われる形になっている。このことから、分散分析で得られた式によって推測した方向に進化が起こったことが解る。

また、前節で様々にパラメータを変化させたときの観測値である表 4.11 と比較をしてみると、表 4.17 で得られた値は、その最大値を達成していると見ることができる。表 4.11 の中には、表 4.17 の確認シミュレーションの存在範囲の中央値を超えるものも存在する (表 4.11 の No.9 と No.13) が、これらで用いられているパラメーターが本節で設定されたパラメータに比較的近いためであると考えられる。したがって、ここで設定した確認シミュレーションのパラメータ (表 4.14) が、最大値を達成することのできるパラメータであるということができる。

これらの結果から、パラメータと進化の方向の関係を解析しその関係を利用することで、種の多様性、あるいは平均寿命の最大化という環境を、パラメータの設定で意図的につくりだすことが可能であり、さらにパラメータに依存する進化の方向を推測することも可能であるということを示すことができた。

この節の最後に、ここまでで得られた結果から、種を多様にするパラメータがどのようなことを意味するのかということについて、考えてみることにする。種を多様にするパラメータを再掲すると表 4.18 のようになる。表 4.18 を見ると解るように、外的要因である日光 (*day*) は多い方が、そして内的要因である枝の維持エネルギー (*edge*) は少ない方が、種が多様になる。

このことをより具体的に解釈すると、種を多様にするという目的で進化を行わせるのであれば、その環境はで

表 4.18: gene_num.log を最大化するパラメータ

	最大化する値	下限値	上限値
day	0.16	0.10	0.16
edge	0.5	0.5	2.0
leaf	——	0.25	1.0
flower	——	0.01	0.22

きるだけ日光の多い環境を選び、そして最初に置く個体 (先祖種) としてはできるだけ枝に対しての維持エネルギーを使わない背の低い個体を選ぶべきである、ということである。

4.4　突然変異の及ぼす影響

この節では、本論文でのシミュレーションの参考として、突然変異のパラメータが変化させたときにどのような影響が現れるのかということについて述べておく。

本論文のシミュレーションで、突然変異が起こるのは、自己複製のとき、つまり breed_flower 命令で新しいたねを作るときのみである (参照 3.8 節)。突然変異において breed_flower を実行した個体のジーンから子供のジーンを作るときに、パラメータ *mGene_vectorVariation*, *mGene_sizeVariation*, *mGene_operatorVariation* の影響を受け、それぞれのパラメータが大きいほど、突然変異の確率が上がり、親子で異なったジーンを持つ傾向が強くなる。ここでは、 3 つのパラメータを変化させたときにどのような影響が現れるのかを見ることにする。

まず、表 3.14 の突然変異に関してのパラメータをもう一度表 4.19 として再掲する。

表 4.19: 突然変異のパラメータとそのデフォルト値

パラメータ名	デフォルト値	簡易説明
mGene_vectorVariation	0.02	gene_separate 単位の命令の挿入削除が起こる確率
mGene_sizeVariation	0.02	1 つの命令の挿入削除が起こる確率
mGene_operatorVariation	0.05	1 つの命令の突然変異が起こる確率

ここで、パラメータ *mutation* を新たに置き、3 つのパラメータを次式で表すことにする。

$$\begin{cases} mGene\_vectorVariation &= &mutation \\ mGene\_sizeVariation &= &mutation \\ mGene\_operatorVariation &= &2.5 \times mutation \end{cases}$$

つまり、それぞれの 3 つのパラメータを一つのパラメータ *mutation* で調整するようにする。したがって、*mutation* を大きく設定すれば突然変異が起きやすくなり、逆に *mutation* を小さくすれば突然変異は起こりにくくなる。

ここから先では、概念的には *mutation* のパラメータのみを変化させ、3 つのパラメータは上の式によって求まる値を用いることにする。また、前節までは $day, edge, leaf, flower$ のパラメータを変化させてきたが、この節ではこれらのパラメータはデフォルト値に固定することにする。

変化させる *mutation* の値は、表 4.19 と上の式からのデフォルト値 0.02 を中心として、

$$\begin{aligned} mutation \quad = \quad & 0.001, 0.002, 0.004, \\ & 0.01,\ 0.02,\ 0.04, \\ & 0.1,\quad 0.2,\quad 0.4 \end{aligned}$$

のように変化させることにする。ここで、それぞれの *mutation* の値に応じてシミュレーションを行い、10000 ターンのところで情報を集めることにする。なお、$mutation = 0.02$ のところのシミュレーションは 4.1 節と同じパラメータになるが、この節とはシミュレーションに用いる乱数の初期値が異なるために結果にも多少の違いがあるが、その 2 つのシミュレーションは同一視してもかまわない程度の違いである。

10000 ターンのところでのジーンの種類 (種の数) とフィールド上の個体数の比率を計算して、*mutation* を対数で横軸に、比率を縦軸にしたものが、図 4.34 である。 なお、この節では、10000 ターン時点でのジーンの種類 (種の数) とフィールド上の個体数をそれぞれ *gene*, *seed* と略記することにする。

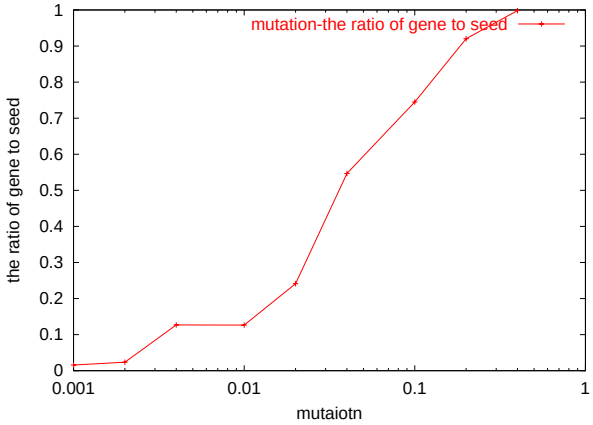

図 4.34: *gene*/*seed* と *mutation* のグラフ

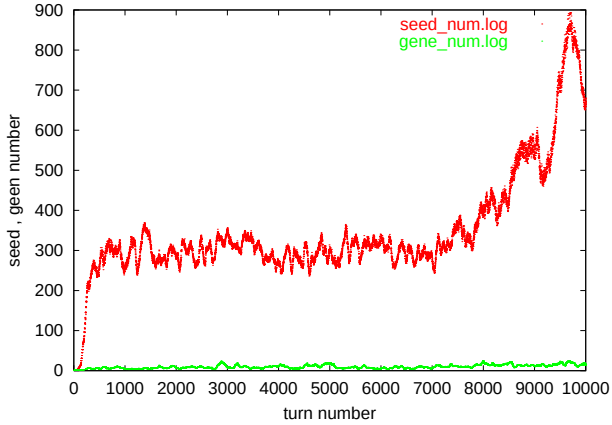

図 4.35: *mutation* = 0.002 のときの *gene* と *seed* の時間推移

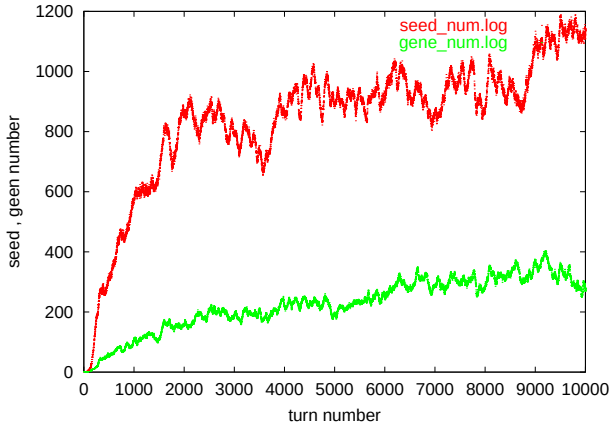

図 4.36: *mutation* = 0.02 のときの *gene* と *seed* の時間推移

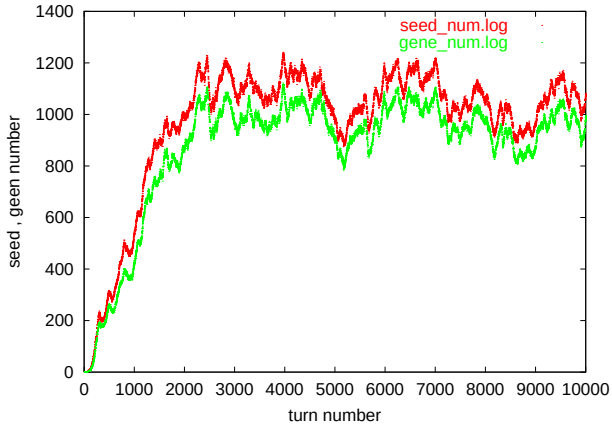

図 4.37: *mutation* = 0.2 のときの *gene* と *seed* の時間推移

図 4.34 を見ると解るように、*mutation* が大きいと *gene* と *seed* がほとんど等しくなり、*mutation* が小さいと *gene* は小さい数のままとどまる傾向にある。要するに、*mutation* が大きくなりすぎると突然変異でジーンがさまざまに変わってしまうので、それぞれの個体で異なったジーンを持つようになり、逆に *mutation* が小さすぎるとほとんど突然変異が起こらないため、先祖種のジーンに近い形がそのまま残されるということである。このことを更に裏付けるのが、図 4.35, 4.36, 4.37 である。この 3 つの図は、それぞれ *mutation* = 0.002, 0.02, 0.2 のときの *gene* と *seed* の時間推移を表したものである。この図からも、*mutation* が 1 に近くなると *gene* と *seed* がほぼ等しい状態で時間推移し、*mutation* が 0 に近くなると *gene* は *seed* の増減とはほとんど無関係に小さい数でとどまることになる。

ここまでで得られた関係を、次は別の角度から見ることにする。図 4.38 は、10000 ターンの時点でもっとも個体数の多いジーンの個体数 (*max gene*) が全体の個体数 (*seed*) に占める比率を、*mutation* を対数で横軸に、比率を縦軸にしたものである。 図 4.38 でもわかるように、*mutation* が小さいと一つのジーンに一極集中の形になり、*mutation* が大きいとさまざまなジーンに分散する形になる。このことは、図 4.39, 4.40, 4.41 からも見て取ることができる。図 4.39, 4.40, 4.41 では、*mutaion* = 0.002, 0.02, 0.2 においての 10000 ターン時点でのジーン (仮想マシン語シークエンス) の長さについての個体数の分布を示している。つまり、個体の中の仮想マシン語シー

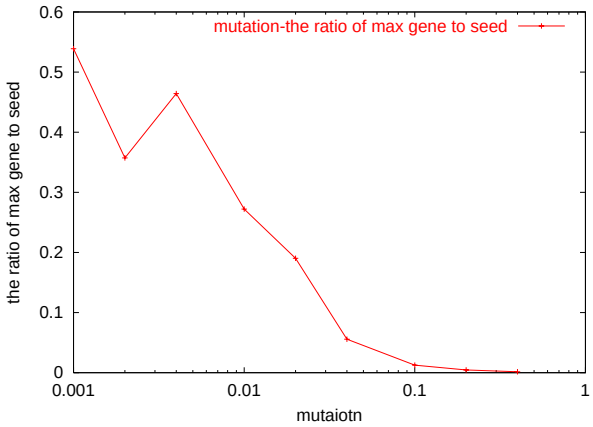

図 4.38: $(max\ gene)/seed$ と $mutation$ のグラフ

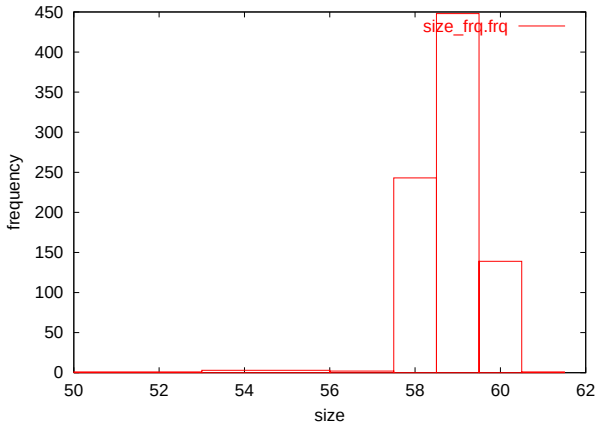

図 4.39: $mutation = 0.002$ のときのジーンの長さについての個体数の分布

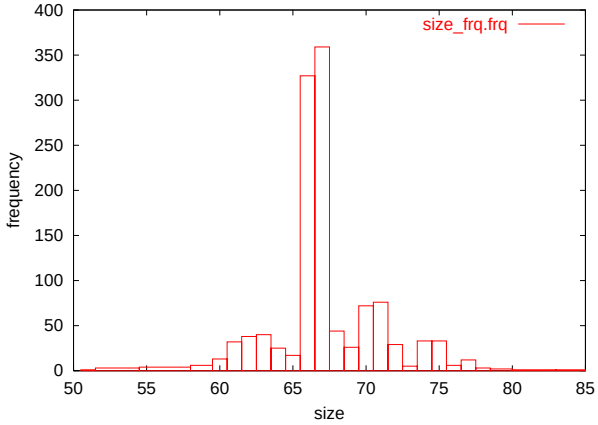

図 4.40: $mutation = 0.02$ のときのジーンの長さについての個体数の分布

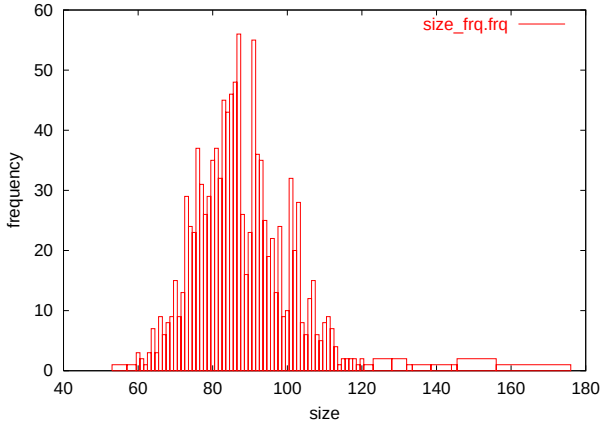

図 4.41: $mutation = 0.2$ のときのジーンの長さについての個体数の分布

クエンスがどれだけの長さで構成されているかで分類したものである。図 4.39, 4.40, 4.41 からも、$mutation$ が小さければ一極集中型であり、$mutation$ が大きければ分散型であることがわかる。

次に、10000 ターン時点でのジーンの長さの平均と $mutation$ の関係をグラフにしたのが、図 4.42 である。図 4.42 から $mutation$ が大きくなればジーンも長くなり、$mutation$ が小さくなればジーンも短くなるということがわかる。このことは、図 4.43 からもわかる。図 4.43 は、それぞれ $mutation = 0.002,\ 0.02,\ 0.2$ のときの ジーンの平均の長さの時間推移を表したものである。

$mutation$ が大きくなるとジーンが長くなる傾向にあるのは、突然変異があまりに起こる状態だと必要な機能を作る部分を重複して持っている方が有利であるためと考えられる。花を作るなどの重要な機能の部分が突然変異で失われないように、数ヶ所にコピーを持っているということである。$mutation$ が大きいときには、フィールドへの広がりの速さ、つまり複製の速さよりも、確実に複製ができるようにすることが重要視される。

逆に $mutation$ があまりに小さいときには、ジーンは短くなる傾向にある。これは、ジーン自体には大きな変更がなく、突然変異で誕生する新しいジーンも表現型としては先祖種の複製の確実さを受け継ぐので、必要な機能を重複して持たせるよりも、ジーンを短くすることで複製を速くすることが重要なのである。$mutation$ が小さ

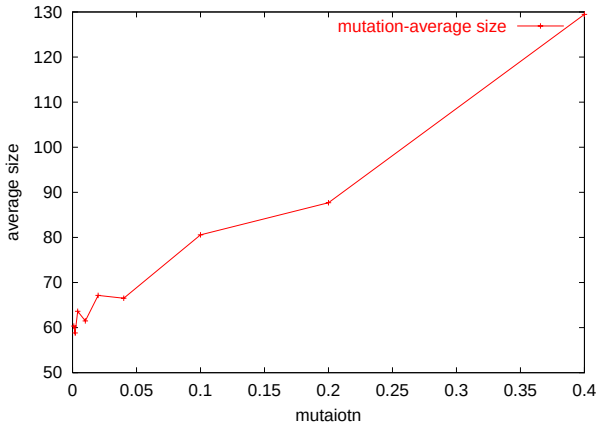

図 4.42: ジーンの長さの平均 と *mutation* のグラフ

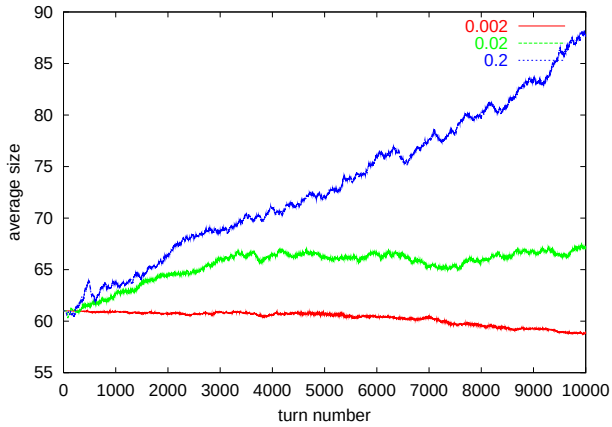

図 4.43: *mutation* = 0.002, 0.02, 0.2 のときのジーンの長さの平均の時間推移

いときは、表現型 (phenotype) がほとんど変わらずに、遺伝子型 (genotype) が変わっていくという中立的な進化を見ることができる。

最後に、*mutation* が変わることでどれだけ進化の速さに影響が出るのか見ることにする。図 4.44 は 10000 ターンまでの *mutation* = 0.002, 0.02, 0.2 のときのフィールドキャパシティの時間推移を表したものである。フィー

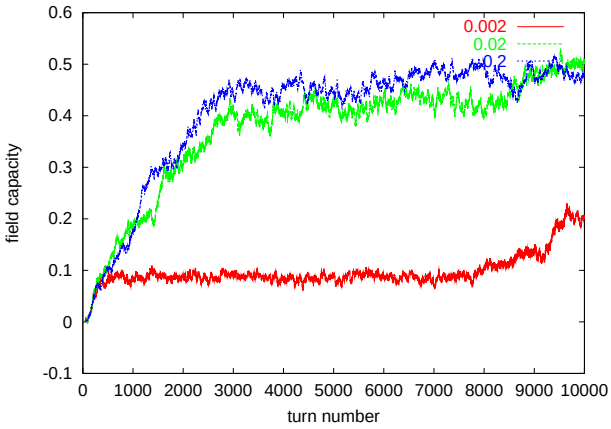

図 4.44: *mutation* = 0.002, 0.02, 0.2 のときのフィールドキャパシティの時間推移

ルドキャパシティは、フィールド全体に降り注ぐ日光の量に対しての植物が受け取る日光の量の比率であるため、正確には進化の速さを表してはいない。しかし、4.1 節で見たようにフィールドキャパシティが大きく変化するところは、新しい種が広がりフィールド全体の様子が変わるところであるので、フィールドキャパシティである程度の進化の速さを見ることができる。

mutation が 0.02 と 0.2 のときを比較すると、フィールドキャパシティの上昇が 0.2 のほうが少しだけ速い。これは、*mutation* が高いほうが新しい種が生まれやすいために、先祖種との空間的な衝突が減少し、フィールド上の空間を有効に活用できるためであると考えられる。また、0.02 と 0.2 の違いが小さいことから、*mutation* はある程度までの大きさになると、それ以上は進化の速さに影響しにくくなることが解る。このことは、必要以上に高い *mutation* は、優れた種に変異することも多いが、それと同時に優れていない種も多くなってしまうことを

意味している。なお、フィールドキャパシティーが 0.5 を超えるあたりから上昇しにくくなるのは、シードの選択が多くなるためであり (参照 3.9 節)、*mutation* の値の影響ではない。

mutation が 0.002 のときには、0.02 や 0.2 と比較してフィールドキャパシティの上昇が明らかに遅い。とくにフィールドキャパシティが 0.08 に近いところで長い時間停滞している。この時期は、4.1 節で見てきたところと比較すると [426 ターン: 先祖種の広がり] の時期に一致する。つまり、*mutation* が小さいと先祖種から大きく変わった種になりにくいので、先祖種が広がったときとほぼ同じようなフィールド上の状態が長時間継続される。4.1 節の [426 ターン: 先祖種の広がりから] の水準 (フィールドキャパシティが 0.08 ぐらい) から [1126 ターン: θ の変化] の水準 (フィールドキャパシティが 0.15 ぐらい) まで $mutation = 0.02$ の場合では 700 ターンぐらいの時間しかかかっていないが、$mutation = 0.002$ の場合では 8000 ターンぐらいかかっている。

この節をまとめると、突然変異率を示す *mutation* が高いとジーンは分散型になり、逆に *mutation* が小さいとジーンは一極集中型になる。また、*mutation* が高いほうが進化は速く起こりやすくなるが、ある程度の高さ以上の *mutation* になると進化の速さはほとんど変わらなくなる。

なお、地球上の実際の生物においては、一年あたりのアミノ酸座位あたりの進化における置換率は表現型の進化速度とは関係なくほぼ一定であるため、分子レベルでの進化の速度はどの生物であってもほとんど同じである [Kimura1988]。そのため、どの生物であっても突然変異率は、ほぼ一定であると考えられ、突然変異率を自由に変えることができるのは、シミュレーションの中だけであると考えることもできる。

第5章　まとめと今後の課題

本論文では、Tierra 的手法を用いて、植物を中間表現などを通してモデル化し、シミュレーションを行った。その結果、あるモデルのシミュレーションという制約の中ではあるが、進化の方向を解析し、注目している進化の方向である種の多様性という環境に意図的に到達することができた。このような手法をさらに拡張してゆけば、進化に対する有力な情報をより多く得ることができるようになるであろうと期待している。

今後のこのような研究をしていく上での課題としては、主に次のようなものが考えられる。

- 中間表現、仮想マシン語の改良

  本論文でシミュレーションを行うのに十分であると考えた生命の定義をベースにした中間表現と Tierra 的手法の中心である仮想マシン語は、本論文のモデル化において重要な役割を果たしているのだが、この 2 つによって表現できる植物のモデルは、かなり制限されたものである。たとえば、blue-green algae などの藻の一種を表現するには適さないと思われる。このことから、本論文では計算時間の関係などで見送ったが、中間表現には植物の表現に用いられる L-System などを用いた方がより良いであろう [Doi1988, Kawasaki,Kikuchi,Shinoda1998, Yanagida1994]。また、仮想マシン語も 32 種類の中にはあまり使われていないものもあり、このような命令をほかの命令といれかえることも意味があると思われる。

- 動物へのモデルの拡張

  本論文では植物のみをモデル化したが、同様に動物をモデル化して同じフィールド上におくということも考えられる。この場合、たとえばアゲハチョウとツツジのような共生の関係を得ることもできるかもしれない。

- 系統図の作成

  本論文が用いたプログラムでは、どの種からどの種が生まれたかという情報を保有していない。これは、突然変異によって異なった種から同じ種が生まれてしまうために、もとになった種を特定することが難しかったためである。そのために新しい種がどのように発生するかというのを調べにくい状態になっている。やはり、系統図を作るための定義を何らかの形でおき系統図を作る方が、何が中で起こっているかということを解析しやすいであろう。

- パラメータの改良

  本論文では、解析の段階においてパラメータを 4 つに絞り込んでおり、またそれらの相互作用も考えていない。シミュレーションにかかる時間の関係上相互作用について調べていないが、実際にはパラメータ間の相互作用は大きな影響もありえるので調べたほうがよいと思われる。また、より一般的にどのようなものをパラメータにすべきかということも、ポイントになる可能性がある。

謝辞

本論文にあたって、指導してくださった小島政和教授、戴陽講師には大変お世話になりました。また、株式会社グッドマンの木目沢司氏と株式会社 CSK の吉川徹氏には多くの助言を頂き、とても参考になりました。この場を借りて改めてお礼申し上げます。最後になりますが、小島・戴研究室、高橋研究室、三好研究室の皆様、ありがとうございました。

参考文献

[Dawkins1991] Richard Dawkins 著、日高敏隆、岸由二、羽田節子、垂水雄二訳
『利己的な遺伝子』(紀伊国屋書店、1991)

[Doi1988] 土居洋文、『生物のかたちづくり、発生からバイオコンピューターまで』(サイエンス社、1988)

[Hattori1994] 服部 桂、『人工生命の世界』(オーム社、1994）

[Kawasaki,Kikuchi,Shinoda1998] 川崎貴裕、菊池彰人、篠田貴子、「Ｌシステムによる植物の形態発生」
http://www.ohmori.k.hosei.ac.jp/Semi98/L-system/html/tsld001.htm

[Kibe1998] 岐部篤弘、「Tierra を用いた生物の進化シミュレーション及びその工学的応用の検討」
http://www.maru.cs.ritsumei.ac.jp/~ak/Lab/thesis_rep/thesis.html

[Kimezawa1996] 木目沢司、「Tierra 入門」http://www.hip.atr.co.jp/~kim/TIERRA/tierra.html

[Kimezawa1997] 木目沢司、「Tierra アプリケーション」http://www.hip.atr.co.jp/~kim/MBT/mbt-j.html

[Kimezawa1999] 木目沢 司、『人工生命システム Tierra』(エイ・ティ・アール人間情報通信研究所、1999）

[Kimura1988] 木村資生、『生物進化を考える』(岩波新書、1988)

[Miura1995] 三浦憲二郎、『OpenGL 3D グラフィックス入門』(朝倉書店、1995)

[NASDA2000] 宇宙開発事業団 (NASDA) ホームページ、「宇宙情報センターへようこそ」
http://spaceboy.nasda.go.jp/Welcome/Welcome_j.html

[Ray1992] Ray.T.S. ,‘Evolution, ecology and optimization of digital organisms.’
(Santa Fe Institute working paper 92-08-042、1992)

[Ray1994] Ray.T.S. ,‘Evolution, complexity, entropy, and artificial reality.’
Physica D 75: 239-263

[Ray1995] Ray.T.S. ,
‘An evolutionary approach to synthetic biology: Zen and the art of creating life.’
Artificial Life 1(1/2): 195-226. Reprinted In : Langton, C. G. [ed.],
Artificial Life, an overview. (The MIT Press、1995)

[Sagan1961] Carl Sagan, ‘The Planet Venus’, SCIENCE, 24 March 1961, Volume 133, Number 3456, 849-858
(『The American Association for Advancement of Science』、1961）

[Sakai1999] 酒井聡樹、「植物の適応戦略」第 11 回 RAMP シンポジウム論文集、53-62 (1999)

[Sakai,Takada,Kon1999] 酒井聡樹、高田壮則、近雅博、
『生き物進化ゲーム – 進化生態学最前線：生物の不思議を解く』(共立出版株式会社、1999)

[Sekimura1998] 関村利郎、「生物の形の多様性と進化」
数理科学,No.415,JANUARY 1998,69-74(『サイエンス社』、1998)

[Taguchi1979] 田口 玄一、『経営工学シリーズ 実験計画法』(日本規格協会、1979）

[Tokoi1997] 床井浩平、「GLUT による「手抜き」OpenGL 入門」
http://www.sys.wakayama-u.ac.jp/~tokoi/opengl/libglut.html

[Yanagida1994] 柳田達雄、「L-システムによる植物の形態の進化」
物性研究, 61-5, 429-439 (『物性研究刊行会』、1994）