2021
秋号

百萬石
Hyakumangoku Vol.57

Published by MMM

部長挨拶

terry

terry@mma.club.uec.ac.jp

「百萬石」をお手に取っていただき，ありがとうございます．弊誌 (以下，部誌) は，電気通信大学の公認サークル MMA が新歓期 (春号) と弊学が開催する調布祭時期 (秋号) の年 2 回発行・頒布しているものです．MMA とは，”Microcomputer Making Association”の頭文字に由来しており，1975 年に創立されたサークルです．現在では計算機に関連することを中心に，ネットワークやセキュリティなどの分野でも活動しています．部誌には，MMA の部員が各々に行なっている活動を記事にしたものが掲載されています．内容は多岐に渡り，高度なプログラミングの話から旅行記など，MMA の活動の自由さが感じられるものとなっております．

さて, 本来この「秋号」は調布祭で頒布されるものでした．今年度の MMA は調布祭にオンラインで参加します．そのため，残念ながら直接お渡しすることはできません．代わりと言ってはなんですが，歴代の部誌がすべて pdf で読めるようになっております．歴代の部誌とともに Web ページでお楽しみいただければと思います．

MMA は幸い遠隔での活動が可能な方ではあります．しかし，直接会うことがないため，部員同士の繋がりは希薄なものになるのは避けられませんでした．この状況下にもかかわらず，部室の利用が可能であるからか，新入部員として活動に残ってくれる後輩が想定より多かったです．部長として嬉しい限りです．また，部員を通年募集しているため，夏秋も少しずつ部員が増えてきています．この部誌を読み興味を持った電通大生がいらっしゃいましたら是非お気軽に HP や Twitter からコンタクトを取ってください．いつでも歓迎いたします．

今年も早いもので，年末を迎えようとしています．今年の MMA では，オンラインでの部費納入システムの導入，入部届の電子化と部費の振込方法のメール自動返信が実現，と変化の多い年になったと思います．12 月には 6 月にオンラインで行った Dentoo.LT の対面開催も予定されています．徐々に活動の制限が緩和され，部員の動きが活発になり始めているように感じます．来年度は合宿も復活させたいという話も出ています．多くのサークルや学生団体が活動制限をせざるを得ない状況から解放されることを願っています．

最後になりますが，MMA はいろんな人とのつながりを得ることができます．加えて，自分のやってみたいことを相談し実現できる環境，充実した機材が揃っています．そういった自由度の高いところで活動している部員が書く部誌は面白いものばかりだと思います．ぜひご一読ください．

2021 年 11 月吉日

百萬石 Vol.57

Hyakumangoku Vol.57

2021 Autumn

MMA

目次

第 1 章

LaTeX でのイカしたオートマトンの描き方 & 関連ツールのお話

terry

概要
TikZ でオートマトンを描く記事の日本語情報が少なかったので紹介する．TikZ の automata の使い方を TeXample.net や英語記事で得た情報をもとにまとめる．加えて，有限オートマトンを課題等で扱う時に便利なツールの紹介もしていく．描き方だけを知りたい人は 1.3 まで飛ばすことを推奨する．

1.1 はじめに

1.1.1 部誌執筆のきっかけ

お久しぶりです．今年度部長の terry です．突然ですが，現在私は形式言語理論と言う授業を受講しています．その授業ではオートマトンを扱います．当然レポート課題には状態遷移図を描くことになりました．今期 1 類 CS あるある[*1]なのですが，手描きや Word レポートを受け取らない授業でした．[*2]選択科目なのでこれを理由に履修をやめる人も出ました... 実験 A や D 演習など推奨や加点と言う授業はあっても LaTeX しか受け付けないと言う授業はコンリテや基礎プロの埋めるだけテンプレ以外初でしたね．ここで，手描きの図を挿入していた人たちが詰みます．私もそうでした．ということで手描き図挿入マンを卒業して「つよつよ LaTeX マン」を目指すわけです．

この記事は LaTeX 上で図を直接かけるようになるまでの**試行錯誤と僕のやり方**を紹介します．よりよい方法や至らない部分があるかもしれませんが，その点ご了承の上参考にしていただけると幸いです．

ところで話変わりますが，担当教員がフランスの学会に行っていました．大学の研究機関としての強さを持っているということでいいと思います．全員のレポートに不正解なら理由をつけてフィードバックをし，メールでの質問対応は親切に答え，「質問に答えるために教師がいる」など学生への対応は素晴らしいです．何一つとして悪いところがない素晴らしい授業に見えますね．でも課題が激ムズなんですよね．英語論文読まされて課題に答えるのだいぶハードでした．ためにはなりますが...

[*1] CS 数値計算 (必修) と合わせて 2 個目．「どうせ卒論は LaTeX で書くから早いいうちから訓練しよう」
[*2] 曰く，「Word はダサい」「Word さんとはおさらばしましょう」

1.1.2 タイトルの由来

「Word はダサい」らしい．それなら LaTeX 直描き図はイカしてるんだろうなという思いでつけました．... 実際イカしてると思う．いいよやっぱり．

1.2 オートマトン，描きたくない？

「はじめに」でも話しましたが，流石に手描きはしんどいし，かといって外部アプリを使うのもいちいち画像挿入するのも大変．となると直接ソースに埋めたくなるわけです．一応色んな方法を考えて比較してみましょう

1. PowerPoint：Word はダサいけど PP ならいけるって？流石にちょっと... 屁理屈がすぎるかなって．PP 芸はなるべくしたくないです．
2. draw.io を使う：GUI 操作が面倒というかしんどすぎる，却下
3. JFLAP：先生オススメなので，まぁ見てみる価値がありそう
4. TikZ：普段使い LaTeX で描けるので導入コスト低，Keyboard のみ図作成 → 慣れで爆速化, よき
5. その他ソフトやサイト：事前知識 0 の環境構築で手間取って課題進まないのは本末転倒

ざっとこんな感じですかね．というわけで JFLAP と TikZ を試してみることにしました．

1.3 オートマトンの描き方

様々あるが，授業で推奨された 2 つの方法を紹介したいと思います．

1.3.1 JFLAP 編

JFLAP の描き方を紹介するといっても GUI なのでポチポチするだけ，かんたんだね！直感的に操作できるのがメリットです．雰囲気で操作してみてください．クリックで頂点が作れたり．マウスで線を引いて，移動するための記号を記せばこのようになります．(画像の薄い線は引いている途中)

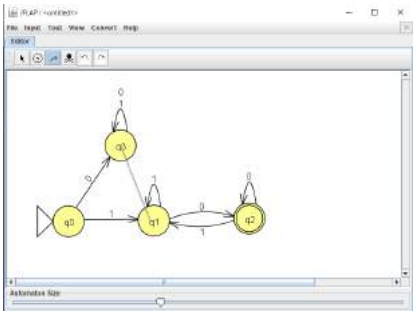

図 1.1 JFLAP の操作画面/先頭 1 の 2 進偶数を受理する DFA(多分)

右クリックで初期状態と受理状態を設定したりできます．その他にもいろいろな機能がありますが，あくまでオートマトンの描き方なのでそれはまた後ほど．

1.3.2 TikZ 編

本記事のメインコンテンツです．公式マニュアル (`https://pgf-tikz.github.io/pgf/pgfmanual.pdf`) は 1322 ページあって英語だし読む気にならない．かといって TEX Wiki の TikZ ページにはオートマトンの項目があるわけではないのでやりにくい．となってググりました．$y = f(x)$ 方のグラフの書き方，交点の計算等の記事はたくさんヒットしました．色々できて優秀な TikZ くんですが，オートマトンの描き方の日本語記事は少なく，英語情報がほとんどでした．英語は読みたくないですが，情報がないので，このチュートリアル PDF[1] をコピペなり翻訳しながらなんとか身につけました．加えて，サンプルだけなら TeXample.net の該当ページ [2] をコピペして改造することである程度理解しました．

以上のように，私でも読める平易な英語の情報はたくさんありました．そして描けるようにもなりました．ただ，やはり日本語情報があると嬉しいので，日本語でまとめてみたいと思います．

JFLAP 編のオートマトンと同じものを描きました．以下にソースコードと結果を載せます．完成品とソースコードを見比べながら描き方を説明していきます．

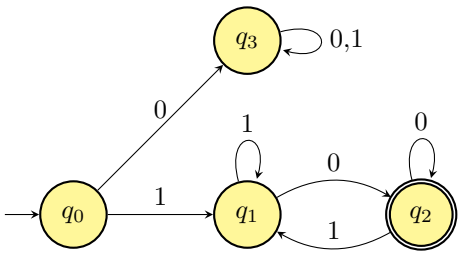

図 1.2 先頭 1 の 2 進偶数を受理する DFA(TikZ 版)

ソースコード 1.1 TikZ automata preamble

```
\usepackage{tikz}%  必要な package
\usetikzlibrary{automata,positioning,arrows} % tikz library
\tikzset{ % 初期設定
->, %makes the edges directed
>=stealth, %makes the arrow heads bold
node distance=2.3cm, %specifies the minimum distance between two nodes.
    Change if necessary.
every state/.style={thick,fill=yellow!50}, %sets the properties for each '
    state' node
initial text=$ $, %sets the text that appears on the start arrow
}
```

プリアンブルに関しては矢印の形や線のスタイル (太さ細さ点線) などのデフォルトの値を定めています．気になったらかえてもいいとおもいますが，そうでなければこのままで (出典 [1], 改造済) いいと思います．option は上書きできるのであくまでよく使うデフォルト設定を書き込むものだと思っておいてください．

ソースコード 1.2 tikz ソース

```
\begin{figure}[ht]% 画像と同じ
    \centering
    \begin{tikzpicture}
        % tikz code goes here
            \node[state, initial]                (q0) {$q_0$};
            \node[state, right of=q0]            (q1) {$q_1$};
            \node[state, right of=q1,accepting] (q2) {$q_2$};
            \node[state, above of=q1]            (q3) {$q_3$};
            \draw   (q0) edge                node[above]{0}  (q3)
                    (q0) edge                node[above]{1}  (q1)
                    (q1) edge[loop above]    node[above]{1}  (q1)
                    (q1) edge[bend left]     node[above]{0}  (q2)
                    (q2) edge[loop above]    node[above]{0}  (q2)
                    (q2) edge[bend left]     node[above]{1}  (q1)
                    (q3) edge[loop right]    node[right]{0,1}(q3);
    \end{tikzpicture}
\end{figure}
```

本体のコードを見てもらえるうと分かる通り基本は node で頂点を作り draw で辺の描画をしています．node の書き方は \node[option] (name) label; のように描きます．name はその頂点の名前です．これを基準に頂点に辺をつなげたりします．node のオプションは以下のとおりです．

- state: 必須です．node に名前をつけられるようにするオプションです．
- initial: 初期状態を表します
- accepting: 受理状態を表します
- {above,below,left,right} of (node name):(node name) の上下左右に配置します
- node [state] (q) at (x,y):xy 座標で配置を指定

\draw (start) edge[edge option] node[node option]cost/next word (goal) の形で start から goal まで辺を張ります．オプションは以下のようになってます．こちらもよく使うやつ限定で抜粋します．

- edge option
  - 何も書かないと始点と終点を直線で結ぶ．重なってしまうときは bend で曲げて対策する
  - bend {left,right}：{ 左, 右 } 周りで曲がった辺を張る
  - loop {above,below,left,right}：{ 上, 下, 左, 右 } 方向へ自己ループを張る
- node option
  - [{above,below,left,right}]：辺の上下左右にコストや文字を表示する．

他にもありますが，気になった方はまた調べてみてください．

このような形でオートマトンをマークアップでかけるようになりましたね．うれしい．

1.4 オートマトン，テストしたくない？

1.4.1 JFLAP について

JFLAP はオートマトンを含む形式言語理論の基礎部分の学習サポートをするための Java ツールです．(多分) 入手方法は公式 HP(`https://www.jflap.org/`) から Form を入れるだけ．チェックボックスと大学名とメアド入れるだけなのでそんなに辛くないと思います．Java で動くツールとなっていますのでそこだけ注意です．

このツール，基礎理論を学ぶ人向けということでいろんな事ができます．私が使っている機能は，シミュレーションと多入力テストです，順番に紹介します．めちゃめちゃ便利ツールなので英語に恐れず使ってほしいです．他にも NFA to DFA 変換や並べ替え，オートマトン以外の機能など色々できるみたいです．

1.4.2 おーとまとん☆しみゅれーしょん

・・・魔法みたいですね．

それくらい便利なんですよね．冗談はさておき．メニューから Input → Step by State などを選ぶと 1 ステップずつオートマトンの処理を追うことができます．それ以外にも，Multiple Run を選んで下の View Trace を押すと下図ように表示することもできる．課題等でしっかりできたかのテストをすることができていいですよね．

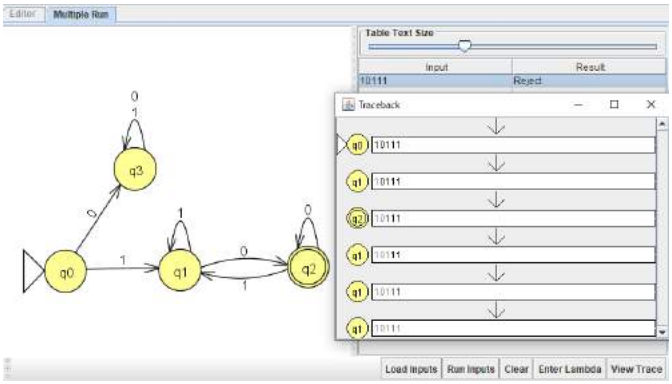

図 1.3 DFA のシミュレーション/ステップごとに見れる

1.4.3 多入力テスト

1 つ 1 つ例を試し，条件にを満たすかどうかを確認することがめんどくさくなるときがあると思います．(人力でやるのは大変ですね．1 個前のセクションで卒業できますが...) Input → Multiple Run から

右下の Load Input で txt ファイルを読み込むと便利です．予め入力内容を用意しておくと使い回せます．このように一気に事前に用意したデータ (今回は 0-31 までの 2 進表記) を読ませることができます．最後に Run Input を押すと一気に全ての実行結果が埋まります．人力よりも圧倒的に楽になりますね．

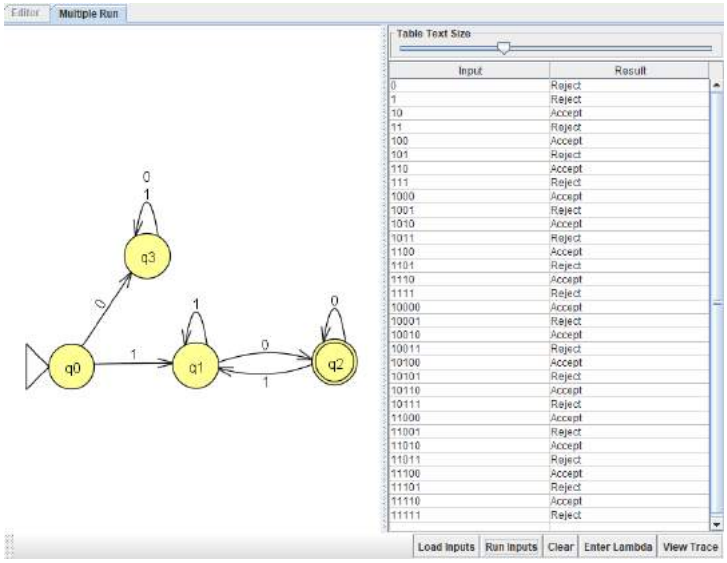

図 1.4　多入力 DFA の Accept/Reject 一覧 (これ便利)

1.5　おわりに

日本語記事が無いとどうしても辛いと思うのでまとめました．何かの役に立ってくれればいいなと思います．形式言語理論を受講している皆様，是非一緒に頑張りましょう．

最後になりますが，ここまで読んでいただきありがとうございました．我々の部誌は，人に読んでもらえなければ何にもならないので，読んでいただいた方全てに感謝です．この記事をきっかけに，Dentoo.LT#26 (12/19) の参加やMMMの活動に興味を持ってもらえたのなら嬉しいです．

参考文献

[1] Satyaki Sikdar, “Drawing Finite State Machines in LaTeXusing `tikz` A Tutorial”, 2017, `https://www3.nd.edu/~kogge/courses/cse30151-fa17/Public/other/tikz_tutorial.pdf`

[2] TeXample.net, “Example : State machine”, 2006, `https://texample.net/tikz/examples/state-machine/`

第 2 章

つかみどころのないカオスを解析したら思ったより綺麗な図が出てきた件

s_sato_grobner

2.1 自己紹介

こんちゃっちゃ〜〜〜〜〜〜〜〜〜〜〜〜(新手の挨拶)
会計の s_sato_grober です、MMA 内で雑魚やってます、III 類です　Twitter は教えません
よく勘違いされがちですが、読みはぐれぶなです　間違えたら怒ります (｀Д´#)
興味があるのは数値解析とか量子力学らへんの話です

2.2 カオスについて

2.2.1 定義について

カオスとは、非線形の系の場合について起こる現象で、決定論的でありながら (確率現象によらないが)、**初期値鋭敏性**を持ちます。わかりやすい例は二重振り子だと思います。[*1]　**初期値鋭敏性**とは初期値のわずかな差によって、全く違う振る舞いをすることです。天気予報の長期予測が難しいのは、マシンイプシロン[*2]程度の差でも初期値鋭敏性を持つためです。バタフライ効果と言った方がわかりやすいかもしれません。

2.2.2 具体例

例えばこういう系がカオスの例です。

- 二重振り子
  現象としてはわかりやすいけど、解析全然わかんねぇなパス
  三重振り子になるとさらにカオスみを感じるぴえん

[*1] 池口研究室 https://www.youtube.com/watch?v=25feOUNQB2Y
[*2] 浮動小数点で表現された数がどうしても持ってしまう相対誤差

- 非線型バネの強制振動 (duffing equation)
  $\ddot{x} + K\dot{x} + x^3 = B\cos t$
  線形だったら x^3 の項はつかない。これは復元力が x^3 に比例する項が加わっていると言うことで、解析的に解けない。(数値計算で解くしかない)
- logistic 写像
  $f(x) = ax(1-x)(3 \leq a \leq 4)$ の時、数列 $x_{n+1} = f(x_n)$ がカオスになる。
  比較的わかりやすいカオスの例、生物の個体数の世代の変化を与えている。
- ローレンツ[*3]方程式
  $dx/dt = -px + py$
  $dy/dt = -xz + rx - y$
  $dz/dt = xy - bz$
  大気変動モデルを単純化したモデル。カオス研究の発端となった。天気予報が長期予測できないのはコイツを調べるとなんとなく分かる。今回はパスさせて

今回は Duffing 方程式について解析してみます。(logistic 写像は非常に面白いのですが、紙面の都合上省きます　UEC Advent に書くかもしれません)

[*3] ちなみにローレンスという物理学者もいるので非常に混乱する、ローレンスゲージはローレンツ不変のゲージ (ややこしい)

2.3 Duffing 方程式の解析

簡単のため、カオスについての性質が出てくる

$$K = 0.05, B = 7.5$$

で考えます。

2.3.1 Runge-Kutta 法で解いてみる

Duffing 方程式は次のような形をしていました。

$$\ddot{x} + K\dot{x} + x^3 = B\cos t$$

これは非線形の方程式ですので、解析的[*4]に解くことができません。そこで、数値的に解くことにします。(一応、非線形で解ける形として kdV 方程式がありますが、ソリトンの話をするときにもう一度やりたいですね)

解きたい範囲について、分割して離散化し、その各所の値を随時的に求めていきます。解く方法としては Euler 法、修正 Euler 法、Runge-Kutta 法がありますが、計算コストに見合った精度が出る Runge-Kutta 法を使います。

次の微分方程式を解いてみます。

$$\frac{dy}{dx} = f(x, y)$$

$y(x)$ がわかっているとき、微小変化 h した時の $y(x+h)$ を求めます。次の式を順番に計算していきます。

$$k_1 = hf(x, y)$$

$$k_2 = hf(x + h/2, y + k_1/2)$$

$$k_3 = hf(x + h/2, y + k_2/2)$$

$$k_4 = hf(x + h, y + k_3)$$

の時

$$y(x + h) \simeq y(x) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

のように計算できます。よって、これを随時的に計算していけば良いです。

今回の場合は 2 階の微分方程式ですから、次のような連立微分方程式として解くことになります。$v = dx/dt$ とおけば

$$\dot{v} = -Kv - x^3 + B\cos t$$

[*4] 微分方程式を式変形で解くこと

より、次の x, v についての連立微分方程式となって

$$\frac{dv}{dt} = f(t, x, v) = -Kv - x^3 + B\cos t$$

$$\frac{dx}{dt} = g(t, x, v) = v$$

よって Runge-Kutta 法を連立微分方程式について拡張して解きます。プログラムとしては次の通りになります。

```
#include <bits/stdc++.h>
using namespace std;
#define rep(i, n) for (int i = 0; i < (int)(n); i++)

double K = 0.05;
double B = 7.5;

double f(double t, double x, double v){
  double ans;
  ans = -0.05 * v - (x * x * x) + 7.5 * cos(t);
  return ans;
}
double g(double t, double x, double v){
  return v;
}
int main() {
  cout << fixed << setprecision(10);
  double a,h;
  int n;
  double x,v,t;
  double k,k1,k2,k3,k4;
  double l,l1,l2,l3,l4;
  string filename = "duffing.dat";

  cout << "Specify the value of a" << endl;
  cin >> a;
  cout << "Specify the number of steps" << endl;
  cin >> n;
  cout << "Spefify the value of x_0" << endl;
  cin >> x;
  cout << "Specify the value of v_0" << endl;
  cin >> v;


  h = a/n;
  cout << h << endl;
  ofstream ofs(filename);
  rep(i,n+1){
    t = h * i;
    ofs << t << " " << x << endl;
    cout << t << endl;
    k1 = h * f(t,x,v);
    l1 = h * g(t,x,v);
    k2 = h * f(t+h/2,x+l1/2,v+k1/2);
    l2 = h * g(t+h/2,x+l1/2,v+k1/2);
    k3 = h * f(t+h/2,x+l2/2,v+k2/2);
    l3 = h * g(t+h/2,x+l2/2,v+k2/2);
    k4 = h * f(t+h,x+l3,v+k3);
```

```
    l4 = h * g(t+h,x+l3,v+k3);

    k = (k1 + 2*k2 + 2*k3 + k4)/6;
    l = (l1 + 2*l2 + 2*l3 + l4)/6;
    v = v + k;
    x = x + l;
  }
}
```

また、この結果をグラフに示すと次の図 2.1 の通りになります。(初期条件は $x = 3, v = 6$ と $x = 3.0001, v = 6.0001$ の場合を解いた) より、初期条件がほんの小さい差だったとしても、次第に大き

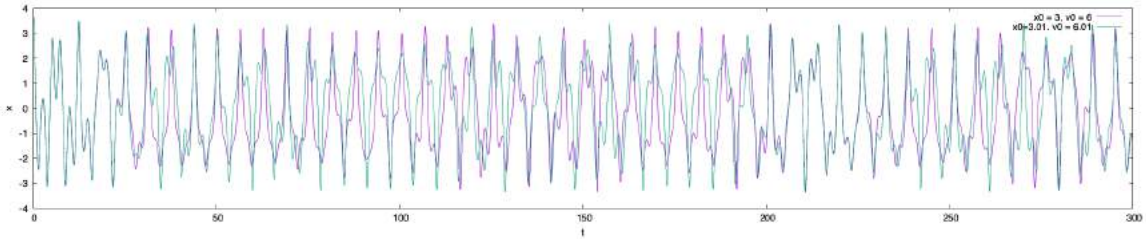

図 2.1 Duffing 方程式の計算結果

な差になってしまうバタフライ効果を得ることができました。これは線形の微分方程式では起こらない現象です。

2.3.2 アトラクターについて

実は Duffing 方程式の解は非常に面白い特徴を持っています。位置と運動量を横軸、縦軸にとった $x - p$ グラフにプロットしてみます。線形バネであれば、ハミルトニアン (全エネルギー) は一定なはずなので

$$H = \frac{p^2}{2m} + \frac{1}{2}m\omega^2 x^2$$

より、$x - p$ グラフは楕円となるはずです。対して Duffing 方程式の解をプロットすると次の図 2.2 の通りになります。線形バネに比べてかなり複雑なグラフに思えます。

これだけだとわかりにくいので、$t = 2n\pi$ の点のみを抽出してプロットしてみます。(図 2.3) 渦巻き状のような不思議な形を得ることができました。このことを**アトラクター**とよび、一種のカオスではしばしばこのようなアトラクターを得ることができます。Duffing 方程式のアトラクターは日本の物理学者の上田先生によって発見されたので、**上田アトラクタ**、または**ジャパニーズ・アトラクタ**と呼ばれます。

また、とる点の位相を変えることでこのアトラクタがうねうね[*5]しているところをみることができます。詳しくは次の Youtube の動画をみてください。`https://youtu.be/xgxJizxMME4`

初期値鋭敏性をもち、めちゃくちゃな軌道をとっているように思えるカオス現象ですが、見方を変えればこのような綺麗な図形を得ることができるわけです。

[*5] 僕は課題が進まない時にうねうねします

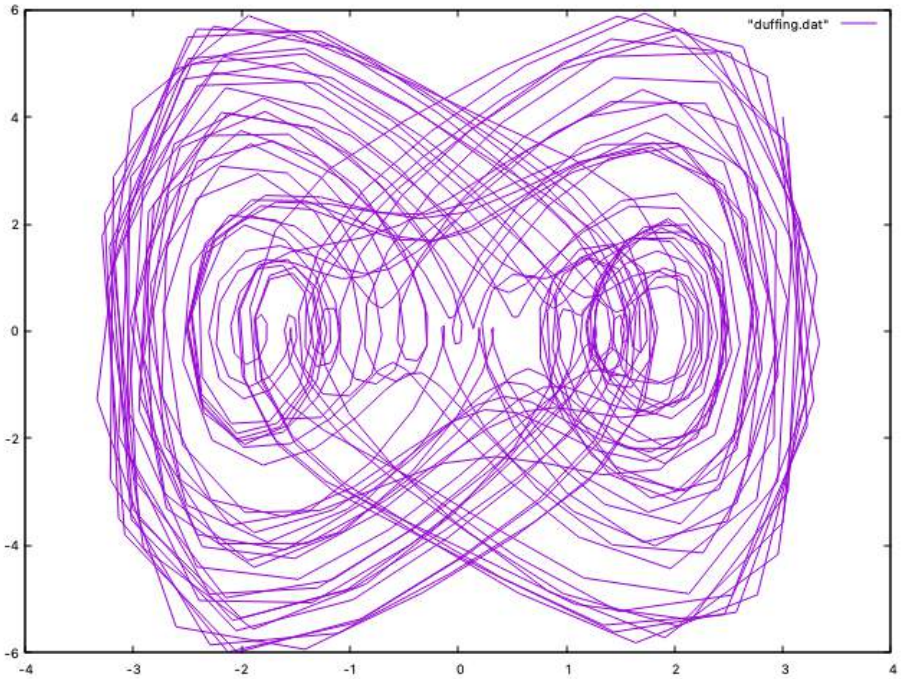

図 2.2　Duffing 方程式の解の x-p グラフのプロット

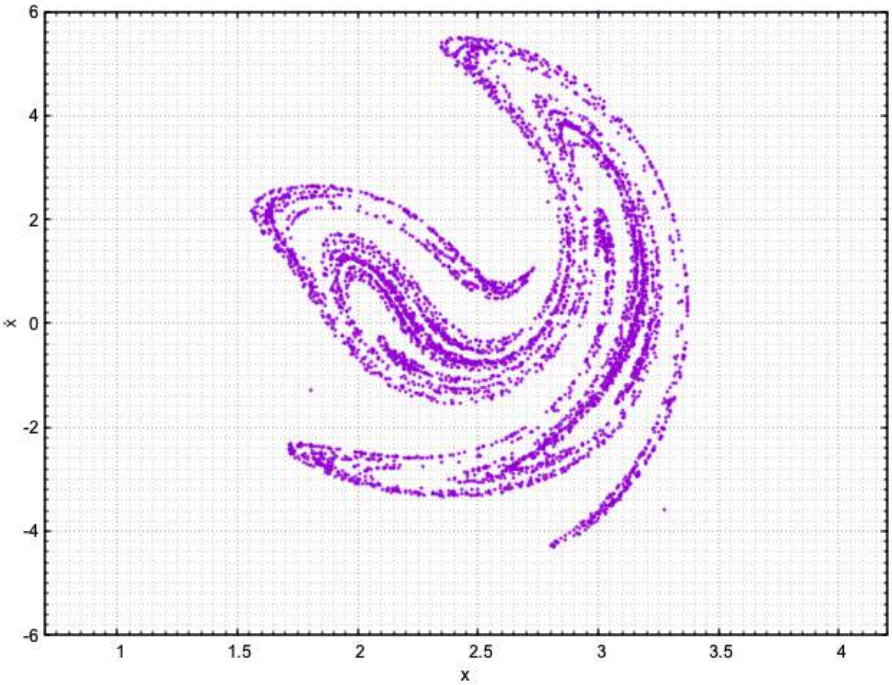

図 2.3　Duffing 方程式のアトラクター

2.4 最後に

このテーマは実は 3 年前期の上級コンピュータ演習の内容なんですが、思ったより面白かったので紹介したいと思って記事を書きました。進捗をあげなすぎて教授っちには迷惑をかけましたが... まぁそれは耐え (何が？)

非線形現象は数値計算で確認することが多いので、僕的には楽しいことが多いです。次回記事を書くとしたら量子コンピュータ関連か、ソリトンとかについて書きたいと思っています。

ありがとうございました。

第 3 章

カラオケで JOYSOUND に入っていない曲を歌う方法

tonphy

3.1 始めに

tonphy です。オタクはマイナーな曲が好きです。でも、マイナーな曲は当然あまりカラオケのラインナップにはありません。歌いたい曲、練習した曲が無くてガッカリしたオタクも多いでしょう。または、カラオケで絶対本人映像で歌いたい、かわいい MMD を流しながら歌いたい、という人もいるかもしれません。

ここでは、カラオケで機種が JOYSOUND の場合に好きな曲や動画を流す方法を紹介します。~~DAM は知りません。~~

3.2 前準備

3.2.1 アプリインストール

まずは JOYSOUND の曲予約をスマホから行える『カラオケ予約-キョクナビ JOYSOUND』というアプリをインストールします。検索で JOYSOUND と打てば大体見つかります。カラオケボックスとかでも多分インストール方法が書いてあります。

3.2.2 wifi の接続

アプリを入れたら、カラオケボックスに備え付けられた JOYSOUND のフリー wifi に接続します。その後、曲を予約する端末から、スマホと連動をするための QR コードを表示してスマホで読み込みます。すると、スマホから曲の予約などができるようになります。

3.3 動画の共有

3.3.1 youtube の動画を共有

前準備でスマホとカラオケ端末を連携したら、スマホから youtube の動画を検索して選択すると、カラオケのディスプレイで動画を流す事ができます。当然カラオケ音源ではなく動画そのままが流れますが、そこは譲歩してください。また、曲の予約と同じ感覚で動画を選択すると、現在流れている曲がストップしてしまうので友達と来ている場合は気をつけましょう。

3.3.2 カメラロールの動画を共有

youtube にすら上がっていないような超マイナーな曲を歌う場合や、新曲が発表されてすぐにカラオケに来た場合などは、自分で動画を作ってカメラロールに保存しておきましょう。スマホのカメラロールに保存されている動画も流す事ができます。mp3 のデータを持っている場合は、MP3Tube などで動画に変換すると簡単です。ちょっと工夫したい人は、動画編集などで歌詞をつけると歌いやすくなるかもしれません。iPhone だとカメラロールに保存すれば OK なのですが、Android は触った事が無いので、正直わからないです。多分 iPhone と同じようにカメラで動画を撮る時と同じ場所に保存すればいいと思います。

3.4 終わりに

カラオケで自分の好きな曲が流せるのはとても良いものです。自分は 1 番はコーラス無し、2 番はコーラス有り、のように曲を編集して MP3Tube で変換したりしています。長すぎない程度にメドレーを作るのも良いかもしれません。自分の好きな動画を流して、よりカラオケを楽しみましょう。

第 4 章

言語選択はとっても大事なことである (C,Ruby,python 編)

initial_d

4.0.1 対象について

今回比較対象にする言語は以下のとおりである。それぞれ学校内の授業で使用することがある言語、あるいは atcoder が公式でテキストなどを出していることから選択している。

C: 数値解析およびプログラミング演習 (2 年前期) で使用/基礎プログラミングおよび演習 (1 年後期) で使用

C++:atcoder のサイト内解説 (APG4b)

python3: アルゴリズムとデータ構造 (3 年前期) で使用/アルゴリズム技能検定公式テキスト解説

Ruby: 基礎プログラミングおよび演習 (1 年後期) で使用

4.1 言語の特徴

4.1.1 TL:DR

C 系統は高速だが便利な機能にやや乏しい
Ruby,Python はモジュールやライブラリにより便利な機能を使えたり、素早く書き上げることなどができるが遅い
Python には PyPy という条件が付くが高速に動かせるコンパイラが存在するのが利点
マラソン形式のコンテストでは実行速度が問われるので基本 C 系統の独壇場

4.1.2 大まかな各言語の傾向

C 言語

メリット 非常に高速であること、メジャーな言語であること、様々な方向に利用されて老いること
デメリット 文法に厳しい、静的型付けであること、オーバーフローの危険性と隣り合わせ

Python3
メリット pypy という高速で動かすコンパイラが存在、ライブラリが豊富、機械学習などに利用
デメリット 遅い、文法にやや厳しい
Ruby
メリット 無茶なコードの書き方ができる、 文法が緩い、利便性が高い、web サイトに利用
デメリット 非常に遅い、若干マイナー

4.2 大まかな言語ごとの特徴

4.2.1 整数、小数、複素数

C は整数にも複数の型が存在し、オーバーフローや要求される精度に注意して型を選ぶ必要がある。複素数は complex クラスという形で存在している。
また、ビット演算をするための整数型がある。
python は整数/浮動小数点型/複素数の 3 種類のみ、ビット演算は整数で使える。
Ruby は整数/浮動小数点型のみ、複素数はクラスとして存在、ビット演算は整数で使える。
Ruby/Python は自動的に桁数が拡張されるのでオーバーフローの危険はないが、桁数増加につれ負荷が増大し遅くなる。

4.2.2 配列/タプルなど

C の配列は型の指定が必要/中身に複数の型のものを混在させられない/個数を後から変更できない/範囲外参照でセグメンテーション違反を起こす
また、C++ にはペア、タプルなどの複数の値を組にして扱うことができる型が存在する。
タプルは python に同名のものが存在しているが、編集ができないだけで、ほとんど配列と同じ扱いのものである。
ruby と python では配列の中に複数の型を混在させられ、個数を後から変更できる。
python は配列外を参照しようとするとエラーを返し終了する。
ruby は範囲外を参照すると nil を返すか、代入を行う際は間を nil で埋めて書き込もうとする。
C にはソートを手作業で記述する必要がある、Python と Ruby には実装されている

4.2.3 文字列

C での文字列は char 型の配列という、文字コードを一つずつ並べた数字の配列であらわされる。
Ruby,Python では文字列のクラスがあり、大文字化や置換などが行える。
C,Ruby,Python ともに実質的に配列であることに変わりはない。

4.2.4 辞書型

C の標準機能にはないが、C++ に std::map として存在している。
ruby,python ともどもほとんどキーと内容という形で、同じ形で標準で存在している。
python は辞書型のキーに配列を使用できない。Ruby ではキーに配列を使用できる。

4.2.5 関数

基本的に C,Ruby,Python ともに定義のやり方などにあまり大きな違いはない。
ただし C のみ関数の受け取りの際に型を指定しなければならないことや、厳密には Ruby の関数はすべてメゾットであることなどがある。

4.3 まとめ

全体的に見て C 系は実行速度、Ruby,python は書きやすさや機能の多さなどで優れているといえる。競技プログラミングにおいて実行速度が速いというのは長所であり、C++ のような高速で動作する言語を習得するべきである。それ以外の用途については、それぞれの言語ごとに特色が強く出ているので、自分の興味のある分野につながる言語を学ぶべきである。また、ここで扱う以外にも言語は多数存在しているので、自分に合う言語を探すことも大事である。

第 5 章

ノート PC の HDD を SSD に換装した話

maccha

私は電通大 B3 の不審者こと、maccha である。幣学では僭越ながら、すみっこぐらしのプロを名乗っている[*1]。たまにアニメを見たり、漫画を見たりしている。最近は VTuber が好き。

MMA では、私は書記として部会の議事録にあることないこと書いている (そんなことはない)。部の議事録を自分色に染め上げたい部員は、ぜひ次期書記として立候補してほしい。MMA の歴史を作るのも、塗り替えるのも、ぶち壊すのも、未来の書記次第だ。

さて、この大学に入学してから 3 度目の調布祭を迎える。この 2 年半はあっという間に過ぎていった。遠隔講義が始まった昨年 4 月からは、講義と課題以外はずっと布団の中にいたので、一年の大半を布団の中で過ごしていたと言っても過言ではない。読者諸君は、電通大の Unique で Exciting な Campus ライフと、弊部 MMA の刺激的なサークルライフを 2 年半も送ってきた私の記事を読むことで、今後の人生に大きな変化が訪れるなんて、淡い期待をしてはいけない。決してそんなことはない。あったとしても、それは間違いなく気のせいである。そんな甘い話は、少なくともこの記事にはない。崇高なる他の方の記事をご覧になるか、各自精進するように。

最近、ストゼロがおいしい。こいつはキマる。空きっ腹のまま流し込んで、アルコールを全身に回しています (良い子は真似しないこと、戻れなくなる)。ずっと、ストゼロに浸かっていたい。こんな僕を憐れむな (生類憐れみの令[*2])。

5.1 ノート PC の HDD を SSD に換装した話

この記事では、技術的な内容で掘り下げるようなことをしてない。サクッと読めるように書いたつもりなので、自分も同じことをやってみたいと思った読者諸君は、ぜひ下調べをするように。

[*1] かの有名な「すみっこぐらし」とは全く関係がない。ガチの陰キャ。

[*2] 「しょうるいあわれみのれい」と読む。江戸幕府の第 5 代将軍である徳川綱吉によって制定された。課題というムチに打たれ続ける電通大生とは無縁な話なので、気にしなくて良い。

5.1.1 動機

現在サブマシンとして使っているノート PC は、およそ 6 年前に購入した。ここ数年、ストレージの使用率が 100% に至りまくるので、いよいよ HDD も限界かと感じ、SSD に換装することを決意した。一応、HDD はまだ読み書きができるので、今後は外部ストレージとして利用するつもりだ。ストレージ換装前のスペックを表にまとめる。

OS	Windows 10 (購入時は Windows 8.1)
CPU	Intel Core i7-4710MQ
RAM	8 GB
Storage	HDD 1 TB

なお、このマシンは長年使ってきたので、一度 OS を改めて入れ直したいと思っていた。そこで今回は、HDD のクローンを SSD に作ることはせず、空っぽの SSD に Windows をインストールし、これまで使っていた HDD から必要なデータだけ取り込むことにした。

5.1.2 準備

購入した SSD は、Crucial の 1000 GB、2.5 inch、7 mm のものであった (9.5 mm アダプター付)。SSD を選ぶ際に注目した点は、容量 (1000 GB)、インターフェース (SATA)、サイズ (2.5 inch) だ。同時にケースも購入した。購入したのは、玄人志向のエントリーモデルのケースだ。以上のパーツを Amazon で購入した。SSD は 10,697 円、ケースは 1,076 円であった。次の図が購入したパーツである[*3]。

なお、使っていた HDD の奥行きが、購入した HDD/SSD ケースと接続状態であればピッタリであっ

[*3] 電子レンジは付いてこないので要注意。また、電子機器などの精密機械にマイクロウェーブを当てる行為は危険な気がするので、絶対に行わないこと。フリじゃないから!

た。つまり、HDD とケースを接続すると、HDD がケースにちょうどはまりそうな奥行きである一方、スライドさせて接続するには、当然スライドの前に HDD がインターフェースに接続されずに入り込めるようなスペースが必要なわけだが、そのスペースが足りなかった。もう少しケースの奥行きがあれば、HDD が入り込むスペースができ、スライドさせてケースと接続が可能であったはずだ。このため、このケースを使うことができなかった。後日、別のスタイルのケースを買うことにした。

5.1.3 ストレージ換装と OS インストール

予めバッテリーを取り外した (これ大事)。ストレージ格納部のカバーを開くと、HDD にアクセスできる (次図)。

HDD を後ろ (図では左方向) にスライドさせると、インターフェースから取り外すことができる。取り外した HDD は案外小さくて軽く、最近の iPhone の方がもっと重いのではと感じてしまうほどであった。

次に、購入した SSD を先ほど HDD があったスペースに入れた。SSD の幅は格納部にぴったりだったが、高さが足りず、底にスキマが空いてしまった。そこで、偶然 SSD の付属品として付いてきたスペー

サー[*4]をその虚無スペースに敷くと、ちょうど SSD の接続部が PC 側の接続部と合う高さとなった。

続いて、OS のインストールをした。予め作成した Windows 10[*5]のインストールメディア (USB メモリ) を PC に挿し、PC の電源を入れた。SSD にはもちろん OS が入っていないので、すぐさま SSD を飛ばして USB メモリからブートされ、Windows のインストール画面に遷移した。Windows のインストールはすんなりと完了した[*6]。その後、各種必要なアプリケーションをインストールした[*7]。

さらに今回は Ubuntu もインストールし、デュアルブートを実現したいので、ストレージ上に Ubuntu 用のパーティションを用意した (パーティション分割)。

Ubuntu のインストールメディアを作成し、一度 PC の電源を切った。インストールメディアを接続して、PC を起動させると、そのまま Ubuntu のインストール画面に進んだ。デュアルブートを実現させるために、OS のクリーンインストールは選択せず、先ほど作成した専用のパーティションをインストール先として選択した。その後、何事もなくインストールが完了した。

Ubuntu をインストールした後は、OS のアップデートや必要なパッケージのインストールなど、環境を整えた。特に Wi-Fi に接続するために、NIC のドライバーをインストールする作業には苦労した。

ストレージ換装後、Windows は起動も早く、処理もサクサク進む。Ubuntu も難なくスムーズに動作するので、デュアルブートの状態も良さそうである。

5.1.4 追記

ここまで記事を読んだ諸君に、一応注意喚起をしておきたい。詳しい方なら、特に気にすることではない。

[*4] さりげない気遣いのできる Crucial くん、マジイケメン彼氏。僕の心のスキマも埋めてほしいなぁ...

[*5] 性能要件が満たないため、Windows 11 はインストールしなかった

[*6] 近年のベンダー製 Windows 機は、プロダクトキーが BIOS に保存されているらしく、OS インストール時に登録する必要がない

[*7] もちろん Google Chrome も例外ではない、Microsoft Edge は Chrome のインストールのためだけにあるのだ (過激派)

- 保証期間内の PC を分解すると、保証対象外となることが多い (要注意)
- パーツの規格やサイズなど、購入する際 (特にオンラインストア) には、改造する PC の要件を必ず確認すること
- ノート PC の構造によっては、マザーボードに直接アクセスしたり、キーボードを取り外したりと複雑な工程を要することがあるので、事前に調べた上で改造するか判断すること
- デュアルブート化は失敗すると取り返しのつかない事態を引き起こしかねないので、バックアップは事前に作成しておくこと

個人的な意見だが、同機種の改造をしているサイトを閲覧したり、動画を視聴することで実際の作業を見学することを勧めたい。

5.2 おわりに

自宅にはもう 1 台、ストレージを換装したいノート PC があるが、そちらはマザーボードに直にアクセスしなければ、取り外しができない (めんどくさそう)。規格は PCI-e である。

5.2.1 近況報告

最近、HHKB の 25 周年を記念した雪モデルのキーボードを購入した。タイプ音がスコスコ鳴り、まるで Slack の通知音のようだ。

あと、ちょうどこの記事を書き上げた日に、私が住む埼玉県で、埼玉バーチャル観光大使に「春日部つくし」[*8]が選ばれた (かわいいし、かわいい)。VTuber 文化が好きな私としては、観光大使として VTuber が起用されることがうれしくて仕方ない。

5.2.2 告知

12 月に UEC Advent Calender を開催する。12/1 から 12/25 のクリスマスまで毎日、電通大生の有志が記事を書いて、カレンダーを電通大に染める企画だ。例年、有志によって開催される。ぜひご覧になってほしい。また、電通大生であれば、誰でも参加可能だ。公序良俗に反しない限り、どんな記事を書いても良いので、興味ある学生はぜひ参加してほしい (参加者多数のため、その 2 が立てられた)。

UEC Advent Calendar 2021 `https://adventar.org/calendars/6400`
UEC Advent Calendar 2021 **その** 2 `https://adventar.org/calendars/6598`

あと、~~承認欲求を満たすために~~ Twitter をやっているので (@macchaakamaccha)、電通大生は差し支えなければ絡んでほしい (閲覧注意)。

[*8] Twitter: @kasukaBe_nyoki

第 6 章

OX ゲームの全探索をやってみよう！ nasatame

6.1 目的

この記事は、OX ゲームの全探索を実装し、実験した結果についてまとめる。この記事の目的は、私が OX ゲームをプログラムで解析し知った知識を誰かに共有すること、ゲーム情報学の分野に親しんでもらうことである。想定読者は、学部 2 年生、3 年生である。その他の読者については、以下の知識を参照しながら読んで頂きたい。記事中では分かりやすくするため定義に厳密でない部分があることと、日本語だと Wikipedia で検索ができないため意図的に英語を使っている点に注意して欲しい。[*1]。

- 木（グラフ）の根、枝、節点、葉などの単語の定義 (適当なサイト)
- 木の探索 [1]
- ゲーム情報学の基礎的な単語 [2]

6.2 方法

6.2.1 ゲーム木とその列挙

チェス、オセロ、OX ゲームなど、手を打って盤面を変化させていくゲームは展開形ゲームで表すことができる。展開形ゲームで表すというのは、ゲーム木で表すことである。ゲーム木は、木を使ってゲームを表したモデルで、ゲームの開始状態を根、盤面を節点、手を枝、終端を葉としている。ゲーム木を列挙するとは、ゲームの開始状態を根として、根のみが存在する木の状態から、手を枝として伸ばし、手が打たれた盤面を節点として追加してくことである。ゲーム木を完全に列挙することを全探索という。全探索をすることでゲームの性質が分かったり、そこから様々な解析ができたりする。展開形ゲームと展開型ゲームなどのように、本によって用いられている言葉が異なったりするが、意味は同じこともある [2]。図 6.1 に、ゲーム木の例として OX ゲームのゲーム木の一部を示す。具体的には開始状態から伸びる枝は本来 9 本だがここでは 3 本にしている。

[*1] 本記事とプログラム中では、OX ゲームの”OX”は、オーとエックスとなっている。本来 ◯× とするべきだが、面倒なのでこうする。

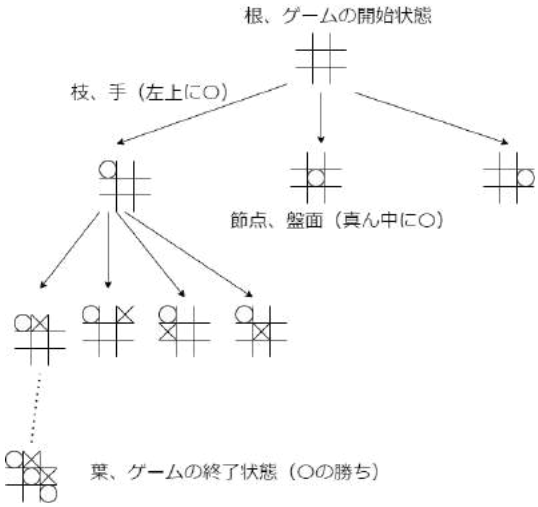

図 6.1　OX ゲームのゲーム木

6.2.2　MinMax 法とは

ゲーム木が完全に列挙されると、MinMax 法による勝利プレイヤの解析ができる。MinMax 法では、数学の帰納法の様に勝ち負けが分かっている状態から解析を初めて、親節点に向かっていく。不確定な要素のないゲームでは、**現在の盤面から勝利するプレイヤを決定できる**という性質がある。勝利するプレイヤを数値を用いて表すことが多くそれをゲーム値という。例えば、先攻が勝つなら、＋１、後攻が勝つならー１、引き分けなら０である。これは将棋や囲碁などでも言えるが、これらのゲームではゲーム木があまりに大きいため極めて長い時間がかかる。図 6.2 に、MinMax 法の概要を示す。丸い節点は先攻の節点のため、ゲーム値が大きいほど有利という前提を置くと、子節点の内、最大のゲーム値を自分のゲーム値とする。四角の節点は後攻の節点のため、ゲーム値が小さい方が有利という前提を置くと、子節点の内、最小のゲーム値を自身のゲーム値とする。

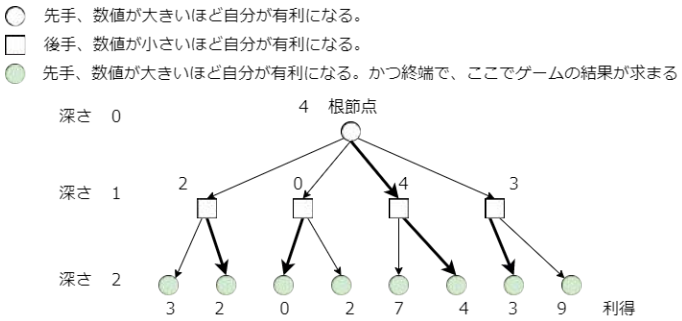

図 6.2　MinMax 法の概要

6.2.3 ゲーム木のノードの圧縮

図 6.3 は、縦 1、横 4 の盤面で行う 2 目並べのゲーム木である。図の色付きの節点は、完全に同じ状態であり、ゲーム木の節点が圧縮できる様子を表している。図 6.3 から、同じ盤面が違う手順でも現れることが分かり、特に手順によりゲームの勝敗が変化しないゲームでは同じ盤面の圧縮ができる。それを行うためのアルゴリズムとして、Transposition table と Zobrist hasing がある。Transposition table は、盤面をハッシュに変換しそれを辞書型のキーとして使うことで盤面を列挙するとき圧縮をするアルゴリズムである。Zobrist hashing は盤面をハッシュに変換するアルゴリズムであるが、OX ゲームは簡単なゲームであるため盤面をそのままハッシュとして利用することにした。ちなみに、同じ盤面でも手順により違う結果になる例として将棋の千日手というルールがある。

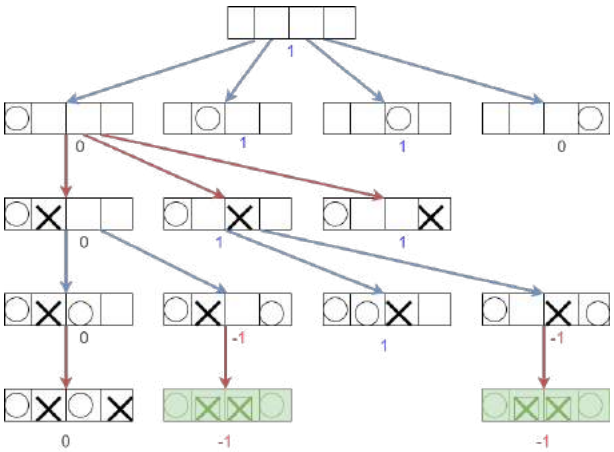

図 6.3 ゲーム木の節点が圧縮できる様子

6.2.4 実装について

細かな実装については、`https://github.com/nasatame/TicTacToe` にソースコードがある。ここでは、おおまかに説明する。

- STONE Board に置いてある石の状態を表すクラス。何もない、白（O)、黒（X）の 3 状態としてある。
- Board Bitboard というテクニックを使って、盤面を 2 進数で表現している。
- Hand 手を、置く位置と石の種類で表している。
- GameState 引き分け、試合中、白勝ち（O 勝ち)、黒勝ち（X 勝ち)。
- Node ゲーム木の節点。
- BoardOpe Board クラスの大小関係を計算する

- HashTable 辞書型、キーを盤面のハッシュ値、バリューを節点。
- NodeTable 節点のテーブル
- PlayerBase プレイヤやコンピュータプレイヤのベース。
- TicTacToeGame OX ゲームのゲームを進行するためのマネージャクラス。
- HumanPlayer 人間がゲームをするときに必要。
- RandomAI ランダムな AI。
- StrongAI 最も自分の勝ちに近い手しか打たない AI。どうせ負ける状態だとやる気を失う。
- WeekAI 最も自分の負けに近い手しか打たない AI。

6.3 結果

6.3.1 ゲーム木の展開

OX ゲームのゲーム木の展開を行った。通常の全探索の際のゲーム木の深さごとの節点数と、ハッシュ表を使った場合のゲーム木の深さごとの節点数を表 6.1 に示す。開始状態は深さ 0 としている。圧縮ありの節点数の合計は、549946。圧縮なしの節点数の合計は、6046。比較すると圧縮ありの節点数が 90 倍近く少ないことが分かる。

表 6.1 OX ゲームのゲーム木の節点数

深さ	圧縮なしの節点数	圧縮ありの節点数
0	1	1
1	9	9
2	72	72
3	504	252
4	3024	756
5	15120	1260
6	54720	1680
7	148176	1260
8	200448	630
9	127872	126

6.3.2 ゲーム木の後退解析

MinMax 法を使いゲーム木の後退解析を行った結果、開始状態から互いに最善手を打つとゲームが必ず引き分けになることが分かった。

6.3.3 AI 同士の対戦実験

MinMax 法を用いて、勝てる状態からなら必ず勝つ AI と負ける状態からなら必ず負ける AI、ランダムに手を打つ AI の 3 種の AI を作成した。それぞれを 1000 回対戦させると表 6.2 のような結果になった。

表 6.2 対戦結果

先手 vs 後手	先手勝ち	引き分け	後手勝ち
最弱 vs 最弱	1000	0	0
最弱 vs ランダム	269	101	630
最弱 vs 最強	0	0	1000
ランダム vs 最弱	879	40	81
ランダム vs ランダム	595	154	251
ランダム vs 最強	0	197	803
最強 vs 最弱	1000	0	0
最強 vs ランダム	990	10	0
最強 vs 最強	0	1000	0

6.4 考察

6.4.1 OX ゲームの対戦結果から分かるゲームの特性について

理論通り最強と最強を戦わせると、必ず引き分けになることが明らかになった。逆に最弱と最弱で対戦すると必ず先攻が勝つことも分かった。つまり、後攻が全力で負けようとすると先攻が否が応でも勝ってしまうということだ。例えば、現実においては”じゃんけんで決めるのはいつもやってるし、OX ゲームで勝った方が仕事をやることにしようぜ！お前先攻な！”の様に誘導すると必ず相手に仕事を押し付けられるということである。

6.4.2 ハッシュテーブルを用いることによる高速化効果

ゲーム木の展開の結果から理論上は、計算速度は節点数に比例するため圧縮した方が 90 倍近く高速になることが分かる。

6.5 おわりに

6.5.1 まとめ

OX ゲームの全探索と MinMax 法を行い、その結果から 3 種類のＡＩを実装し対戦させゲームの特性についてまとめた。OX ゲームは、最善手を打つと引き分けになることと、探索の過程で盤面の圧縮により 90 倍近く高速化が出来ることが分かった。

6.5.2 今後の展望

今回は、実装量の関係で盤面の回転、反転を使った圧縮は行わなかったが、実装により 8 倍近くの高速化が見込まれる。

参考文献

[1] プログラミングコンテストチャレンジブック , 秋葉 拓哉 , 岩田 洋一 , 北川 宜稔
[2] ゲーム情報学概論 - ゲームを切り拓く人工知能 - , 保木 邦仁, 三宅 陽一郎 , コロナ社
[3] ゲーム理論 , 岡田 章 , 有斐閣

百萬石 2021 -秋- © 電気通信大学 MMA

2021 年 11 月 16 日　初版第一刷発行　【本書の無断転載を禁ず】

著　者
表　紙
編集者
発行者　電気通信大学 MMA
発行所　電気通信大学 MMA 部室
〒182−8585 東京都調布市調布ヶ丘 1−5−1 サークル会館 208 号室
https://www.mma.club.uec.ac.jp/
印刷所　電気通信大学 MMA 部室
製本所　電気通信大学 MMA 部室


製作室
(MMA)
MMA
SECCON
MMA
Welcome to MMA
M.M.A.
電気通信大学
MMA
部誌です