6. 要求仕様書

1. 要求機能（課題を解決するために，プログラムに要求する機能を箇条書きで記述せよ）

・超音波センサーで壁までの距離を測定する
・モーターを動かしボール（壁）の方へ移動する
・距離に合わせて速度を変える
・光センサーで目標物（赤玉、青玉、壁）の色を判別する
・目標物が赤玉以外の場合、右側周辺の探索を行う
・目標物（赤玉）が周辺に存在しない場合、別の目標物を探索する
・アームを制御して赤玉を打つ

2. 要求条件（ソフトウェアの使用条件，資源条件，性能条件などを記述せよ）

・レゴマインドストリーム NXT
・デスクトップパソコン（学科計算機室）またはノートパソコン（各自所有）
・Bluetooth ドングル
・Visual Studio 2015 Express Edition
・Fantom ドライバ
・ソフトウェア開発キット（SDK）

7. 外部設計書

1. 実現機能（要求仕様書の要求機能を入力，処理，出力を明確にして記述せよ）

・超音波センサーで壁までの距離を測定する

入力：超音波センサー

処理：距離を測定する

出力：壁までの距離の数値

・モーターを動かしボール（壁）の方へ移動する

入力：超音波センサーの数値に対応する現在の位置情報

処理：位置情報に従ったモーターの最適速度で前進

出力：モーターの出力（前進）

・距離に合わせて速度を変える

入力：超音波センサー

処理：GA による最適速度を距離から得る

出力：モーターの回転速度

・光センサーで目標物（赤玉、青玉、壁）の色を判別する

入力：光センサー

処理：目標物（ボール）の色を判別

出力：目標物の色の数値

・目標物が赤玉以外の場合、右側周辺の探索を行う

入力：光センサー

処理：右側周辺に向きを変え前進し、色を判別

出力：右モーターの出力（右に回転)、モーターの出力（微量前進）

・目標物（ボール）が異なる場合、別の目標物を探索する

入力：光センサー

処理：目標物の色を判別し、赤以外の場合、後退・向き調整し、探索を続ける

出力：モーターの出力（後退）（向きの回転）

・アームを制御して赤玉を打つ
　入力：光センサー
　処理：目標物が赤玉であるか判別し、アームで打つ
　出力：アームの回転速度

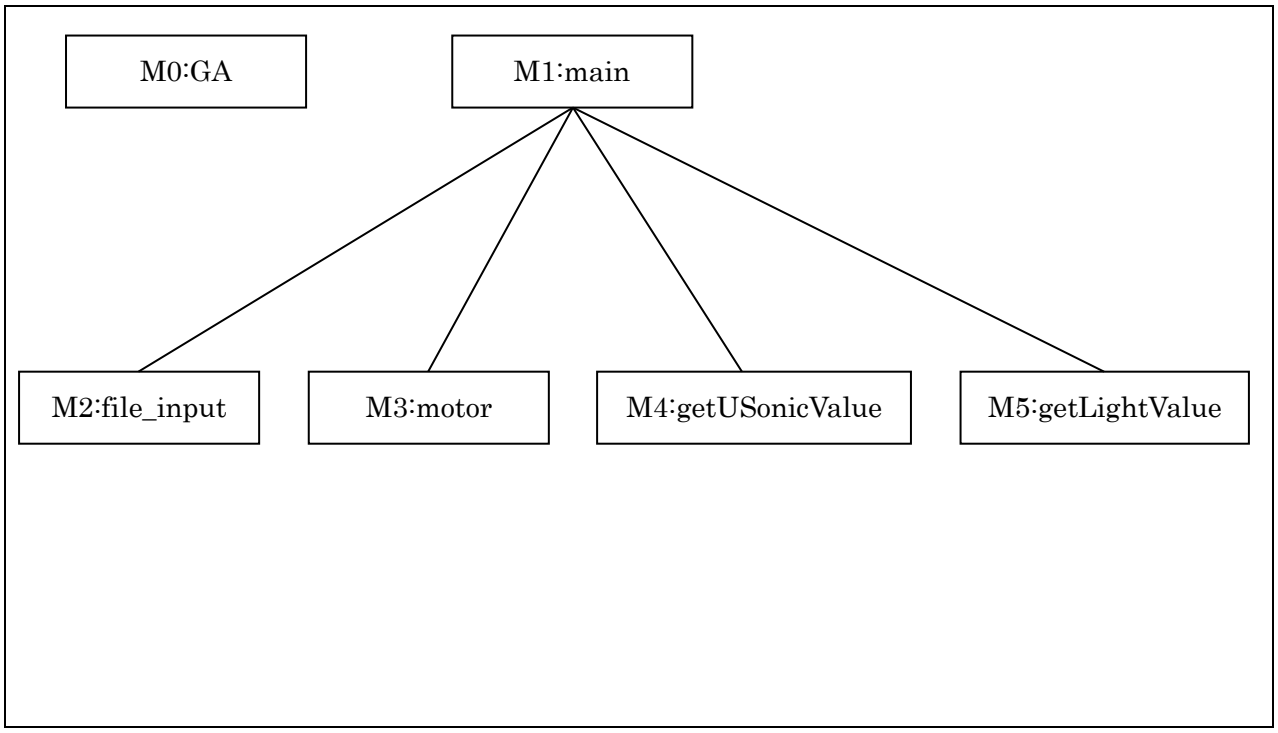
2. プログラムの入出力データ

入力データ	名称	型	説明
	超音波センサー	int	壁までの距離を測定する
	光センサー	int	目標物の色を判別する
	GA による 最適速度データ	int	距離に応じた最適速度を決定する
出力データ	**名称**	**型**	**説明**
	右モーター	int	モーターの回転速度（移動）
	左モーター	int	モーターの回転速度（移動）
	アーム	int	アームの回転速度（ボールを打つ）

8. 内部設計書

1. モジュール構造

(1) モジュール構造図

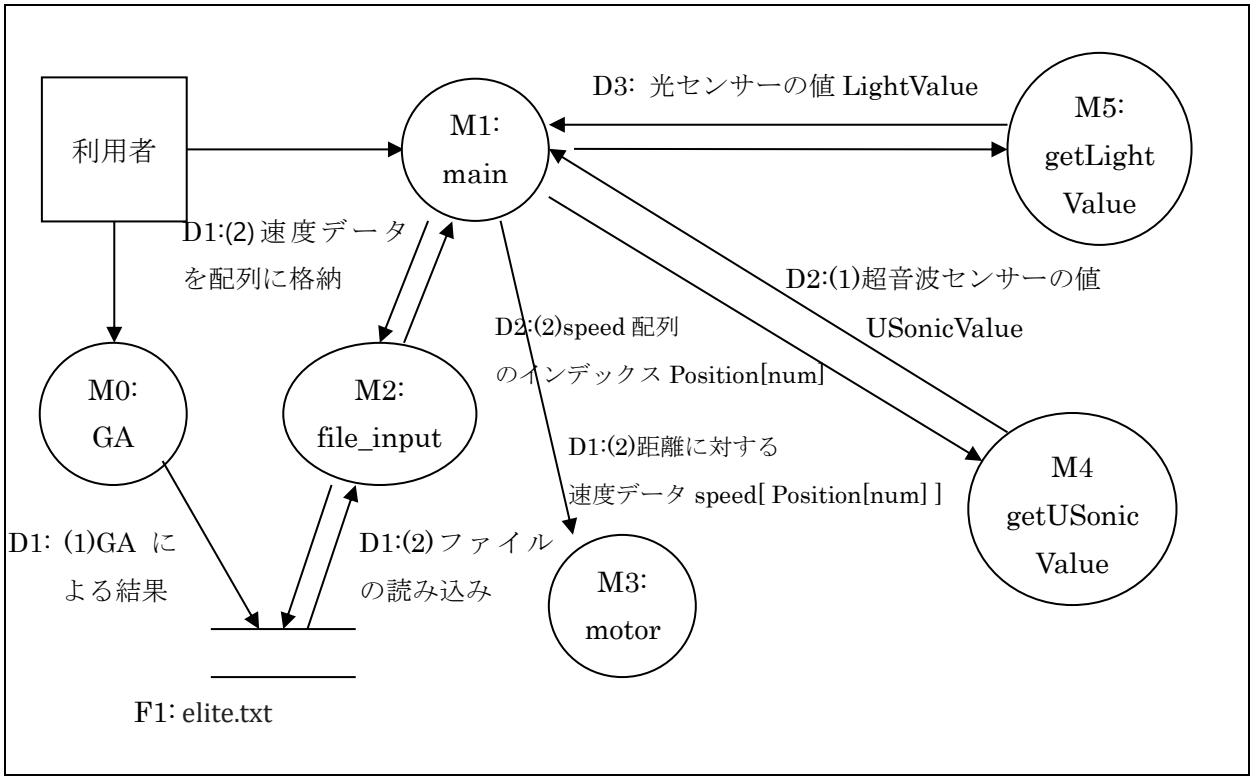


(2) モジュールの詳細

ID	モジュール名	概要
M0	GA	遺伝的アルゴリズムによって距離に応じた最適速度を決定し、ファイルに出力する
M1	main	各関数を実行する
M2	file_input	GA で出力した elite.txt 内の速度データを読み込む
M3	motor	ロボットの各モータ(車輪、アーム)の回転・速度を制御する
M4	getUSonicValue	超音波センサーの値を取得し、壁までの距離を測定する
M5	getLightValue	光センサーの値を取得し、対象物の色を識別・判定する

2. データフロー

(1) データフロー図



(2) データフローとファイルの詳細

ID		名称	型	説明
D1		最適速度		GA による距離に応じた最適速度データ
	要素	(1) elite[30]	int	最適速度データを格納する最終エリート個体用配列
		(2) speed[30]	int	ファイルから読み込む際に格納する配列 この配列で、スピードを制御する
D2		超音波センサー		超音波センサーによる距離測定データ
	要素	(1) USonicValue	int	getUSonicValue によって得られた距離数値を格納するための変数
		(2) Position[30]	int	USonicValue を配列 speed に対応するインデックスに変換した値
D3		光センサー		光センサーによって得られた色の数値データ
	要素	LightValue	int	getLightValue によって得られた色の数値データを格納するための変数

F1		elite.txt		GA による最適速度データを出力したテキストファイル
	要素	fp	FILE	ファイルを書き込む・読み込むためのファイルポインタ

10. ソースコード

次ページ以降に、「遺伝的アルゴリズム(GA)のプログラム」、「LEGO ロボット用制御プログラム」の順に、ソースコードを添付する。

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>      // msleepを使うために必要

//パラメータ宣言
#define IND_NUM     51              // 個体数（個体番号）
#define gene_length 30              // 遺伝子長（遺伝子番号）
#define EVOL_NUM      (IND_NUM - 1)          // エリートを除いた残りの進化個体数（トーナメント選択・交叉を行う個体数）（偶数）

#define generation  10000           // 世代数
#define CROSS_RATE      0.5       // 交叉率
#define MUT_RATE    0.01            // 突然変異率

#define First_Speed 20              // 初期速度固定
#define Last_Speed  0               // 最終速度固定


int main(void)
{
    srand((unsigned)time(NULL));

    int individual[IND_NUM][gene_length];     // 各個体の遺伝子情報（遺伝子[個体番号][遺伝子番号]）
    int fitness[IND_NUM] = {};                // 各個体の適合度
    int all_fitness[generation] = {};         // 全世代における各エリート個体の適合度
    int speed_sum[IND_NUM] = {};              // 遺伝子の値の合計
    int speed_change[IND_NUM] = {};           // 速度変化の2乗の合計

    int elite[gene_length] = {};            // エリート個体用配列
    int elite_number = 0;                   // エリート個体の遺伝子番号（0に初期化）
    int MAX_fitness = 0;                    // 個体の適応度の最大値
    int MAX_IND_NUM = 0;                    // 0番目の個体の遺伝子がエリートであると仮定

    int tournament[EVOL_NUM][gene_length];    // individual[][]から、トーナメント方式で選択した個体
    int cross[EVOL_NUM][gene_length];         // トーナメント選択した個体間で交叉させた新たな個体
    int temp = 0;                             // 交叉用データ一時保管変数
    int mutation[EVOL_NUM][gene_length];      // 交叉した個体を突然変異させた個体

    int rnd1 = 0;     // 乱数格納変数
    int rnd2 = 0;

    int i, j, g;      // for文カウンタ


    /********** 1. 初期個体群生成 **********/
    for (i = 0; i < IND_NUM; i++) {
        for (j = 0; j < gene_length; j++) {
            if (j == 0) {
                individual[i][j] = First_Speed;
            }
            else if (j == gene_length - 1) {
                individual[i][j] = Last_Speed;
            }
            else {
                individual[i][j] = rand() % 100 + 1;      // 1 ～ 100までの乱数
            }
        }
    }

```

```
61
62      /*********************************************************/
63      /*************** 遺伝的アルゴリズム開始 ***************/
64      /*********************************************************/
65      for (g = 0; g < generation; g++) {      // generation回進化する
66
67          printf("generation : %d世代¥r", g + 1);            // 世代の表示
68
69          /*************** 2. 評価 ***************/
70          // 初期化
71          for (i = 0; i < IND_NUM; i++) {
72              speed_sum[i] = 0;
73              speed_change[i] = 0;
74          }
75
76          // 遺伝子の値の合計
77          for (i = 0; i < IND_NUM; i++) {
78              for (j = 0; j < gene_length; j++) {
79                  speed_sum[i] += individual[i][j];
80              }
81          }
82
83          // 速度変化の二乗の合計
84          for (i = 0; i < IND_NUM; i++) {
85              for (j = 0; j < gene_length - 1; j++) {
86                  speed_change[i] += (int)pow((individual[i][j + 1] - individual[i][j]), 2.0);
87              }
88          }
89
90          // 適合度の計算
91          for (i = 0; i < IND_NUM; i++) {
92              fitness[i] = speed_sum[i] - speed_change[i];
93          }
94
95
96          /*************** 3.選択 ***************/
97          /***** 3-1. エリート選択 *****/
98          // 初期化
99          MAX_fitness = fitness[0];      // 0番目の個体の適応度を最大値に設定
100         MAX_IND_NUM = 0;                  // 0番目の個体の遺伝子をエリートに設定
101
102         // 適応度の最大値を求める
103         for (i = 1; i < IND_NUM; i++) {                 // 0番目をエリートに初期設定しているので、i = 1からスタート
104             if (MAX_fitness < fitness[i]) {
105                 MAX_fitness = fitness[i];
106                 MAX_IND_NUM = i;
107             }
108         }
109
110         // エリート個体の遺伝子情報・適応度を保存
111         for (j = 0; j < gene_length; j++) {
112             elite[j] = individual[MAX_IND_NUM][j];      // 遺伝子情報保存
113         }
114         all_fitness[g] = MAX_fitness;            // 適合度保存
115
116
117         /***** 3-2. トーナメント選択 *****/
118         // エリートを含む計 IND_NUM個(51個)の個体individual[][]から、TEMP_NUM個（IND_NUM - 1個）(50個)をトーナメント方式で選択する
119         for (i = 0; i < EVOL_NUM; i++) {      // EVOL_NUM 回トーナメントを行う
120             rnd1 = rand() % IND_NUM;      // ランダムに2ペア選ぶ(indivisualの個体番号)（エリートを含む）
121             rnd2 = rand() % IND_NUM;
122
123             if (fitness[rnd1] >= fitness[rnd2]) {      // rnd1の個体番号の適応度が大きい、もしくは適応度が等しいまたは rnd1 = rnd2の場合
124                 for (j = 0; j < gene_length; j++) {
125                     tournament[i][j] = individual[rnd1][j];
126                 }
127             }
128             else if (fitness[rnd1] < fitness[rnd2]) {
129                 for (j = 0; j < gene_length; j++) {
130                     tournament[i][j] = individual[rnd2][j];
131                 }
132             }
133         }
134
```

```
135
136         /*************** 4. 交叉 ***************/
137         for (i = 0; i < EVOL_NUM; i += 2) {      // 交叉対象のEVOL_NUM 個（偶数個）の個体間で、EVOL_NUM / 2 回交叉させる
138             rnd1 = rand() % EVOL_NUM;
139             rnd2 = rand() % EVOL_NUM;            // ランダムに交叉させる2組を決定
140
141             for (j = 0; j < gene_length; j++) {
142                 cross[i][j] = tournament[rnd1][j];          // 交叉後の配列に交叉前の遺伝子をコピー
143                 cross[i + 1][j] = tournament[rnd2][j];
144
145                 if ( (double)rand() / RAND_MAX < CROSS_RATE ) {              // 0.0～1.0の乱数が交叉率より大きいか
146                     temp = cross[i][j];                     // 交叉させる
147                     cross[i][j] = cross[i + 1][j];
148                     cross[i + 1][j] = temp;
149                 }
150             }
151
152         }
153
154
155         /*************** 5. 突然変異 ***************/
156         for (i = 0; i < EVOL_NUM; i++) {              // 全ての交叉済みの個体に対して突然変異を行う
157             for (j = 1; j < gene_length - 1; j++) {                  // 初期速度と最終速度は変化させない
158                 if ( (double)rand() / RAND_MAX < MUT_RATE ) {
159                     cross[i][j] = rand() % 100 + 1;          // 突然変異させる
160                 }
161             }
162         }
163         for (i = 0; i < EVOL_NUM; i++) {
164             for (j = 0; j < gene_length; j++) {
165                 mutation[i][j] = cross[i][j];     // 突然変異後のcrossを突然変異済み配列に格納
166             }
167         }
168
169
170         /*************** 6. 最終的な子世代の作成 ***************/
171         for (i = 0; i < IND_NUM; i++) {
172             if (i == 0) {
173                 for (j = 0; j < gene_length; j++) {
174                     individual[i][j] = elite[j];          // individual の先頭にエリートを格納
175                 }
176             }
177             else {
178                 for (j = 0; j < gene_length; j++) {
179                     individual[i][j] = mutation[i - 1][j];     // 残りは進化後の個体に更新
180                 }
181             }
182
183         }
184
185     }
186     /*********************************************************/
187     /*************** 遺伝的アルゴリズム終了 ***************/
188     /*********************************************************/
189
```

```
190
191     // 最終的なエリートの遺伝子情報を表示
192     printf("¥n¥n○last elite:¥n");
193     for (j = 0; j < gene_length; j++) {
194         printf("%d¥t", elite[j]);      // elite[j] = individual[0][j]
195     }
196
197     // 最終的なエリートの適応度を表示
198     printf("¥n○last MAX_fitness:¥t%d¥n¥n", MAX_fitness);
199
200     // 遺伝的アルゴリズムによる最適速度をelite.txtに書き込み
201     FILE *f_elite;
202     fopen_s(&f_elite, "elite.txt", "w");
203     for (j = 0; j < gene_length; j++) {
204         fprintf(f_elite, "%d¥n", elite[j]);
205     }
206     fclose(f_elite);
207
208     // 遺伝的アルゴリズムによる各世代の適応度をfitness.txtに書き込み
209     FILE *f_fitness;
210     fopen_s(&f_fitness, "fitness.txt", "w");
211     for (g = 0; g < generation; g++) {
212         fprintf(f_fitness, "%d¥n", all_fitness[g]);
213     }
214     fclose(f_fitness);
215
216     return 0;
217 }
218
```

ソースコード 1. 遺伝的アルゴリズム(GA)による LEGO ロボットの距離に対する最適速度を求めるソースコード

```
#include "stdafx.h"
#include "fantom/iNXT.h"
#include "fantom/iNXTIterator.h"
#include "fantom/tStatus.h"
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>

#include <time.h>            // msleepを使うために必要
#include "msleep.h"          // n秒間動作を停止する関数を使うのに用いる
#include "choonnpa.h"        // 超音波センサーを用いるのに使う
#include "hikari.h"          // 光センサーを用いるのに使う

#define distance      240    // 壁（ボール）の手前までの超音波センサーの距離(255以下で均等に30分割するための最適距離)
#define gene_length 30       // 読み込む遺伝子長
#define Shot_Power  40       // 赤玉を打つパワー

#define MOTOR_A 0      // アームの番号
#define MOTOR_R 1      // 右モータの番号
#define MOTOR_L 2      // 左モータの番号


//---------------------------------------- プロトタイプ宣言 ----------------------------------------
void motor(nFANTOM100::iNXT* nxtPtr, nFANTOM100::tStatus status, int mot_num, char motor_speed);
void file_input(char *filename, int speed[gene_length]);
//--------------------------------------------------------------------------------------------------

int _tmain(int argc, _TCHAR* argv[]) {
    int speed[gene_length] = {};              // GAによる最適速度をファイルから読み込み格納する配列
    int Position[gene_length] = {};           // 0〜255までの距離(位置情報)を格納するための配列

    /***** 1. speed配列の速度に対応する超音波センサの値（位置情報）の決定 *****/
    //------------------------------ 位置情報の計算 (17 〜 249 まで) ------------------------------
    for (int i = 0; i < gene_length; i++) {
        // 配列の先頭から降順に、位置情報を格納する。
        // LightValue = 24 の時に停止 (speed[29] = 0)するように、
        // Position[29]に24が格納されるように計算する。
        // Position配列は、「8」ずつ値が変化していくように設定。

        Position[i] = (distance - distance * i / gene_length) + 9;

    }
    //--------------------------------------------------------------------------------------------
    /***** 2. エリート（最適速度）ファイルの読み込み*****/
    file_input("elite.txt", speed);

    // 各超音波センサーの値に対する、各speed配列内の最適速度の出力
    for (int i = 0; i < gene_length; i++) {
        printf("num: %d¥t¥tUSonicValue >= %d¥tspeed: %d¥n", i, Position[i], speed[i]);
    }

```

```
55
56      //---------------------------------- 前処理 ----------------------------------
57      // ライブラリ参照用変数
58      nFANTOM100::iNXTIterator* nxtIteratorPtr = NULL;
59      nFANTOM100::tStatus status;
60      nFANTOM100::iNXT* nxtPtr = NULL;
61
62      char exitcode = 0;
63
64      // NXTへの接続．参照用ポインタの取得（USB or Bluetooth）
65      nxtIteratorPtr = nFANTOM100::iNXT::createNXTIterator(
66          true,
67          5,
68          status
69      );
70
71      // 接続に失敗した場合，終了
72      if (status.isFatal()) {
73          printf("接続に失敗しました。¥n¥n");
74          return 0;
75      }
76
77      // 接続成功時の出力
78      printf("Communication start with NXT...¥n");
79
80      // ライブラリ用ポインタの取得
81      nxtPtr = nxtIteratorPtr->getNXT(status);
82
83      // 参照用ポインタの破棄
84      nFANTOM100::iNXT::destroyNXTIterator(nxtIteratorPtr);
85
86      //各センサーの初期化
87      setUSonicValue(nxtPtr, status, 3);          // 超音波センサ初期化   ポート番号4は3
88      setLightValue(nxtPtr, status, 2);          // 光センサー初期化        ポート番号3は2
89  // ------------------------------------------------------------------------------
90
```

```
91
92  /*************************************************************/
93  /*************** 3. ロボットの自動制御開始 ***************/
94  /*************************************************************/
95
96      int num = 0;      // 超音波センサーの数値（距離）に対応する、speed配列のインデックス
97
98      while (exitcode == 0) {                  // exitcode = 1となるまで（赤玉を打つまで）無限ループ
99
100         int LightValue = getLightValue(nxtPtr, status, 2);            // 光センサーの値
101         int USonicValue = getUSonicValue(nxtPtr, status, 3);      // 超音波センサーの値
102
103         /***** step1. 壁までGAでの最適速度で近づく *****/
104         // 実際の距離を超音波センサーの数値に対応させ、壁まで走行。
105         if (USonicValue) {
106             for (int i = 0; i < gene_length; i++) {                  // 超音波センサーの値を用いて現在の位置を把握
107                 if (USonicValue >= Position[i]) {                  // 超音波センサーの値に対応したスピードにする
108                     num = i;
109                     break;            // 距離に応じた最適速度にした後、for文から抜ける。
110                 }
111             }
112         }
113         motor(nxtPtr, status, MOTOR_R, speed[num]);            // 右車輪
114         motor(nxtPtr, status, MOTOR_L, speed[num]);            // 左車輪
115
116         // 超音波センサーの数値、超音波センサーの値に対応するPositionの値、GAの速度、光センサーの値
117         printf("USonicValue=%5d¥tPosition=%5d¥tspeed=%5d¥tLightValue=%5d¥r", USonicValue, Position[num], speed[num], LightValue);
118
119         /***** step2. 壁の間際(USonicValue = 24)まで移動した場合(speed配列の末尾まで速度データを読み込んだ場合) *****/
120         if (num == gene_length - 1) {
121             /***** step2. 目の前の対象物体の色を判定 *****/
122             Msleep(1000);
123             LightValue = getLightValue(nxtPtr, status, 2);
124             Msleep(1000);
125
126             /***---------- step2-1. 光センサーの値が170以上、すなわち、赤玉以外の場合 ----------***/
127             if (LightValue > 170) {
128                 Msleep(1000);
129
130                 /*****  step3. 対象物の右側部分を探索 *****/
131                 // 3-1. 右側を向く
132                 motor(nxtPtr, status, MOTOR_R, -speed[27]);
133                 Msleep(1000);
134
135                 // 3-2. 少しだけ前進
136                 motor(nxtPtr, status, MOTOR_R, speed[27]);
137                 motor(nxtPtr, status, MOTOR_L, speed[27]);
138                 Msleep(800);
139
140                 // 3-3. 停止
141                 motor(nxtPtr, status, MOTOR_R, 0);
142                 motor(nxtPtr, status, MOTOR_L, 0);
143                 Msleep(1000);
144
145                 /***** step4. 目の前(右側前方)の対称物体の色を判定 *****/
146                 Msleep(1000);
147                 LightValue = getLightValue(nxtPtr, status, 2);
148                 Msleep(1000);
149
150                 /*****  step4-A. 対称物体が赤玉である場合 *****/
151                 // 対称物体との距離が近いため、LightValueの赤玉条件を165としている。
152                 if (LightValue < 165) {
153                     // A-1. アームを時計回りにShot_Powerだけ回す
154                     motor(nxtPtr, status, MOTOR_A, Shot_Power);
155                     Msleep(180);
156
157                     // A-2. アームを停止
158                     motor(nxtPtr, status, MOTOR_A, 0);
159                     Msleep(20);
160
161                     // A-3. アームを反時計回りにShot_Powerだけ回す
162                     motor(nxtPtr, status, MOTOR_A, -(Shot_Power));
163                     Msleep(200);
164
165                     // A-4. アームを停止
166                     motor(nxtPtr, status, MOTOR_A, 0);
167
168                     break;      // 無限ループから抜ける(プログラム終了)
169                     // exitcode = 1;とすると、step5を実行してしまう。（exitcodeを評価するのは、先頭のwhile文）
170                 }
```

```
171
172                 /***** step4-B. 対象物が赤玉以外の場合 *****/
173                 // step3.の、対象物の右側探索直前の状態（向き）に戻す
174                 else {
175                     // B-1. 少しだけ後退
176                     motor(nxtPtr, status, MOTOR_R, -speed[27]);
177                     motor(nxtPtr, status, MOTOR_L, -speed[27]);
178                     Msleep(1200);
179
180                     // B-2. 左側を向く
181                     motor(nxtPtr, status, MOTOR_R, speed[27]-8);
182                     Msleep(800);
183
184                     // B-3. 停止
185                     motor(nxtPtr, status, MOTOR_R, 0);
186                     motor(nxtPtr, status, MOTOR_L, 0);
187                     Msleep(1000);
188                 }
189
190                 /***** step5. 未探索な右側領域を探索できるように後退する *****/
191                 // 5-1. 右斜め後ろに後退
192                 motor(nxtPtr, status, MOTOR_R, -speed[24]);
193                 motor(nxtPtr, status, MOTOR_L, -speed[24] - 10);
194                 Msleep(2000);
195
196                 // 5-2. 左斜め後ろに後退し、斜体の向きをまっすぐに調整
197                 motor(nxtPtr, status, MOTOR_R, -speed[24] - 10);
198                 motor(nxtPtr, status, MOTOR_L, -speed[24]);
199                 Msleep(2000);
200
201                 // 5-3. 一度止まる
202                 motor(nxtPtr, status, MOTOR_R, 0);
203                 motor(nxtPtr, status, MOTOR_L, 0);
204                 Msleep(1000);
205
206             }
207
208             /***---------- step2-2. 対象物が赤玉の場合 ----------***/
209             else {
210                 // 2-2-1. アームを時計回りにShot_Powerだけ回す
211                 motor(nxtPtr, status, MOTOR_A, Shot_Power);
212                 Msleep(180);
213
214                 // 2-2-2. アームを停止
215                 motor(nxtPtr, status, MOTOR_A, 0);
216                 Msleep(20);
217
218                 // 2-2-3. アームを反時計回りにShot_Powerだけ回す
219                 motor(nxtPtr, status, MOTOR_A, -(Shot_Power) );
220                 Msleep(200);
221
222                 // 2-2-4. アームを停止
223                 motor(nxtPtr, status, MOTOR_A, 0);
224
225                 exitcode = 1;      // 無限ループ終了(プログラム終了)
226             }
227
228             num = 0;      // speed配列のインデックスの初期化（壁から離れた状態にする）
229         }     // 最初の光センサーの値によるif文終了
230
231     }     // 無限ループ(while文)終了
232
233     /*********************************************************/
234     /************** 3. ロボットの自動制御終了 **************/
235     /*********************************************************/
236
```

```
237
238
239     //------------------------------ 後処理 ------------------------------
240     // 参照用ポインタの破棄
241     nFANTOM100::iNXT::destroyNXT(nxtPtr);
242     return 0;
243     // --------------------------------------------------------------------
244
245 }
246
247
248 /***** ファイル読み込み関数 *****/
249 void file_input(char *filename, int speed[gene_length]) {
250     FILE *fp;
251     int i = 0;
252     int temp = 0;
253
254     if ((fopen_s(&fp, filename, "r")) != 0) {
255         printf("ファイルがありません:%s¥n", filename);
256         exit(1);
257     }
258
259     // speed配列にeliteファイルのデータを格納
260     for (i = 0; i < gene_length; i++) {
261         fscanf_s(fp, "%d", &speed[i]);
262     }
263 }
264 /***** ここまで*****/
265
266
267 /***** モーター制御関数（アーム、右車輪、左車輪）*****/
268 void motor(nFANTOM100::iNXT* nxtPtr, nFANTOM100::tStatus status, int mot_num, char motor_speed) {
269     ViUInt8 directCommandBuffer[] = {
270
271     4, // コマンド．「SETOUTPUTSTATE《を意味する4が入る．
272
273     0, // 出力ポート番号．0：「A《ポート，1：「B《ポート，2：「C《ポート．
274
275     0, // モーターのパワー．値の範囲：-100〜+100．0：モーター停止．
276
277     7, // モーターのモード．1：モーターオン，2：ブレーキオン，4：制御オン．
278
279     1, // 制御モード．0：制御なし，1：スピードの制御，2：複数モーターの同期
280
281     0, // 回転レート．値の範囲：-100〜+100.
282
283     0x20, // 動作状態．0x10：徐々に加速，0x20：通常動作，0x30：徐々に減速.
284
285     0,0,0,0 // 制御する回転量．0：回転継続.
286
287     };
288
289
290     // モーターの左右・速度指定（アーム: mot_num = 0, 右車輪：mot_num = 1, 左車輪：mot_num = 2モーター）
291     directCommandBuffer[1] = mot_num;           // 動かす部位の指定
292     directCommandBuffer[2] = motor_speed;     // 速度の指定
293
294
295     // ダイレクトコマンドの送信
296
297     nxtPtr->sendDirectCommand(
298
299         false,
300
301         reinterpret_cast<ViByte*>(directCommandBuffer),
302
303         sizeof(directCommandBuffer),
304
305         NULL,
306
307         0,
308
309         status
310
311     );
312
313 }
314 /***** ここまで ****/
315
```

ソースコード 2. GA による最適速度で LEGO ロボットを制御するソースコード

11.実験と結果

11.1 第 6 週までの実験の実施内容

以下の手順で実験を行った。

①LEGO マインドストームの説明書に基づいて、LEGO ロボットを組み立てた。

②Bluetooth による通信設定、及びサンプルプログラムの動作確認を行った。

③使用するソフトコンピューティング手法の決定、及び、実現する機能を分割し、班員内で役割分担を行った。
なお、使用するソフトコンピューティング手法、及び、実現する機能とその役割分担状況を以下に示す。

○使用するソフトコンピューティング手法：GA

○実現する機能
(0)遺伝的アルゴリズムによる最適速度制御
(1)モータ制御
(2)超音波センサーによる距離・向き識別
(3)色判別
(4)アームの制御

○班の役割分担状況（上記の(0)~(4)で役割分担）
(0)：鯨田
(1)(2)：荒田
(3)(4)：尾崎

④要求仕様書、外部設計書、内部設計書を作成し、それに基づき、「遺伝的アルゴリズム(GA)のプログラム」、「LEGO ロボット用制御プログラム」のソースコードを作成した。

⑤作成したプログラムを LEGO ロボットに対して実行し、本実験における課題（要求仕様）である「ボールを探索し、青ボールの場合は打たず、赤ボールの場合のみ打つ」動画を撮影した。

⑥我々の班で作成したLEGOロボットの知的制御システムの発表スライドを作成し、成果発表を行った。

11.2 作成した GA の設計、アルゴリズム及び結果

11.2.1 作成した GA における各種設定パラメータ

ソースコード 1 に示す、遺伝的アルゴリズムにより、ロボットの距離に対する最適速度を求めるプログラムを作成した。その際、遺伝子長、世代数等の設定したパラメータを以下に示す。

・世代数：10000
・初期速度：20
・最終速度：0
・遺伝子長：30
・個体数：51
・選択方法方法：エリート選択及びトーナメント選択
・トーナメントサイズ：2
・交叉方法：一様交叉
・交叉率：0.5
・突然変異確率：0.01

11.2.2 作成した GA における個体表現

距離（超音波センサーの値）に対するスピードを遺伝的アルゴリズムで獲得する際の遺伝子型を表現した図を図 6 に示す。

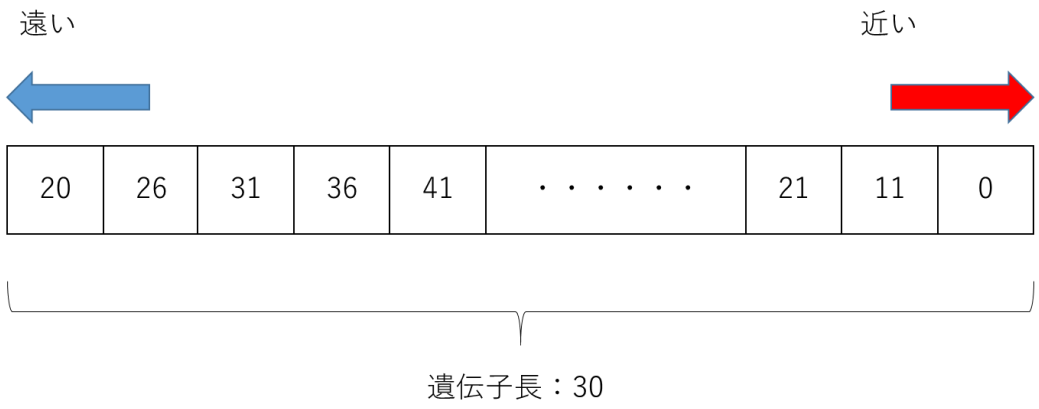


図 6. GA における速度データの遺伝子型（個体表現）

各遺伝子座の数値は、距離に対する速度に一致しており、長さ 30 の配列で表現することで、距離を 30 分割した場合の最適速度を求めた。また、配列の先頭から末尾にかけて、距離が遠い際の速度から近い際の速度という設計にした。なお、初期速度・最終速度は固定値で、それぞれ 20, 0 と設定した。

11.2.3 作成した GA の処理手順

作成した最適速度を求める GA の処理手順のフローチャートを図 7 に示す。

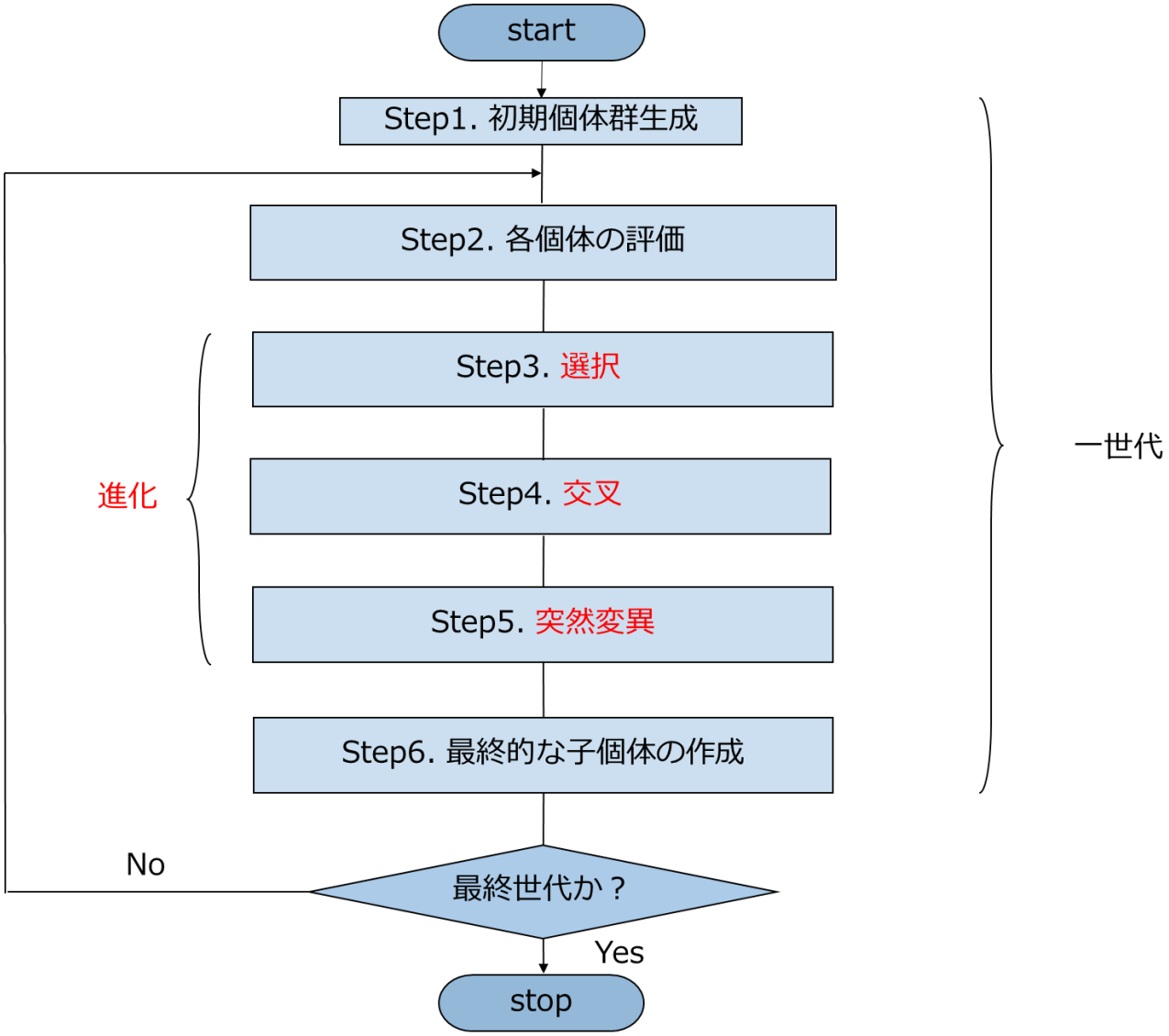


図 7. 作成した最適速度を求める GA の処理手順のフローチャート

次に、各処理手順を以下に説明する。

Step1. 初期個体群の生成

図 6 に示した遺伝子型の個体を 51 個生成する。初期個体群は、ランダムに生成された数字列により構成しており、1 ~ 100 までの乱数を 1 ~ 28 番目の遺伝子座に割り当てている。なお、0 番目と 29 番目の遺伝子座には、初期速度と最終速度である 20 と 0（固定値）を割り当てている。

Step2. 各個体の評価

本実験において、求める最適速度データは、「できるだけ速く」と「滑らかな速度変化」という二つの目的を同時に満たすような速度データである。つまり、本問題は「多目的最適化問題」であるといえる。この多目的最適化問題を満たす個体の評価方法として、以下に示す適合度関数を用いることで、適合度を数値化し、評価した。

$$\text{fitness} = \text{speed\_sum} - (\text{speed\_change})^2 \quad \text{—②}$$

fitness ：適合度

speed_sum：遺伝子の値の合計（スピードデータの速度合計値）

$-(\text{speed\_change})^2$：隣り合う遺伝子座の差の二乗（速度変化の二乗の合計値）

Step3. 選択

ここでは、「エリート選択」及び「トーナメント選択」を行っている。

Step3-1. エリート選択

次世代に残す個体として、「その世代において最も適合度の高い個体」、すなわち、「エリート個体」を 1 つ求める。

Step3-2. トーナメント選択

エリートを含む計 51 個の個体から、計 50 個をトーナメントサイズ 2 のトーナメント方式で選択する。また、トーナメントで比較する個体の選び方はランダムで、計 50 回トーナメントを行い、ランダムで選択した 2 つの個体の内、適合度の高い方の遺伝子情報を保存している。

Step4. 交叉

ここでは、「一様交叉」によって交叉している。

具体的には、Step3 で選択した個体の隣り合う二組ずつを親として、交叉率 0.5（平

均的には、二組の親個体の遺伝子の半分を入れ替える）で各親個体の遺伝子毎にランダムに交叉するかしないかを決定している。つまり、一度の交叉で 2 組の親個体から 2 つの新たな子個体が生成されるため、計 25 回（=50 個体分）の一様交叉を実行している。

Step5. 突然変異

全ての交叉済みの新たな 50 個の子個体に対して、突然変異を行っている。突然変異では、1～28 番目の遺伝子座に対して、突然変異確率 0.01（個々の遺伝子座に対して突然変異を行うかどうかを決める確率）で 1 ~ 100 までの乱数に変異させている。なお、突然変異確率を 0.01 とした理由は、通常の GA では、突然変異確率として記号列の長さの逆数が用いられることが多いからである。つまり、記号列の長さの逆数ということは、平均的には、1 個の遺伝子座に対して突然変異が適用されることになる。すなわち、遺伝子の 1%を乱数に変更すればよいということになるので、突然変異確率を 0.01 と設定した。

Step6. 最終的な子個体の作成

Step3 で選択したエリート個体 1 個と Step3 ~ 5 で進化させた子個体 50 を一つの新たな次世代の個体群（計 51 個）として結合させている。

以上の Step2～Step6 を 10000 回（世代）繰り返すことにより、最適解（最適速度データ）を求めた。

11.2.4 作成した GA の実行結果

・ソースコード 1（GA）を実行して得られた最適速度データを図 8 に示す。なお、横軸は目標物までの距離、縦軸は距離に対する最適速度である。また、この最適速度データ（最終世代のエリート個体）の適合度は 782 であった。図 8 より、設計した適合度関数を最大化するように、滑らかな速度変化で、かつ、できるだけ速い速度となるように、縦軸の頂点が最高速度 61 の傾き負の二次関数型のグラフの概形となっていることを確認でき、条件を満たす最適速度データを求めることに成功した。

・次に、ソースコード 1（GA）を実行した際の、1 世代目から 10000 世代までの各世代における GA の進化曲線（各世代における適合度の変化）を図 9 に示す。また、図 9 を拡大した曲線を図 10 に示す。なお、横軸は世代数、縦軸はその世代の適合度の最大値（エリート個体の適合度）である。図 9、及び図 10 より、1～1000 世代ほどで、爆発的に適

合度が上がり、適合度関数の式②を最大化するように進化していることを確認できた。
そして、約 3000 世代で適合度が 782 に収束していることを確認できた。

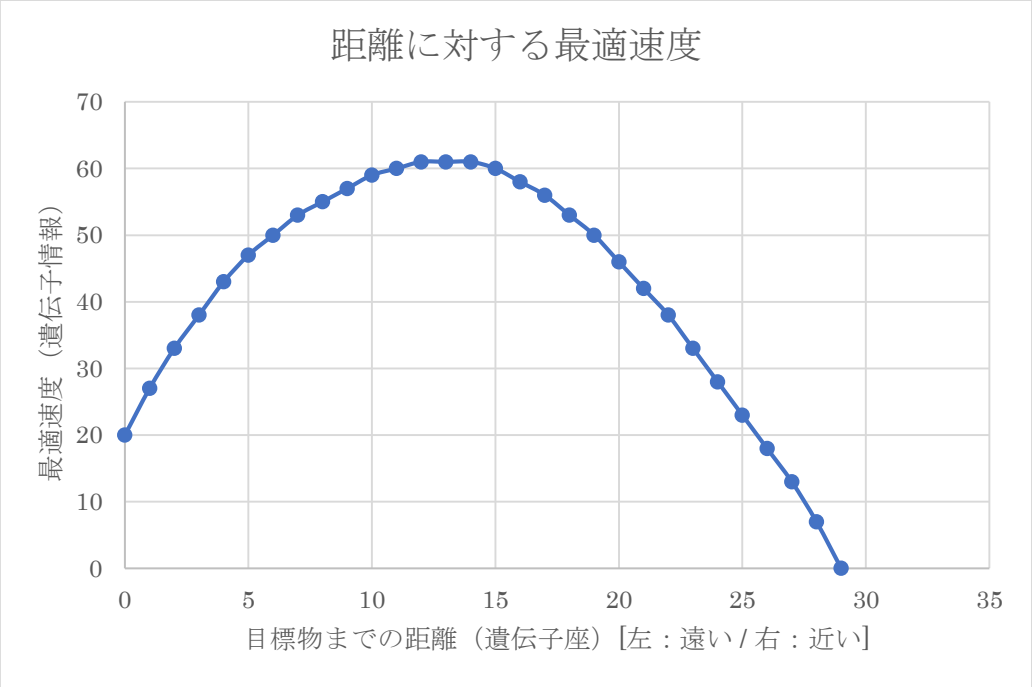


図 8. ソースコード 1（GA）を実行して得られた最適速度データ

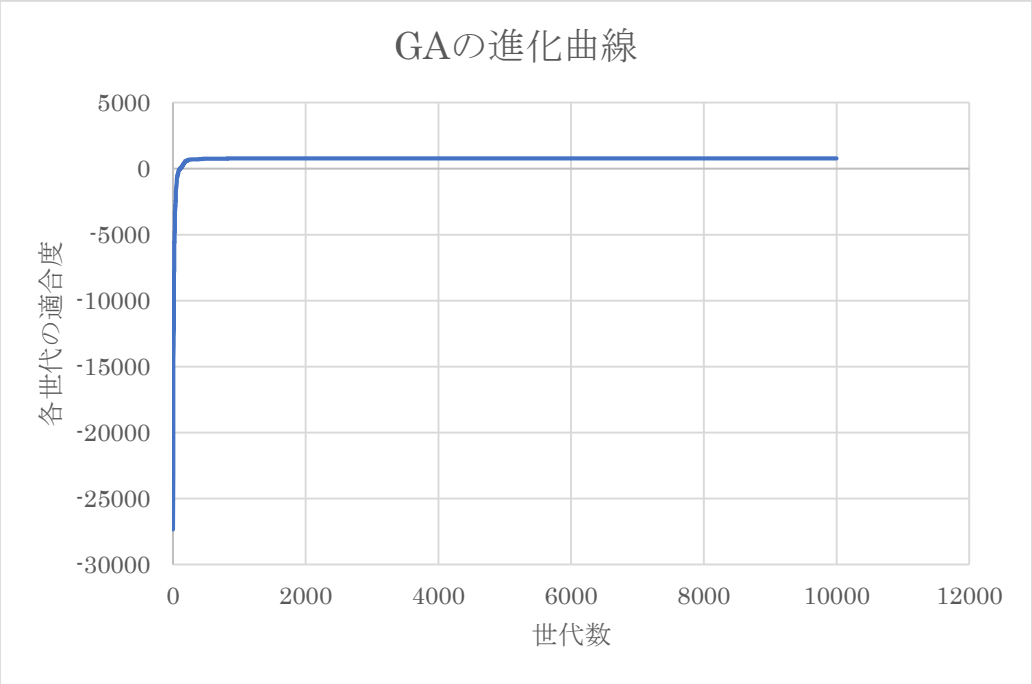


図 9. ソースコード 1 を実行した際の GA の進化曲線（各世代における適合度の変化）

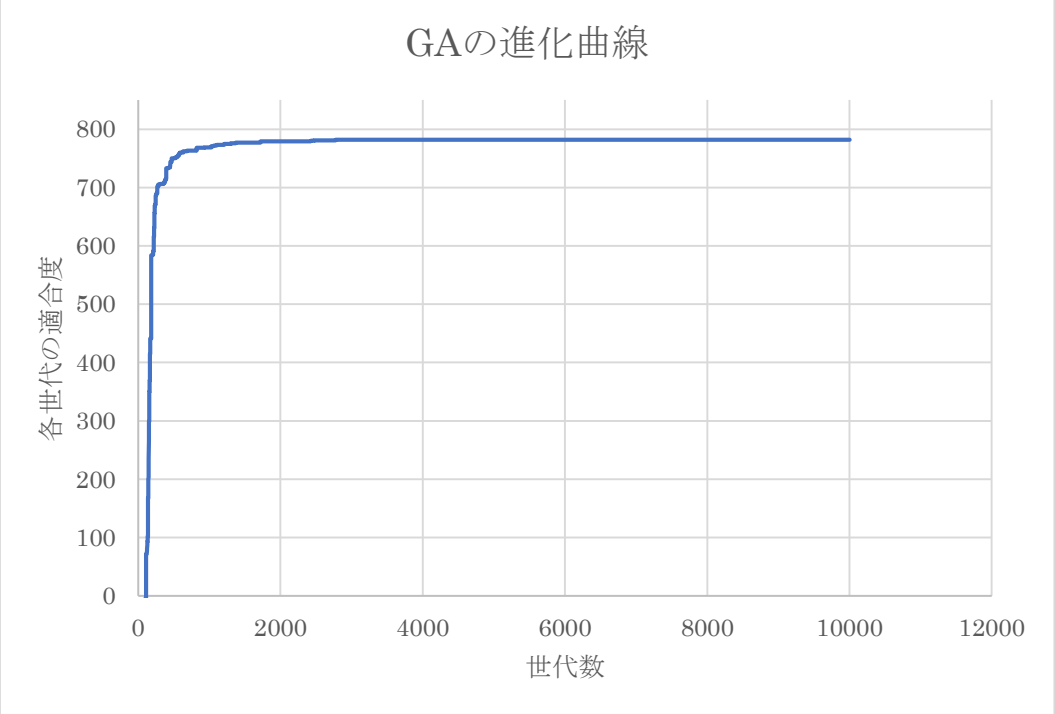


図 10. 図 9 を拡大した曲線（縦軸(適合度)の最小値を 0 としている）

11.3 作成した LEGO ロボットのプログラムの設計、アルゴリズム及び結果

11.3.1 作成した LEGO ロボット制御プログラムにおける各種設定パラメータ

ソースコード 2 に示す、LEGO ロボット用制御プログラムを作成した。その際、設定したパラメータを以下に示す。

・読み込むスピードデータの配列長（遺伝子長）：30
・赤玉を打つパワー：40
・後退する時の速度：最低速度データ配列の 25 番目の速度データ

11.3.2 作成した LEGO ロボット制御プログラムの処理手順

作成した LEGO ロボット制御プログラムの処理手順のフローチャートを図 11 に示す。

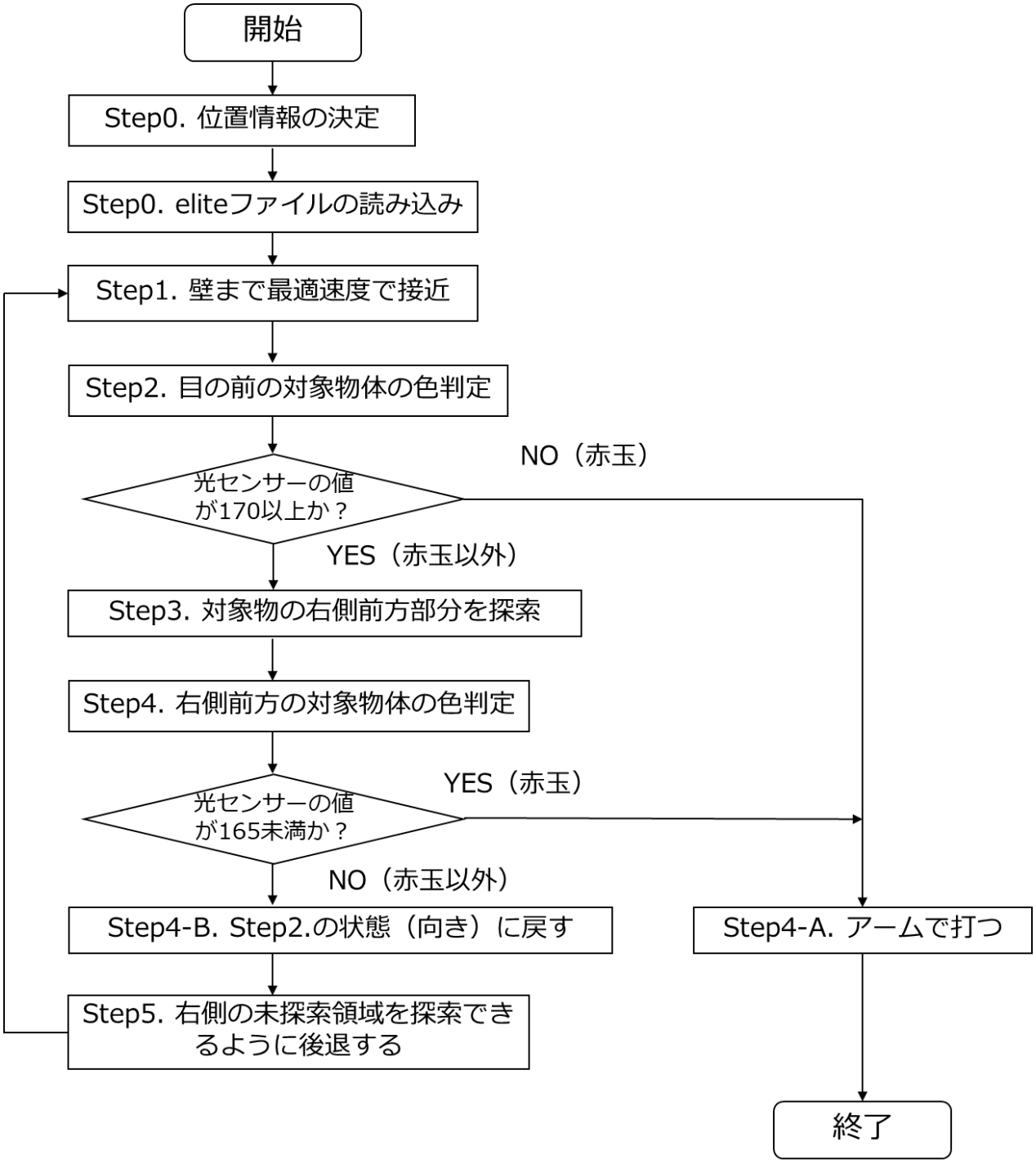


図 11. 作成した LEGO ロボット制御プログラムの処理手順のフローチャート

次に、各処理手順を以下に説明する。

Step0. 位置情報の決定及び、elite ファイルの読み込み

初めに、GA による最適速度データ（遺伝子長：30）に対応する超音波センサーの値（位置情報）を決定している。具体的に説明すると、読み込む最適速度データは、距離を 30 分割して考えた場合の速度データなので、超音波センサーの取り得る距離の数値 0 ~ 255 を 30 等分するように考えている。そして、30 分割した値（位置情報）を境に、超音波センサーの値が、その位置情報の値以上であれば、対応する最適速度データで走行するように考えた。しかし、255 は、整数値では 30 等分することができない。255 以内でかつ、整数値で 30 分割することのできる値の最大値は 240 である。よって、超音波センサーの取り得る値を 0 ~ 240 として 30 分割した。（つまり、超音波センサーの値が 240 / 30 = 8 変わる毎に、最適速度データを変更するように考えた。）

以下に、上記の位置情報の分割計算式及び、概略図を示す。

$$\text{Position} = \text{distance} - \frac{distance}{gene\_length} * i \qquad \text{—③}$$

Position：位置情報

distance：240（255 を整数値で 30 分割することができる最大値）

gene_length：読み込む最適速度データの個数[= 30]（遺伝子長）

i：分割点（0 ~ 29）

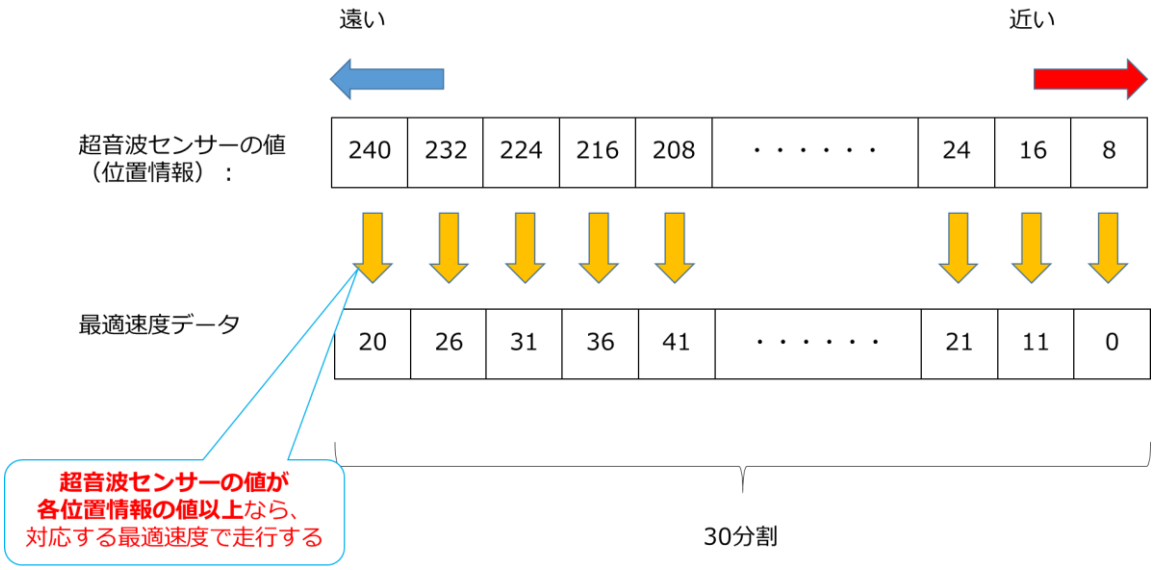


図 12. 超音波センサーの取り得る値を 30 分割した際の位置情報

次に、壁間際に近付いた時の、LEGO ロボットの超音波センサーの値を調べると、「24」であった。つまり、超音波センサーの値が「25」の時に、最適速度データの最終速度の一つ前の速度で走行し、超音波センサーの値が「24」となった瞬間に最終速度（＝0）となればよい（停止すればよい）。したがって、図 12 の位置情報の右端の一つ前の値「16」（＝240 を 30 分割した位置情報における壁間際前での値）を「25」（＝壁間際での超音波センサーの値）となるように全ての 30 等分した位置情報に「9」を足せばよい。すなわち、式③に 9 を足せばよい。

（9 を足すと、図 12 の右から二番目の距離情報は 32～25 となり、この区間では最適速度データの最終速度の一つ前の速度：11 で走行することになる。そして、超音波センサーの値が 24 となった瞬間にこの位置情報の区間から外れるため、最終速度の 0 となる。）

以下に、上記の壁間際での超音波センサーの値を考慮した位置情報の計算式及び、概略図を示す。

$$\text{Position} = \left(\text{distance} - \frac{distance}{gene\_length} * i\right) + 9 \qquad \text{—④}$$

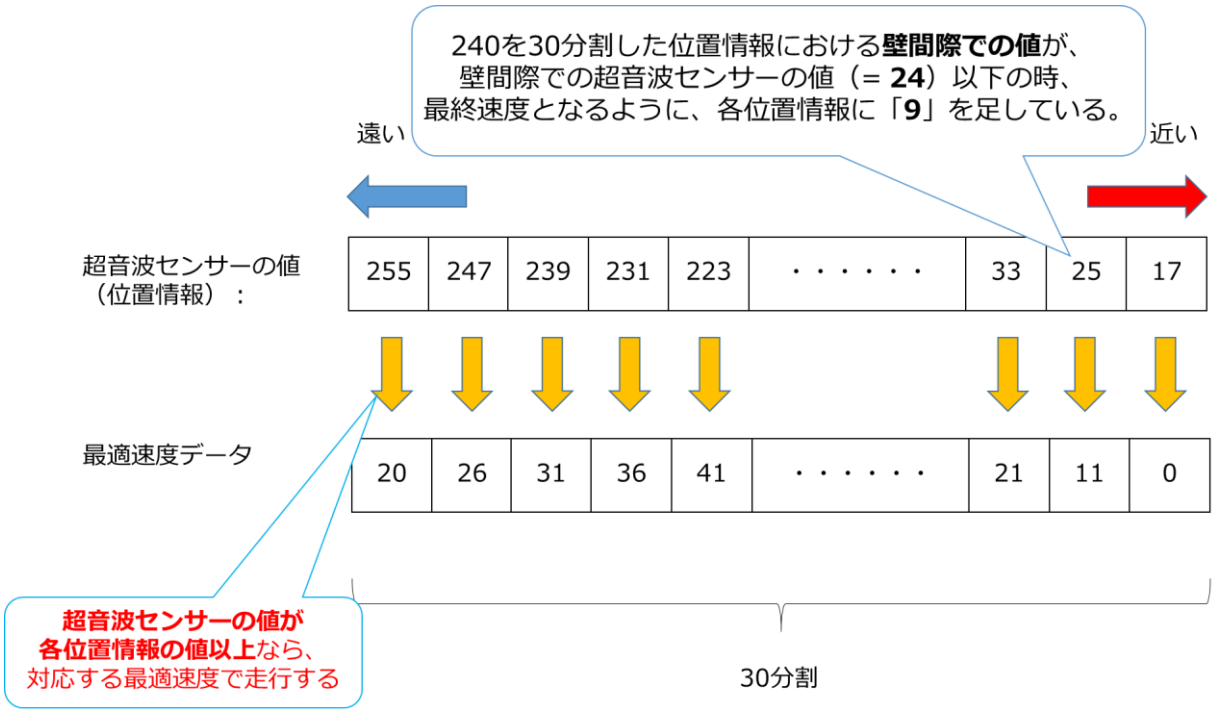


図 13.「30 等分した位置情報の壁間際での値（区間）」と「超音波センサーの壁間際での値」を一致するように調整した位置情報

以上の位置情報の決定により、超音波センサーの値がその位置情報以上であれば、「位置情報を格納している配列のインデックス」に対応する、最適速度を格納している配列の値（最適速度）で走行するように工夫した。

位置情報を決定した後は､GA により求めた最適速度データが格納されている elite ファイルを読み込んでいる。

Step1. 壁まで最適速度で接近

Step0. で決定した位置情報を基に、最適速度で壁まで（超音波センサーの値が 23 となるまで）走行している。なお、壁まで接近したかの判定は、読み込んだ最適速度データを格納している配列（以下、スピード配列と呼ぶ）のインデックスが配列末尾に一致しているかを調べている。

ここで、壁間際に到達したか判定する条件で「超音波センサーの値「24」」を使用せず、スピード配列の配列長を用いている理由は、LEGO ロボットの超音波センサーは、その機体によって取りうる値が若干異なるためである。全てのレゴロボットにおいても、作成したプログラムを有効にするため、スピードが 0 となった場合、すなわち、配列の末尾まで読み込んだ場合（壁間際まで近づいた場合）という条件にした。

Step2. 目の前の対称物体の色判定

色判別を行う前に、Msleep 関数を用い、1 秒間一時停止させることで、光センサーでの色判別に十分な時間を設け、光センサーで正確な値を取得するように工夫した。

次に、光センサーの値が 170 以上かどうかで、赤玉か赤玉以外かを判定している。この「170」という判定時の閾値を決めた理由は､、超音波センサーの値が「24」の時の青玉、壁、赤玉の「光センサーの数値」を調べたところ、以下に示す範囲となったためである。

超音波センサーの値が「24」の距離における光センサーの数値

青玉：189 ~ 200
壁　：181 ~ 191
赤玉：91 ~ 164

なお、上記の検査結果の数値は、実験室後方の明るい場所で観測した値であり、実験室前方の位場所で観測すると、光センサーの取り得る値が低くなることを観測した。また、光センサーと対象物との距離が近すぎると、光センサーの値が極端に小

さくなることも観測した。
赤玉であると判定した場合は、アームを 40 の強さで回転させている。なお、アームの強さは、20 以下にすると、アーム自身の重さにより、アームが動きにくくなることを観測した。

Step3. 対象物の右側前方部分を探索
赤玉以外であると判定した場合、右側領域の探索を行っている。具体的には、
・右車輪のみを少し前進させることで右側を向く
・少しだけ両輪で前進する
・停止する
という車輪の動きで右側を向くように工夫した。

Step4. 右側前方の対象物の色判定
Step3. と同様の方法で色判定を行っている。色判定で 165 未満であれば、赤玉であればアームを回転し、赤玉以外なら Step3.の逆の処理を行うことで Step2. の状態（向き）に戻している。
ここで、閾値を Step2.で用いた値の「170」から「165」に下げた理由は、右側探索時の、光センサーと対象物との距離が近くなってしまい、光センサーの取り得る値が低くなってしまうためである。

Step4.のように、一度の探索で二度の赤玉判定の処理を行うことで、赤玉を見逃すことが少なくなり、成功動画撮影効率を大幅に上げることができた。

Step5. 右側の未探索領域を探索できるように後退
後退方法として、
・右斜め後ろに後退
（右車輪よりも左車輪の出力を 10 だけ多くして、両輪を 2 秒間逆回転させる）
・左斜め後ろに後退
（左車輪よりも右車輪の出力を 10 だけ多くして、両輪を 2 秒間逆回転させる）
・一時停止する
という車輪の動きで、次の探索開始位置に後退しながら移動している。
この後退方法を行うことで、毎回の探索で壁に向かって「正面」から前進するため、青玉と赤玉の距離が離れていても安定的に赤玉探索可能となった。

以上の Step1. ~ Step5.の処理を、赤玉を探索するまで実行することで、LEGO ロボットの知的制御を実現した。

11.4 各班員の貢献度

以下に、3 班における各班員の貢献度を報告する。

鯨田：40%
尾崎：30%
荒田：30%

12.検討（考察）

12.1 考察概要

ソースコード 1 の GA で得られた図 8 の最適速度データは、滑らかな速度変化ではあるが全体的な速度は速いとは言えない。よって、本考察では、作成した GA を改良して、全体的な速度がより速く、かつ、速度変化も滑らかなデータ、すなわち、「速度の合計値」の高い個体（最低速度データの図の形が釣鐘型でなく矩形波のような形のデータ、つまり、速度の高い状態を維持する区間が多いデータ）を得ることを目標とする。

より「速さ」を重視した最適速度データを求めるために考えられる手段として、
・適応度関数に制約条件を設ける（具体的には、制約違反の場合、ペナルティを与える）
・適応度関数の設計を工夫する（具体的には、重視したい目的関数の部分に係数を付ける）
という 2 点を考えることができる。

しかし、上記の 2 点を考える上で、「GA によって得られる個体データの各目的関数値の取り得る範囲」という情報が必要となる。
例えば、特定の目的関数に対して制約条件を設ける際に、制約の目安とするための、目的関数値の範囲を調べる必要がある。また、適応度関数の優先したい目的関数に係数を付与する場合も、競合する目的関数の単位の違いによる比較・優先レベルの違いを調べる必要がある。適応度関数を構成する各目的関数は、共通の単位を使用していないため、単純に比較することができない。[2)]

そこで、適応度関数を構成する目的関数間のトレードオフ関係（パレートフロント）を求めることで、目的関数間の競合関係を考慮した制約条件を設けることができると考察した。

ソースコード 1 における適応度関数式②は、「速度の合計値」を最大化すると同時に「速度変化の二乗の合計値」を最小化する「2 目的最適化モデル」（多目的最適化問題）である。通常、多目的最適化問題における複数の評価基準は互いに競合することが多く，そのような場合にはただ 1 つの最適解は存在しない。そのため，多目的最適化では，「パレート最適解」という概念を用いて探索を行う。パレート最適解とは，「ある目的関数の値を改善するためには，少なくとも他の 1 つの目的関数の値を改悪せざるを得ないような解」と定義されている。パレート最適解は「複数の評価基準を持つ多目的問題における、トレードオフ関係を考慮した解の集合」であるので複数存在する。つまり、「多目的最適化問題」における「パレート最適解」を求めることで、目的関数間に存在するトレードオフ関係を考慮した GA の

個体分布を調べることができ、その結果から、「速度合計値の高い個体」の目的関数値の範囲を調べ、速度合計値の高い解個体を目標とした適切な制約条件とペナルティを設けることで、全体的な速度が速く、かつ、速度変化も滑らかな速度データを求めることができると考察した。

12.2 ソースコード 1 における GA の最終世代の解個体の分布

12.2.1 パレート最適解

単一目的問題の場合は、全ての解候補の中で単純に最も優れた解を「最適解」とすることができる。しかし、多目的最適化問題では、各目的関数が互いに競合するため、必ずしも「最適解」を一意に定めることができない。つまり、多目的最適化問題においては、各解の優劣を単純に比較することができない。そこで、通常は他の解と比較して勝るとも劣らない解の集合を得ることになる。このような解の集合をパレート最適解と呼ぶ。[2)]
つまり、他のいかなる解にも優越されない解（全ての目的関数値が他のどの解よりも低くない解、**片方の目的関数値を犠牲にすることなくその解に到達することができない解個体**）が、「パレート最適解」となる。

12.2.2 GA における解個体の分布及びその最終世代のパレート解を求める手法

多目的最適化問題のパレート曲線を求める前に、ソースコード 1 の GA の、各目的関数の評価値における最終世代の解個体の分布、及び最終世代のパレート解を調べることで、パレート曲線を推測した。最終世代の解個体の分布、及びパレート解を求める方法として、各目的関数値を別のテキストファイルに出力し、他の解と比較して少なくとも一つの目的関数値に関して優劣されない解のみをさらに異なるテキストファイルに出力し、Excel を用いて散布図にした。

本多目的最適化問題における「パレート解となるための条件」は、以下のようになる。

「*ある解個体 i において、全ての解個体 j と比較した時、(i != j)*
speed_sum[i] <= speed_sum[j] かつ −speed_change[i] <= −speed_change[j]
が不成立の時、解個体 i は「パレート解」といえる。
成立する場合、解個体 i は「劣解」という。

speed_sum：速度の合計値（目的関数）
−speed_change：速度変化の差の二乗の合計値（目的関数）」

なお、speed_change では、最小化問題を最大化問題に変えるために−（マイナス）を付けている。

12.2.3 実行結果

最終世代における解個体の分布及び、最終世代におけるパレート解を求めるプログラムソースをソースコード 3 に示す。
なお、曲線を描く個体数が多いほど、パレート曲線の形を具体的に可視化することができるため、個体数をソースコード 1 と比べて 5 倍の 251 とした。

・ソースコード 3 を実行して得られた最適速度データ及び、各世代における GA の進化曲線（各世代における適合度の変化）を図 9 に示す。を図 14、図 15 に示す。また、この最適速度データ（最終世代のエリート個体）の適合度は 782 であった。図 14、図 15 より、個体数を 5 倍に増やしても今回作成した GA では最適解を求めることができ、適応度も 782 に収束していることを確認することができた。

・ソースコード 3 を実行して得られた最終世代の個体の分布、及び最終世代におけるパレート解をそれぞれ図 16、図 17 に示す。図 16 より、設計した適応度関数による最終的な個体の分布は二次関数上に収束することが分かる。これは、適応度関数に、二次関数である速度変化の差の二乗と、一次関数である速度の合計値があるため、横軸を speed_change として、縦軸を speed_sum とした時、speed_sum だけ平行移動させた横軸対称の負の二次関数の曲線となっているためであると考察することができる。
また、図 17 より、この二次関数曲線中の優越している解の部分の曲線を得られたことを確認できる。この結果から、パレート曲線は、横軸対称の負の二次関数のグラフを、縦軸正方向にずらした曲線になると推測することができる。

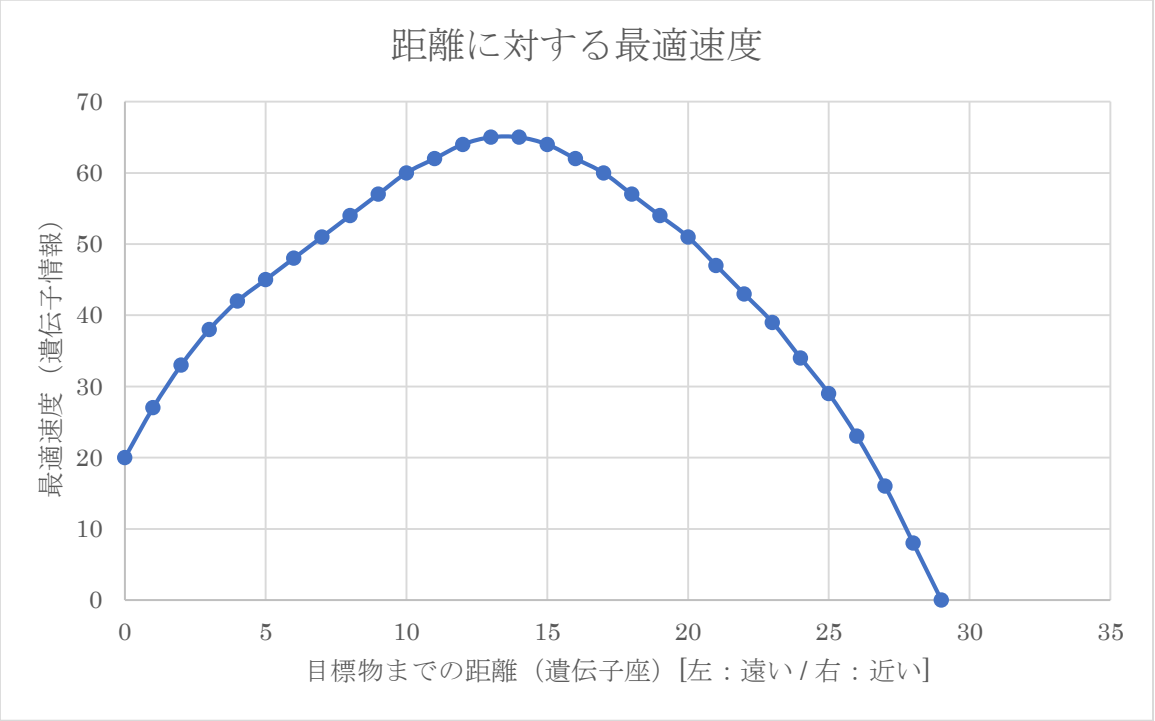


図 14. ソースコード 3（GA）を実行して得られた最適速度データ（個体数 251）

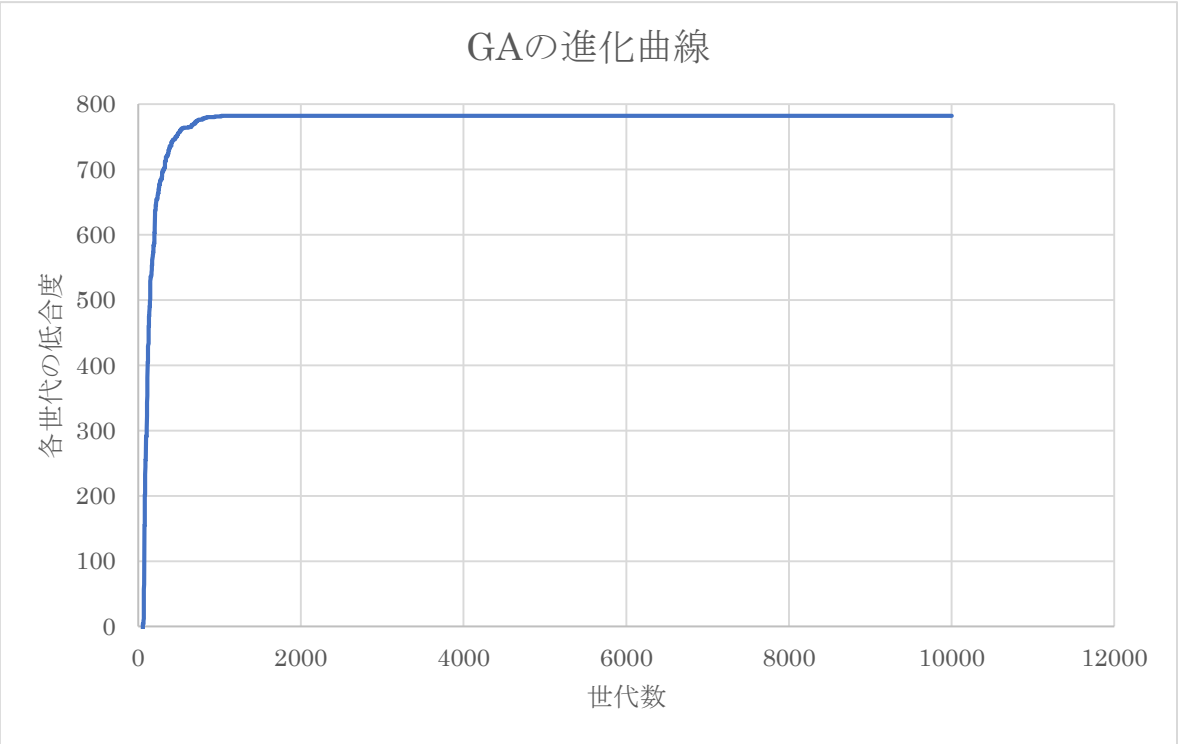


図 15. ソースコード 3 を実行した際の GA の進化曲線（個体数 251）

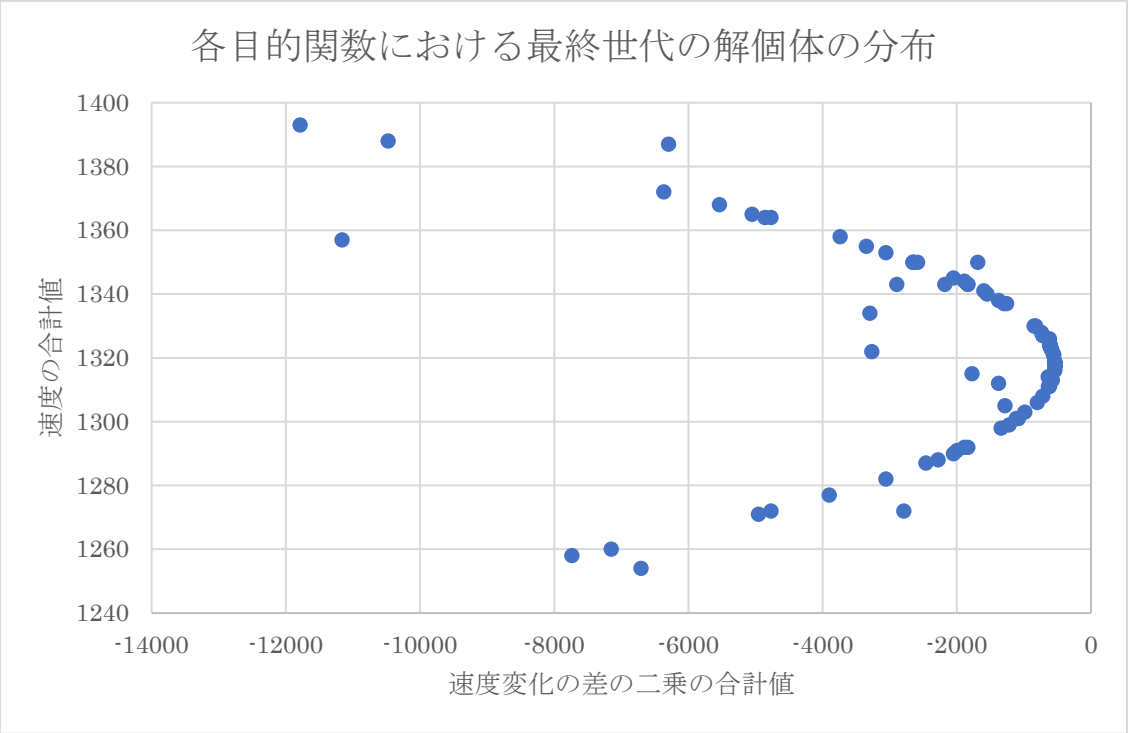


図 16. ソースコード 3 を実行して得られた最終世代の個体の分布

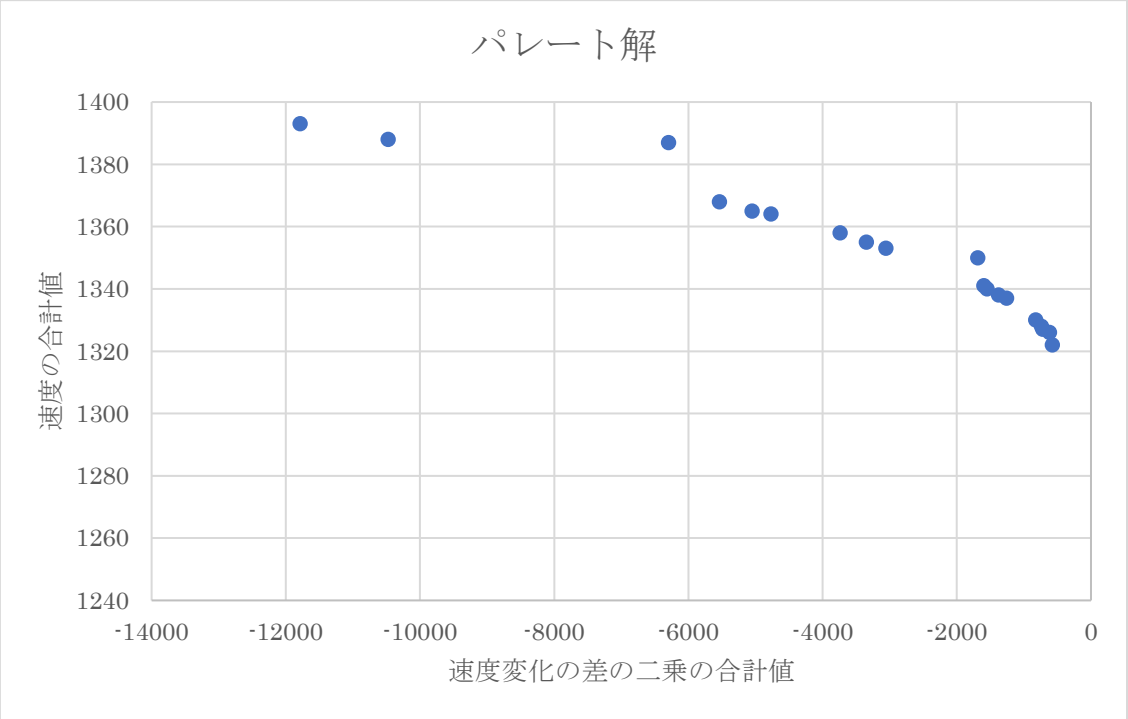


図 17. ソースコード 3 を実行して得られたパレート解の分布

12.3 パレートフロントの導出

12.3.1 多目的 GA の概要

遺伝的アルゴリズムは多点探索法であるため、多目的最適化問題に GA を適用すると、一度の探索で複数のパレート最適解（ある目的関数の値を改善するためには少なくとも 1 つの他の目的関数の値を改悪せざるを得ない解）を求めることができる。
多目的最適化では、単一目的最適化において「1 つの最適解」を求める場合とは異なり、パレート最適解が全て解候補となるため、単純に単一目的における適合度の割り当てをそのまま適用させることはできない。そこで、多目的 GA では、以下のような要求を満たす必要がある。

1. パレート最適解を適切に評価・選択し、次世代に残していくこと。
2. パレート最適開集合をうまく特徴づけるように個体集合の多様性を維持し、パレート最適解を極端な隔たり無くサンプリングすること。
3. 交叉や突然変異などの遺伝演算がパレート最適解を効果的に生成するように構成すること。

12.3.2 作成した多目的 GA の手法・処理手順

パレート最適解が形成するパレート曲線（パレートフロント）を求めるために、多目的 GA（moGA：Multi-Objective-GA）を作成した。
パレートフロントを求めるプログラム（moGA）をソースコード 4 に示す。

多目的最適化問題に遺伝的アルゴリズムを適用する場合に、各世代で得られたパレート最適解を適切に評価・選択し、次世代に残していくことが最も重要である。
よって、作成した moGA は、次世代に残す個体として、その世代のエリート個体に加え、その世代における「パレート最適解」を次世代に引き継がせるように作成した。
なお、各世代におけるパレート最適解は全個体数 251 個中、30～50 個であったため、子世代の構成は、「エリート個体：1 個、パレート最適解：30～50 個、全個体内で進化させた個体：200～220 個」とした。

次に、作成した moGA の各処理手順を以下に示す。

Step1. 初期個体群の生成

図 6 に示した遺伝子型の個体を 251 個生成する。ソースコード 1 同様、初期速度と最終速度は固定値である。

Step2. 各個体の評価

ソースコード 1 同様、以下に示す適合度関数を用いることで、適合度を数値化し、評価した。

$$\text{fitness} = \text{speed\_sum} - (\text{speed\_change})^2 \quad \text{—②}$$

fitness ：適合度

speed_sum：遺伝子の値の合計（スピードデータの速度合計値）

$-(\text{speed\_change})^2$：隣り合う遺伝子座の差の二乗（速度変化の差の二乗の合計値）

Step3. 選択

ここでは、「エリート選択」、「パレート最適解の選択」及び「トーナメント選択」を行っている。

Step3-1. エリート選択

次世代に残す個体として、「その世代において最も適合度の高い個体」、すなわち、「エリート個体」を 1 つ求め、遺伝子情報を保存している。

Step3-2. パレート最適解の探索及び選択

次世代に残す個体として、「その世代にけるパレート最適解の個体」を保存している。パレート最適解として、その個体における 2 つの目的関数値のどちらともが、他のどの解よりも低くない個体の遺伝子情報を保存している。

Step3-3. トーナメント選択

全個体計 251 個の個体から、計 250 個をトーナメントサイズ 2 のトーナメント方式で選択している。なお、各世代でパレート最適解の個数は変動するので、最大数である 250 個を選択している。

Step4. 交叉

Step3-3.で選択した個体間で「一様交叉」を行っている。

Step5. 突然変異

Step4.で交差させた 250 個の子個体に対して、突然変異確率 0.01 で突然変異を行っている。

Step6. 最終的な子個体の作成

Step3-1・3-2 で選択したエリート個体 1 個とパレート最適解、Step3 ~ 5 で進化させた子個体 250 から、子世代（計 251 個）を作成する際に必要となる個数だけを用いて、新たな次世代の個体群を作成している。

以上の Step2 ~ Step6 を 10000 回（世代）繰り返すことにより、パレート最適解を求めた。

12.3.3 実行結果

・パレート最適解を求めるために遺伝的アルゴリズムを用いているので、プログラムの試行毎に、エリートの適応度が変動した。エリート個体の適応度の違いによって、得られたパレート曲線の形は変化しなかったが、パレート曲線の位置に関して「若干のずれ」が生じていることを確認したので、エリート個体の適応度が 752、688、581 の場合の計三種類の最終世代の個体の分布及びパレートフロントを図 18 ~ 23 に示す。なお、横軸は速度変化の差の二乗（負）、縦軸は速度の合計値である。つまり、この目的関数間のグラフにおいて、右に分布するほど「速度変化の差の二乗の合計値」が小さく、上に分布するほど「速度の合計値」が大きいので、「右上」に分布する個体がより適応度が高い個体であると言える。

・図 18 ~ 23 から、得られた曲線の概形は、12.2 で推測した「横軸対称の負の二次関数のグラフを、縦軸正方向にずらした曲線」のような曲線になっているので、この推測は正しかったといえる。しかし、12.2 の図 17 の曲線と異なり、全体的に速度の合計値が大きいような（曲線が縦軸正方向に平行移動した）結果になった。この原因は、次世代に引き継ぐ個体として、その世代のエリート個体に加え、その世代における「パレート解」を引き継いでいる（選択している）ため、元の GA で求めた個体よりも、適応度を上げつつ、曲線をより縦軸正方向にずらそうとしたからである考察することができる。

・図 18～23 から、エリート個体の適応度が高いほど、最終世代の解の分布は速度の合計値がより低い方向に広がりを見せており、逆に適応度が低いほど、最終世代の解の分布は速度の合計値がより高い方向へ（すなわち、パレートフロント周辺へ）集まっていることを観測できる。これは、適応度関数の設計上、速度変化の二乗の合計値の優先度が高いため、適応度を高くしようとすると、速度の合計値を小さくしても、速度変化の二乗の合計値を優先的に小さくしようとするためであると考察することができる。しかし、これらのデータの中でどれが良い個体であるということはなく、あくまでもこれらすべてが多目的最適化問題における「最適解」であると言えるため、今回の解の分布は、適応度に関わらない、目的関数間のトレードオフ関係を表している。（今回作成した moGA の適合度はその作成者の考えた適応度関数における指標にすぎない。）

・平均的なパレート解を求めるために、図 19, 21, 23 のパレート最適解を同じグラフにプロットした散布図（最終的なパレートフロント）を図 24 に示す。エリート個体の適応度が低くなるにつれてパレートフロントが上昇していることを観測できる。エリート個体の適応度が高ければ、「速度の合計値が低く、速度変化の差の二乗の合計値が小さい（速度変化が少ない)」データ分布を得ることができ、逆に適応度が低ければ、「速度の合計値が高く、速度変化の差の二乗の合計値が高い（速度変化が大きい)」データ分布を得ることができた。この原因は、上記でも述べたように、適応度関数を設計している目的関数間で、「速度変化の差の二乗の合計値」が重要視されていることが考えられる。

・図 24 における解個体データ内で、最も右端かつ上端に位置する個体（少なくとも一つの目的関数値を下げなければ到達することが不可能な解個体）の成す曲線がこの問題における「パレートフロント」であると言える。このパレートフロントから、目的関数「速度の合計値」と「速度変化の差の二乗の合計値」とのトレードオフ関係を考察する。
目的関数間のトレードオフ関係としては、速度の合計値が 1500 ~ 2200 に変化すると、速度変化の差の二乗の合計値は-1000 ~ -8000 に大きく変動していることから、速度合計値を約 100 あげる毎に、速度変化の差の二乗の合計値は約 1000 小さくなることが分かる。つまり、速度合計値が大きい個体ほど、適応度としては小さくなってしまうことが分かる。このことは、適応度関数を設計している目的関数間で、「速度変化の差の二乗の合計値」が重要視されているという考察にも一致しており、妥当的な結果であると言える。

したがって、今回設計した適応度関数では、「より速度変化の小さい個体」が生き残ることになり、速度データとしては「滑らかな曲線」が得られると考察することができる。

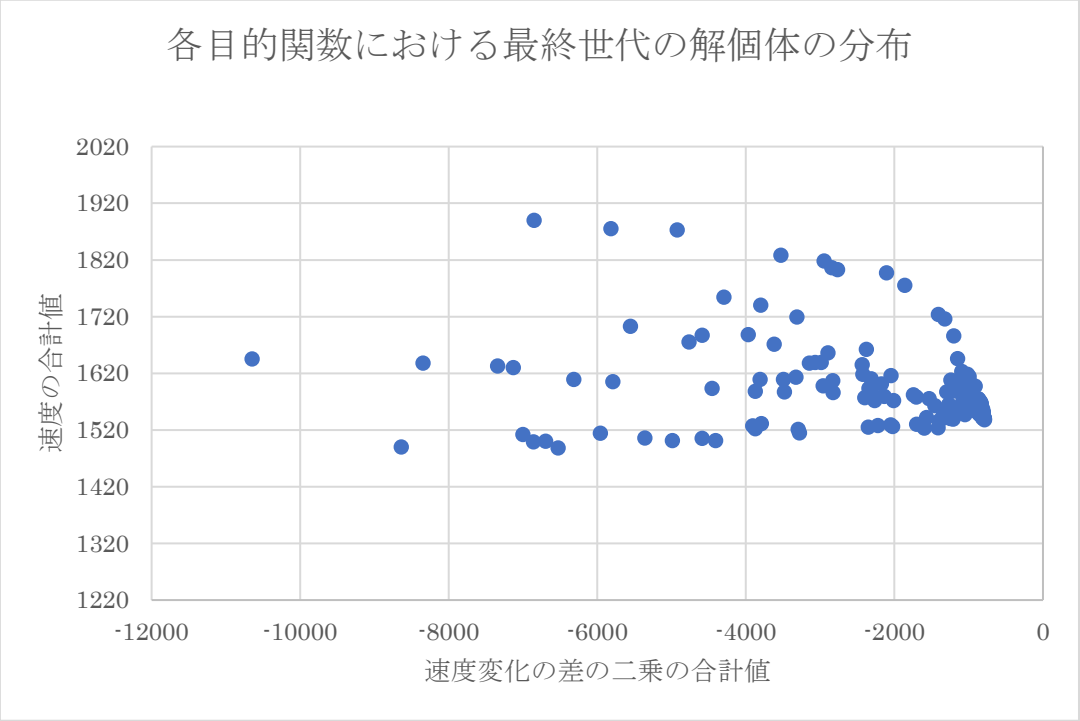


図 18. ソースコード 4 を実行して得られた最終世代の個体の分布
（エリート個体の適応度：752）

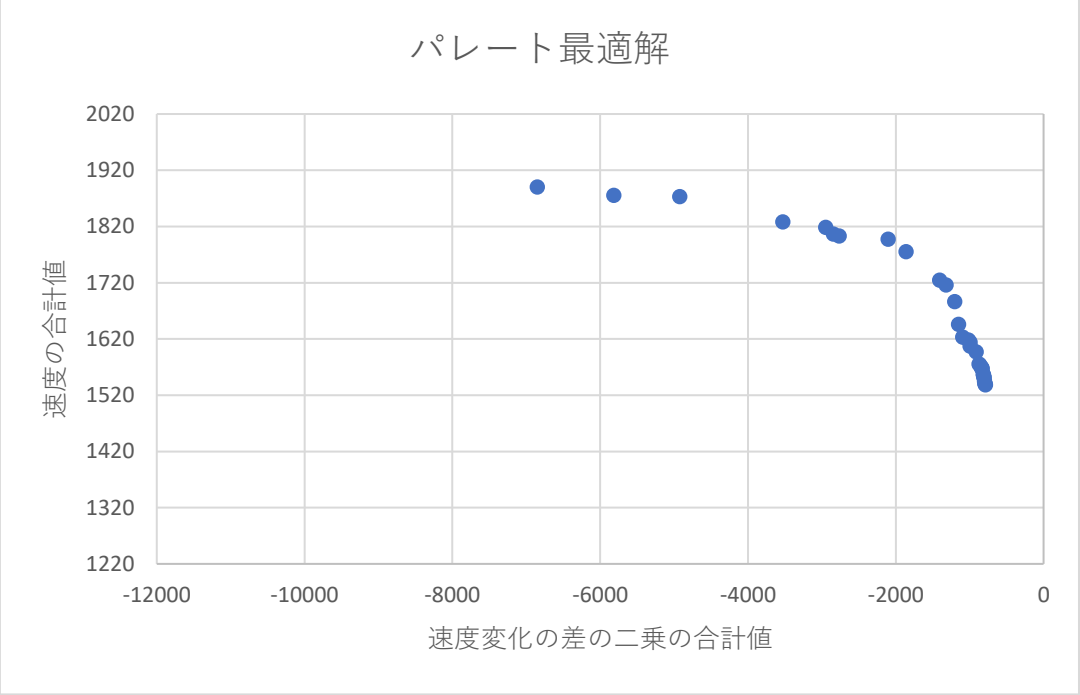


図 19. ソースコード 4 を実行して得られたパレートフロント（エリート個体の適応度：752）

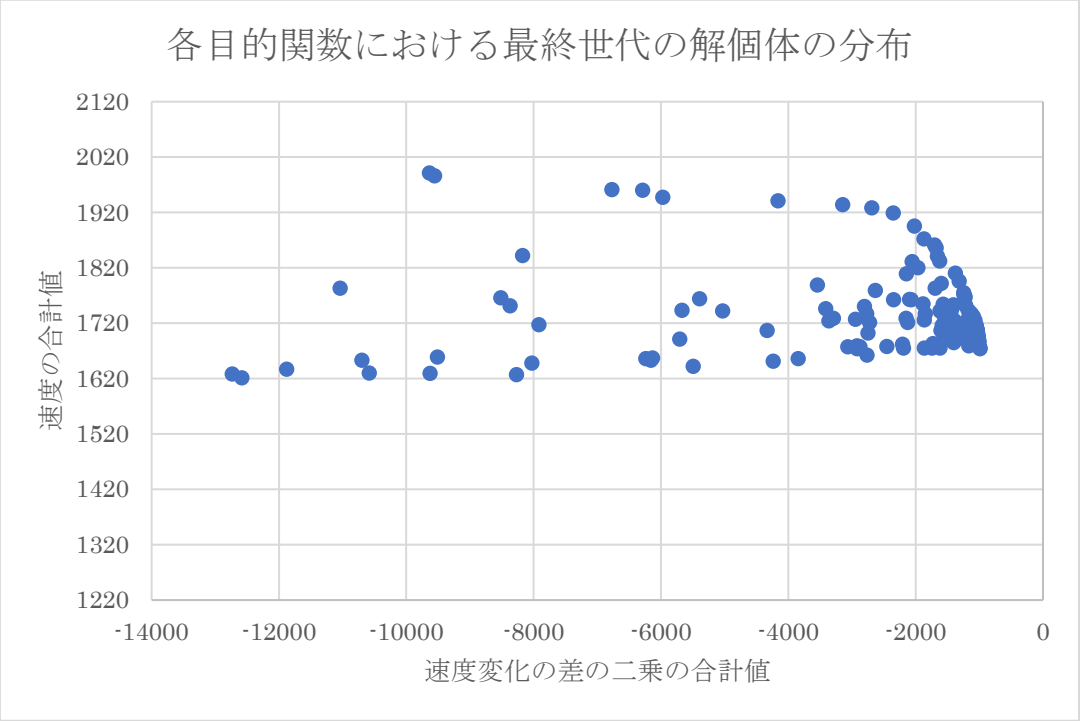


図 20. ソースコード 4 を実行して得られた最終世代の個体の分布
（エリート個体の適応度：688）

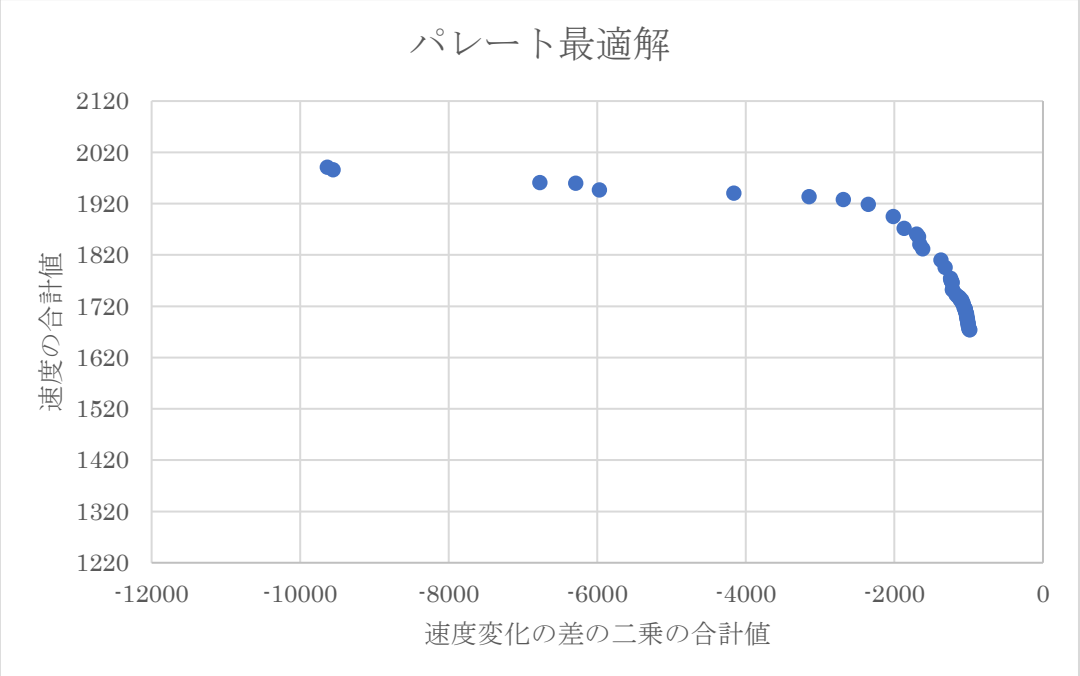


図21. ソースコード4を実行して得られたパレートフロント(エリート個体の適応度:688)

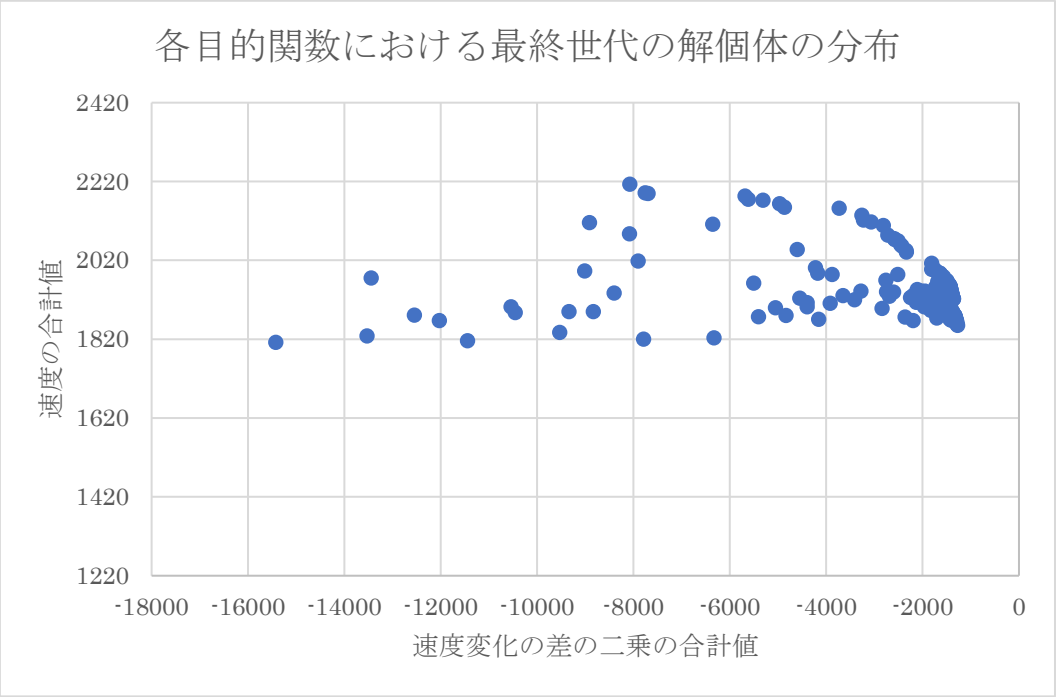


図 22. ソースコード 4 を実行して得られた最終世代の個体の分布
（エリート個体の適応度：581）

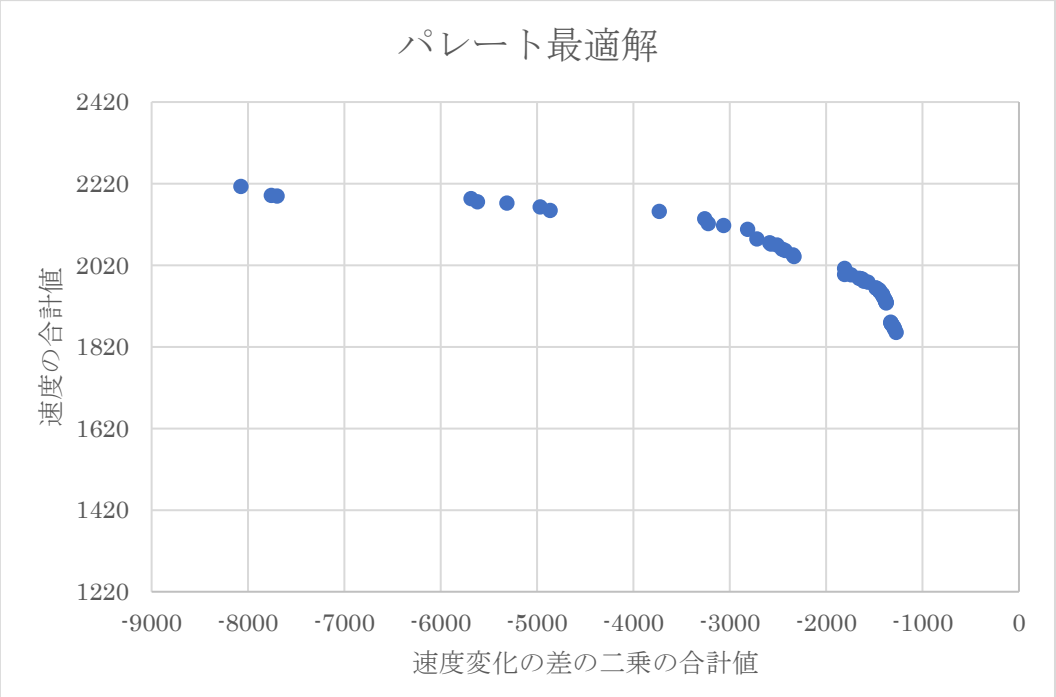


図23. ソースコード4を実行して得られたパレートフロント(エリート個体の適応度:581)

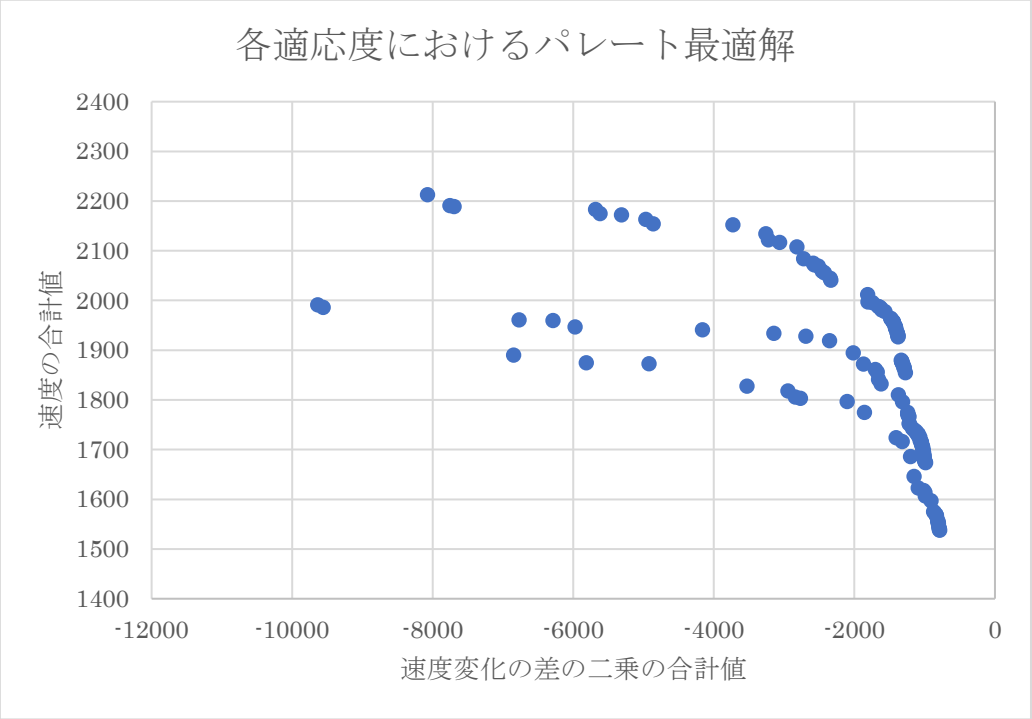


図 24. 図 19, 21, 23 のパレート最適解を同じグラフにプロットした散布図
（最終的なパレートフロント）

12.4 速度合計値の目的関数に制約条件を設けた結果

12.4.1 制約条件、及びペナルティの決定方法

速度の合計値が高い速度データを得るための制約条件について述べる。
12.3 の図 24 の結果から、「速度の合計値が大きく、かつ、速度変化が最小限」である速度データは、速度変化の差の二乗の合計値が「-1500」以内であることが判明した。よって、この範囲内で最適解となる個体の、取り得る速度合計値を調べると「1800～2000」であることが分かる。したがって、目的関数「速度の合計値」に設ける制約条件として、「速度の合計値が 1800 以上」という制約の下、適応度関数で評価することにした。

次に、設けた制約条件を満たさなかった場合に与える「ペナルティ」について述べる。
12.3 の図 24 の結果から、個体の速度の合計値が高くなるほど、爆発的に速度変化の差の二乗の合計値が最大「-8000」程まで大きくなることが判明した。また、速度変化を最小に抑えようとしても、速度変化の差の二乗の合計値は「-1000」以上の値を取ることも判明した。そして、制約条件である「速度の合計値が 1800」の時に取り得る速度変化の差の二乗の合計値の範囲を調べると「-1000 ~ -5000」であることが分かった。これらの結果から、ペナルティを「-10000」に設定すれば、速度の合計値を 1800 以上の個体を優先的に、かつ、確実に、次世代に残すことができると考察した。

12.4.2 作成した制約条件を設けた GA の評価方法の変更点

ソースコード 1 の元の GA に制約条件とペナルティを設けたプログラムを作成した。また、この際、適応度関数の速度合計値の目的関数に係数を付与した。(係数値は 1 としている。)
制約条件及びペナルティを設け、速度合計値の目的関数に係数を付与したプログラムをソースコード 5 に示す。

変更点としては、適応度の計算後に、i 番目の個体の速度の合計値（目的関数）speed_sum が 1800 以上でという制約条件を満たしているかを判定し、1800（= Restriction）以下であれば、計算した適応度（= fitness）からペナルティとして「-10000（= Penalty)」を付与した。プログラム中での簡略化した表現を以下に示す。

if (speed_sum[i] < Restriction) ⇒ fitness[i] -= Penalty;

12.2.3 実行結果

・ソースコード 5（制約条件を設けた GA）を実行して得られた最適速度データを図 25 に示す。なお、横軸は目標物までの距離、縦軸は距離に対する最適速度である。また、この最適速度データ（最終世代のエリート個体）の適合度は 510（速度合計値：1800，速度変化の差の二乗の合計値：-1290）であった。
図 25 より、設計した適合度関数を最大化するように、速度の合計値を 1800 以上にしつつ、かつ、できるだけ速度変化の差の二乗の合計値が最小となるように、速度データの曲線の頂点付近が最大速度 76 を取りながら維持している、矩形波に近いグラフの概形となっていることを確認でき、目標とした速度合計値の高い最適速度データを求めることに成功した。

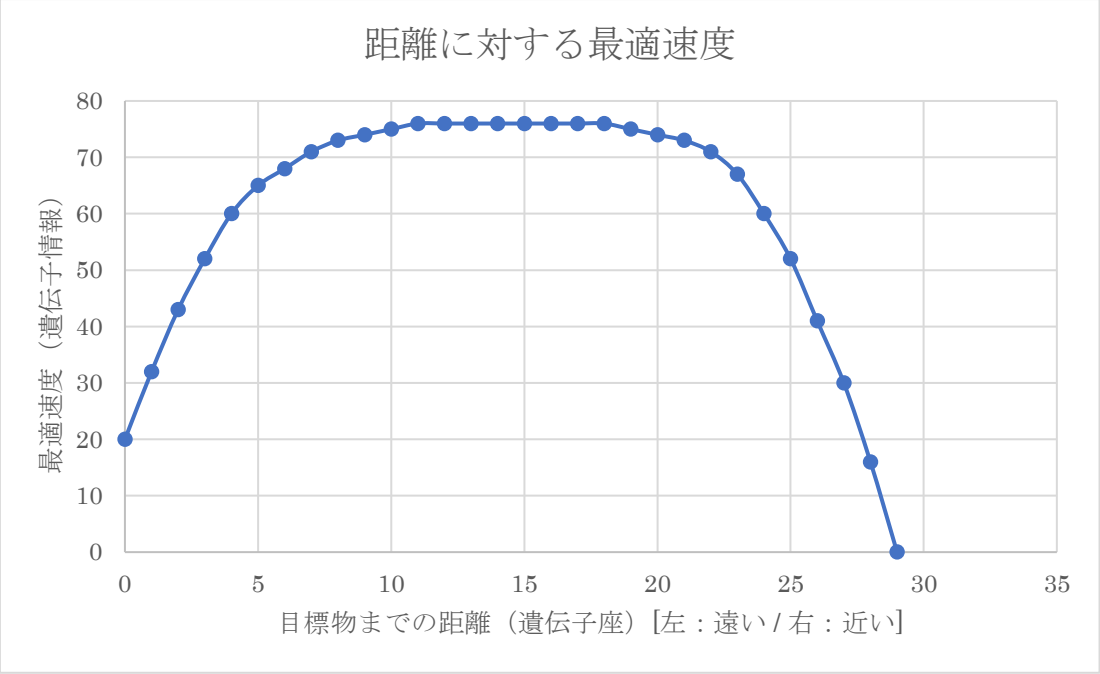


図 25. ソースコード 5（制約条件を設けた GA）を実行して得られた最適速度データ

12.5 適応度関数の改良

12.5.1 目的関数に係数を付与する

12.4 では、目標とした最適速度データを求めることに成功したが、このデータを得るために、ソースコード 5 の GA を 10～15 回程試行する必要があった。「制約条件とペナルティ」を設けるだけでは、局所解に陥ることが多く、ペナルティを受けても速度変化を小さくしようとする動きを観測した。そもそも、適応度関数を設計している目的関数間で、「速度変化の差の二乗の合計値」が重要視されているため、速度重視のデータを得ることが困難であると考えられる。この改善策として、優先したい目的関数に正の係数を付与し、目的関数値を大きくとるようにすることで、他の目的関数と同じ評価レベルとすることができ、改善を図ることができると考察した。

ソースコード 5 の適応度関数内の速度合計値「speed_sum」に、係数として、定義した変数「SPEED_SUM_COEFFICIENT」をかけるように設計したので、この変数を 2, 3, 4 をした場合に得られる速度データを比較する。

12.5.2 実行結果

・ソースコード 5（GA）の速度合計値の目的関数の係数「SPEED_SUM_COEFFICIENT」を 2, 3, 4 とした場合の、実行して得られた最適速度データを図 26 ~ 28 に示す。なお、横軸は目標物までの距離、縦軸は距離に対する最適速度である。また、この最適速度データ（最終世代のエリート個体）の適応度はそれぞれ、2428, 4728, 7022 であった。

速度合計値の制約条件を 1800 以上として、かつ、係数を付与すると、ほぼ 100%に近い確率で図 26～28 のような速度重視の速度データを得ることができた。そして、それらの速度データの曲線は、いずれも釣鐘型ではなく、最高速度を長い区間維持するような速度を重視した曲線となったので、目標とする曲線を得ることに成功したといえる。
図 26～28 から、係数値を大きくするにつれ、速度データの取り得る速度の合計値が大きくなっていることが分かる。また、係数を付与することで、適応度も大きくなっていることから、速度の合計値の目的関数値の優先度が向上したと考察することができる。
しかし、係数を 3 よりも大きくすると、最高速度 100 を取るようになり、得られる速度データは実用性の高いデータとは言えないので、速度合計値が 1800 以上という制約条件を設けた場合に付与する係数は「2」が妥当であると考察した。

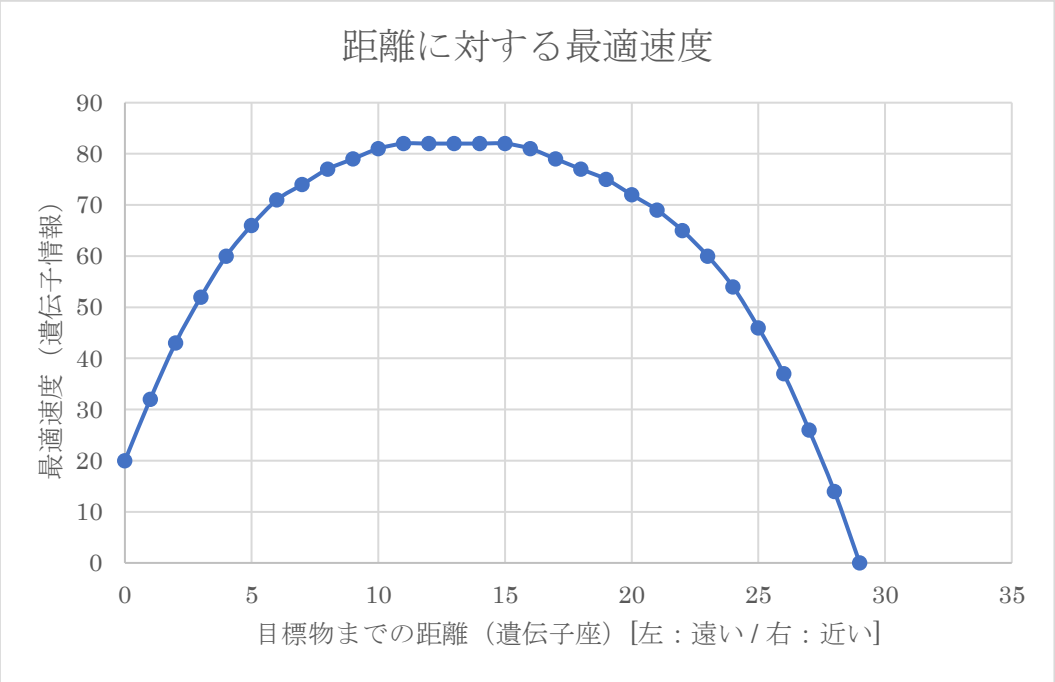


図 26. ソースコード 5（目的関数の係数：2）を実行して得られた最適速度データ

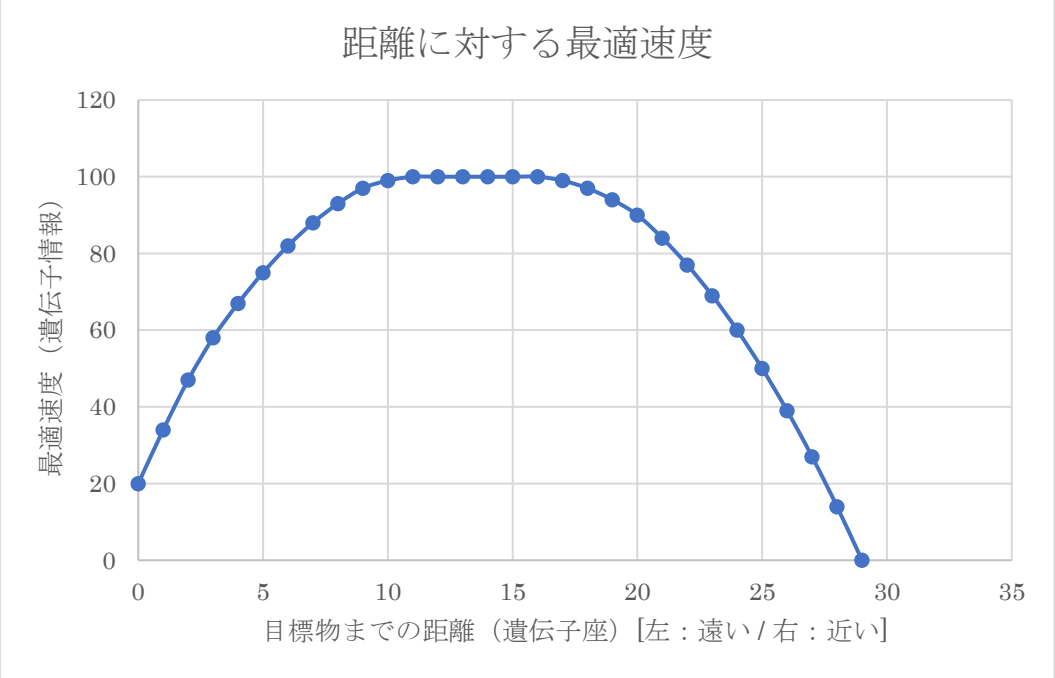


図 27. ソースコード 5（目的関数の係数：3）を実行して得られた最適速度データ

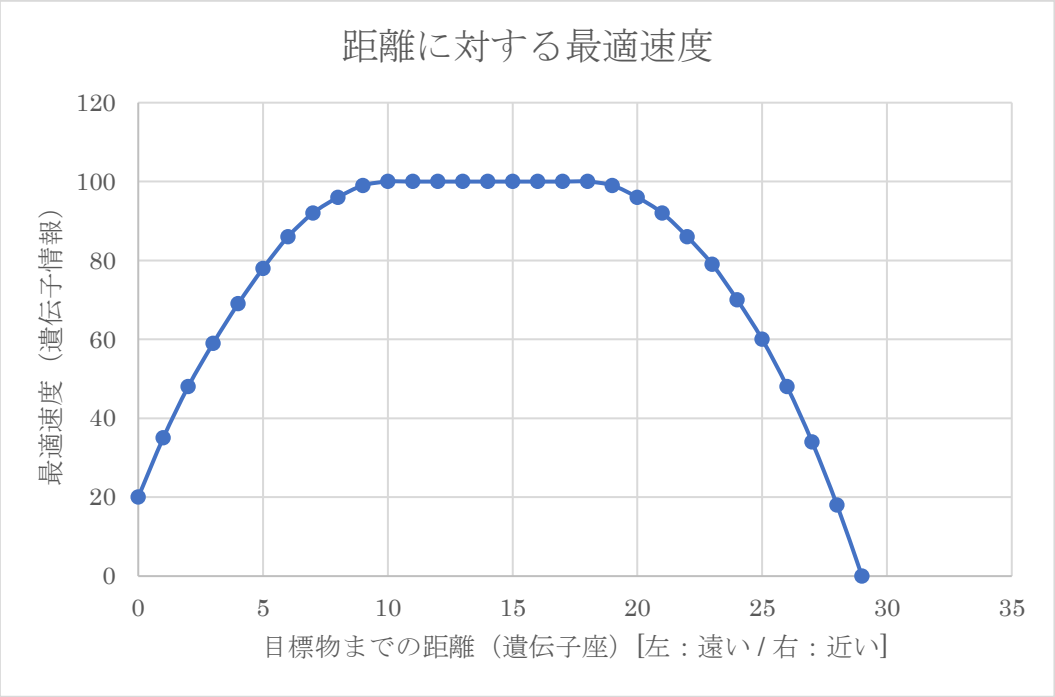


図 28. ソースコード 5（目的関数の係数：4）を実行して得られた最適速度データ

12.6 結論

多目的最適化問題において、目的間のトレードオフ関係を正確に把握するためには、精度と幅広さに優れたパレート解集合を導出することが重要となる。しかし、解集合の幅広さを向上させることによって探索の精度が低下するため、精度と幅広さを同時に向上させることは困難であった。今回の考察では、作成した moGA を実行すると、エリート個体の適応度の違いで得られるパレートフロントの分布が異なったため、各適応度におけるパレートフロントを重ね合わせることで、正確なパレートフロントを導き出すことに成功した。

なお、作成した moGA では、次世代に残す個体として各世代におけるパレート最適解を選択することにより、設計した式②の適応度関数の多目的最適化問題におけるパレートフロントは、「目的関数である速度変化の差の二乗の合計値を横軸かつ対称軸とした負の二次関数の曲線を、目的関数である速度の合計値を縦軸として正方向に平行移動させた曲線」となることを検証することができた。

また、求めたパレートフロントからは、「設計した式②の適応度関数において、目的関数である速度の合計値を 100 上げると、目的関数である速度変化の差の二乗の合計値は 1000 下がる」というトレードオフ関係を見出すことができた。

そして、これらの関係性から、速度変化を損なわずに速度の合計値を最大化するためには、「速度の合計値を 1800 ~ 2000、速度変化の差の二乗の合計値を-1000 ~ -2000」に設定する必要があることを確認した。

さらに、パレートフロントから、上記に示した求めるデータの取り得る目的関数値の範囲を調べ、目的関数に適切な制約条件・ペナルティを設けることで、目標とする速度データを得られることを検証することができた。

最後に、目的関数間に、優先度（比較・評価レベル）の差が存在する場合に、安定的に速度データを獲得するための改善策として、優先したい目的関数に適切な係数を付与することで優先度を上げ、目的関数間での評価の差を減らすことができることも検証することができた。

以上の検証事実から、

速さの合計値の制約条件　　：1800 以上、
制約違反のペナルティ　　：－10000
目的関数（速度の合計値）の係数：2

という、上記の改良を行うことで、目標とする速度データを得られることができると結論される。

12.7 ソースコード

次ページ以降に、「最終世代における解個体の分布及び、最終世代におけるパレート解を求めるソースコード」、「パレートフロントを求めるソースコード（多目的 GA（Multi-Objective-GA : moGA））」「制約条件及びペナルティを設け、速度合計値の目的関数に係数を付与したソースコード」の順に、ソースコードを添付する。

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>      // msleepを使うために必要

//パラメータ宣言
#define IND_NUM     251              // 個体数（個体番号）
#define gene_length 30               // 遺伝子長（遺伝子番号）
#define EVOL_NUM      (IND_NUM - 1)          // エリートを除いた残りの進化個体数（トーナメント選択・交叉を行う個体数）（偶数）

#define generation  10000              // 世代数
#define CROSS_RATE      0.5          // 交叉率
#define MUT_RATE    0.01             // 突然変異率

#define First_Speed 20               // 初期速度固定
#define Last_Speed  0                // 最終速度固定


int main(void)
{
    srand((unsigned)time(NULL));

    int individual[IND_NUM][gene_length];     // 各個体の遺伝子情報（遺伝子[個体番号][遺伝子番号]）
    int fitness[IND_NUM] = {};                // 各個体の適合度
    int all_fitness[generation] = {};         // 全世代における各エリート個体の適合度
    int speed_sum[IND_NUM] = {};              // 遺伝子の値の合計
    int speed_change[IND_NUM] = {};           // 速度変化の2乗の合計

    int elite[gene_length] = {};          // エリート個体用配列
    int elite_number = 0;                 // エリート個体の遺伝子番号（0に初期化）
    int MAX_fitness = 0;                  // 個体の適応度の最大値
    int MAX_IND_NUM = 0;                  // 0番目の個体の遺伝子がエリートであると仮定

    int tournament[EVOL_NUM][gene_length];     // individual[][]から、トーナメント方式で選択した個体
    int cross[EVOL_NUM][gene_length];          // トーナメント選択した個体間で交叉させた新たな個体
    int temp = 0;                              // 交叉用データー時保管変数
    int mutation[EVOL_NUM][gene_length];     // 交叉した個体を突然変異させた個体

    int rnd1 = 0;      // 乱数格納変数
    int rnd2 = 0;

    int i, j, g;      // for文カウンタ


    /*************** 1. 初期個体群生成 ***************/

    for (i = 0; i < IND_NUM; i++) {
        for (j = 0; j < gene_length; j++) {
            if (j == 0) {
                individual[i][j] = First_Speed;
            }
            else if (j == gene_length - 1) {
                individual[i][j] = Last_Speed;
            }
            else {
                individual[i][j] = rand() % 100;      // 1 ～ 100までの乱数
            }
        }
    }

```

```
62
63      /*********************************************************/
64      /*************** 遺伝的アルゴリズム開始 ***************/
65      /*********************************************************/
66      for (g = 0; g < generation; g++) {      // generation回進化する
67
68          printf("generation : %d世代¥t", g + 1);            // 世代の表示
69          printf("〇 MAX_fitness:%7d¥r", MAX_fitness);      // 現世代の適合度
70
71          /*************** 2. 評価 ***************/
72          // 初期化
73          for (i = 0; i < IND_NUM; i++) {
74              speed_sum[i] = 0;
75              speed_change[i] = 0;
76          }
77
78          // 遺伝子の値の合計
79          for (i = 0; i < IND_NUM; i++) {
80              for (j = 0; j < gene_length; j++) {
81                  speed_sum[i] += individual[i][j];
82              }
83          }
84
85          // 速度変化の二乗の合計
86          for (i = 0; i < IND_NUM; i++) {
87              for (j = 0; j < gene_length - 1; j++) {
88                  speed_change[i] += (int)pow((individual[i][j + 1] - individual[i][j]), 2.0);
89              }
90          }
91
92          // 適合度の計算
93          for (i = 0; i < IND_NUM; i++) {
94              fitness[i] = speed_sum[i] - speed_change[i];
95              //fitness[i] = speed_sum[i] - 2 * speed_change[i];
96          }
97
98
99          /*************** 3.選択 ***************/
100         /***** 3-1. エリート選択 *****/
101         // 初期化
102         MAX_fitness = fitness[0];      // 0番目の個体の適応度を最大値に設定
103         MAX_IND_NUM = 0;                  // 0番目の個体の遺伝子をエリートに設定
104
105         // 適応度の最大値を求める
106         for (i = 1; i < IND_NUM; i++) {                  // 0番目をエリートに初期設定しているので、i = 1からスタート
107             if (MAX_fitness < fitness[i]) {
108                 MAX_fitness = fitness[i];
109                 MAX_IND_NUM = i;
110             }
111         }
112
113         // エリート個体の遺伝子情報・適応度を保存
114         for (j = 0; j < gene_length; j++) {
115             elite[j] = individual[MAX_IND_NUM][j];      // 遺伝子情報保存
116         }
117         all_fitness[g] = MAX_fitness;            // 適合度保存
118
119
120         /***** 3-2. トーナメント選択 *****/
121         // エリートを含む計 IND_NUM個(51個)の個体individual[][]から、TEMP_NUM個（IND_NUM - 1個）(50個)をトーナメント方式で選択する
122         for (i = 0; i < EVOL_NUM; i++) {      // EVOL_NUM 回トーナメントを行う
123             rnd1 = rand() % IND_NUM;      // ランダムに2ペア選ぶ(indivisualの個体番号)（エリートを含む）
124             rnd2 = rand() % IND_NUM;
125
126             if (fitness[rnd1] >= fitness[rnd2]) {      // rnd1の個体番号の適応度が大きい、もしくは適応度が等しいまたは rnd1 = rnd2の場合
127                 for (j = 0; j < gene_length; j++) {
128                     tournament[i][j] = individual[rnd1][j];
129                 }
130             }
131             else if (fitness[rnd1] < fitness[rnd2]) {
132                 for (j = 0; j < gene_length; j++) {
133                     tournament[i][j] = individual[rnd2][j];
134                 }
135             }
136         }
137
```

```
138
139         /*************** 4. 交叉 ***************/
140         for (i = 0; i < EVOL_NUM; i += 2) {      // 交叉対象のEVOL_NUM 個（偶数個）の個体間で、EVOL_NUM / 2 回交叉させる
141             rnd1 = rand() % EVOL_NUM;
142             rnd2 = rand() % EVOL_NUM;            // 交叉させる2組を決定
143
144             for (j = 0; j < gene_length; j++) {
145                 cross[i][j] = tournament[rnd1][j];          // 交叉後の配列に交叉前の遺伝子をコピー
146                 cross[i + 1][j] = tournament[rnd2][j];
147
148                 if ( (double)rand() / RAND_MAX < CROSS_RATE ) {              // 0.0〜1.0の乱数が交叉率より大きいか
149                     temp = cross[i][j];                     // 交叉させる
150                     cross[i][j] = cross[i + 1][j];
151                     cross[i + 1][j] = temp;
152                 }
153             }
154
155         }
156
157
158         /*************** 5. 突然変異 ***************/
159         for (i = 0; i < EVOL_NUM; i++) {              // 全ての交叉済みの個体に対して突然変異を行う
160             for (j = 1; j < gene_length - 1; j++) {                  // 初期速度と最終速度は変化させない
161                 if ( (double)rand() / RAND_MAX < MUT_RATE ) {
162                     cross[i][j] = rand() % 100 + 1;          // 突然変異させる
163                 }
164             }
165         }
166         for (i = 0; i < EVOL_NUM; i++) {
167             for (j = 0; j < gene_length; j++) {
168                 mutation[i][j] = cross[i][j];      // 突然変異後のcrossを突然変異済み配列に格納
169             }
170         }
171
172
173         /*************** 6. 最終的な子世代の作成 ***************/
174         for (i = 0; i < IND_NUM; i++) {
175             if (i == 0) {
176                 for (j = 0; j < gene_length; j++) {
177                     individual[i][j] = elite[j];          // individual の先頭にエリートを格納
178                 }
179             }
180             else {
181                 for (j = 0; j < gene_length; j++) {
182                     individual[i][j] = mutation[i - 1][j];      // 残りは進化後の個体に更新
183                 }
184             }
185
186         }
187
188     }
189     /*********************************************************/
190     /*************** 遺伝的アルゴリズム終了 ***************/
191     /*********************************************************/
192
193
194     /***** エリート個体の情報出力 *****/
195
196     // 最終的なエリートの遺伝子情報を表示
197     printf("¥n¥n○last elite:¥n");
198     for (j = 0; j < gene_length; j++) {
199         printf("%d¥t", elite[j]);      // individual[0][j]
200     }
201
202     // 最終的なエリートの適応度を表示
203     printf("¥n○last MAX_fitness:¥t%d¥n¥n", MAX_fitness);
204
205     // 最終的なエリートの速度の合計値を表示
206     printf("¥n○last speed_sum:¥t%d¥n¥n", speed_sum[MAX_IND_NUM]);
207
208     // 最終的なエリートの速度変化の二乗の合計値を表示
209     printf("¥n○last speed_change:¥t%d¥n¥n", speed_change[MAX_IND_NUM]);
210
```

```
211
212     /*****************************************************************************************/
213     /***** パレート解の探索 *****/
214     int Pareto_NUM = 0;
215     int Pareto_speed_sum[IND_NUM] = {};
216     int Pareto_speed_change[IND_NUM] = {};
217     int inferiority_flag = 0;
218
219     for (i = 0; i < IND_NUM; i++) {
220         for (j = 0; j < IND_NUM; j++) {
221             if (i != j && speed_sum[i] <= speed_sum[j] && -speed_change[i] <= -speed_change[j]) {
222                 inferiority_flag = 1;
223                 break;
224             }
225         }
226         if (inferiority_flag == 0) {
227             Pareto_speed_sum[Pareto_NUM] = speed_sum[i];
228             Pareto_speed_change[Pareto_NUM] = speed_change[i];
229             Pareto_NUM++;
230         }
231         inferiority_flag = 0;
232     }
233     /*****************************************************************************************/
234
235
236     // 遺伝的アルゴリズムによる最終的な最適速度をelite.txtに書き込み
237     FILE *f_elite;
238     fopen_s(&f_elite, "elite.txt", "w");
239     for (j = 0; j < gene_length; j++) {
240         fprintf(f_elite, "%d¥n", individual[0][j]);
241     }
242     fclose(f_elite);
243
244     // 遺伝的アルゴリズムによる各世代での適応度をfitness.txtに書き込み
245     FILE *f_fitness;
246     fopen_s(&f_fitness, "fitness.txt", "w");
247     for (g = 0; g < generation; g++) {
248         fprintf(f_fitness, "%d¥n", all_fitness[g]);
249     }
250     fclose(f_fitness);
251
252
253
254     /***** 最終世代の各目的関数の値 *****/
255     // 最終世代の適応度をlast_generation_fitness.txtに書き込み
256     FILE *f_last_fitness;
257     fopen_s(&f_last_fitness, "last_generation_fitness.txt", "w");
258     for (i = 0; i < IND_NUM; i++) {
259         fprintf(f_last_fitness, "%d¥n", fitness[i]);
260     }
261     fclose(f_last_fitness);
262
263     // 最終世代の各個体の速度合計をspeed_sum.txtに書き込み
264     FILE *f_speed_sum;
265     fopen_s(&f_speed_sum, "speed_sum.txt", "w");
266     for (i = 0; i < IND_NUM; i++) {
267         fprintf(f_speed_sum, "%d¥n", speed_sum[i]);
268     }
269     fclose(f_speed_sum);
270
271     // 最終世代の各個体の速度変化の二乗の合計をspeed_change.txtに書き込み
272     FILE *f_speed_change;
273     fopen_s(&f_speed_change, "speed_change.txt", "w");
274     for (i = 0; i < IND_NUM; i++) {
275         fprintf(f_speed_change, "%d¥n", -speed_change[i]);
276     }
277     fclose(f_speed_change);
278
279
280
281     /***** パレート最適解の各目的関数の値 *****/
282     // パレート最適解の個体の速度合計をPareto_speed_sum.txtに書き込み
283     FILE *f_Pareto_speed_sum;
284     fopen_s(&f_Pareto_speed_sum, "Pareto_speed_sum.txt", "w");
285     for (i = 0; i < Pareto_NUM; i++) {
286         fprintf(f_Pareto_speed_sum, "%d¥n", Pareto_speed_sum[i]);
287     }
288     fclose(f_Pareto_speed_sum);
289
290     // パレート最適解の個体の速度変化の二乗の合計をPareto_speed_change.txtに書き込み
291     FILE *f_Pareto_speed_change;
292     fopen_s(&f_Pareto_speed_change, "Pareto_speed_change.txt", "w");
293     for (i = 0; i < Pareto_NUM; i++) {
294         fprintf(f_Pareto_speed_change, "%d¥n", -Pareto_speed_change[i]);
295     }
296     fclose(f_Pareto_speed_change);
297
298     return 0;
299 }
300
```

ソースコード 3. 最終世代における解個体の分布及び、最終世代におけるパレート解を求めるソースコード

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>      // msleepを使うために必要

//パラメータ宣言
#define IND_NUM     251              // 個体数（個体番号）
#define gene_length 30                 // 遺伝子長（遺伝子番号）
#define EVOL_NUM      (IND_NUM - 1)          // エリートを除いた残りの進化個体数（トーナメント選択・交叉を行う個体数）（偶数）

#define generation  10000            // 世代数
#define CROSS_RATE      0.5                      // 交叉率
#define MUT_RATE    0.01             // 突然変異率

#define First_Speed 20               // 初期速度固定
#define Last_Speed  0                // 最終速度固定


int main(void)
{
    srand((unsigned)time(NULL));

    int individual[IND_NUM][gene_length];     // 各個体の遺伝子情報（遺伝子[個体番号][遺伝子番号]）
    int fitness[IND_NUM] = {};                // 各個体の適合度
    int all_fitness[generation] = {};         // 全世代における各エリート個体の適合度
    int speed_sum[IND_NUM] = {};              // 遺伝子の値の合計
    int speed_change[IND_NUM] = {};           // 速度変化の2乗の合計

    int elite[gene_length] = {};          // エリート個体用配列
    int elite_number = 0;                 // エリート個体の遺伝子番号（0に初期化）
    int MAX_fitness = 0;                  // 個体の適応度の最大値
    int MAX_IND_NUM = 0;                  // 0番目の個体の遺伝子がエリートであると仮定

    int pareto[IND_NUM][gene_length] = {};    // パレート解である個体(各世代で次世代に残すパレート解)
    int PARETO_NUM = 0;                       // 各世代におけるパレート解の個数(配列paretoのインデックス)
    int Pareto_speed_sum[IND_NUM] = {};       // パレート解の目的関数値（速度変化の合計）
    int Pareto_speed_change[IND_NUM] = {};    // パレート解の目的関数値（速度変化の二乗の合計）
    int inferiority_flag = 0;                 // 劣等解である場合、1を立てるフラグ

    int tournament[EVOL_NUM][gene_length];    // individual[][]から、トーナメント方式で選択した個体
    int cross[EVOL_NUM][gene_length];         // トーナメント選択した個体間で交叉させた新たな個体
    int temp = 0;                             // 交叉用データー時保管変数
    int mutation[EVOL_NUM][gene_length];      // 交叉した個体を突然変異させた個体

    int rnd1 = 0;     // 乱数格納変数
    int rnd2 = 0;

    int i, j, g;      // for文カウンタ


    /********** 1. 初期個体群生成 **********/

    for (i = 0; i < IND_NUM; i++) {
        for (j = 0; j < gene_length; j++) {
            if (j == 0) {
                individual[i][j] = First_Speed;
            }
            else if (j == gene_length - 1) {
                individual[i][j] = Last_Speed;
            }
            else {
                individual[i][j] = rand() % 100;      // 1 〜 100までの乱数
            }
        }
    }

```

```
68
69      /*******************************************************/
70      /*************** 遺伝的アルゴリズム開始 ***************/
71      /*******************************************************/
72      for (g = 0; g < generation; g++) {      // generation回進化する
73
74          printf("generation : %d世代¥t", g + 1);            // 世代の表示
75          printf("○ MAX_fitness:%7d¥r", MAX_fitness);      // 現世代の適合度
76
77          /*************** 2. 評価 ***************/
78          // 初期化
79          for (i = 0; i < IND_NUM; i++) {
80              speed_sum[i] = 0;
81              speed_change[i] = 0;
82          }
83
84          // 遺伝子の値の合計
85          for (i = 0; i < IND_NUM; i++) {
86              for (j = 0; j < gene_length; j++) {
87                  speed_sum[i] += individual[i][j];
88              }
89          }
90
91          // 速度変化の二乗の合計
92          for (i = 0; i < IND_NUM; i++) {
93              for (j = 0; j < gene_length - 1; j++) {
94                  speed_change[i] += (int)pow((individual[i][j + 1] - individual[i][j]), 2.0);
95              }
96          }
97
98          // 適合度の計算
99          for (i = 0; i < IND_NUM; i++) {
100             fitness[i] = speed_sum[i] - speed_change[i];
101             //fitness[i] = speed_sum[i] - 2 * speed_change[i];
102         }
103
104
105         /*************** 3.選択 ***************/
106         /***** 3-1. エリート選択 *****/
107         // 初期化
108         MAX_fitness = fitness[0];      // 0番目の個体の適応度を最大値に設定
109         MAX_IND_NUM = 0;                  // 0番目の個体の遺伝子をエリートに設定
110
111         // 適応度の最大値を求める
112         for (i = 1; i < IND_NUM; i++) {                  // 0番目をエリートに初期設定しているので、i = 1からスタート
113             if (MAX_fitness < fitness[i]) {
114                 MAX_fitness = fitness[i];
115                 MAX_IND_NUM = i;
116             }
117         }
118
119         // エリート個体の遺伝子情報・適応度を保存
120         for (j = 0; j < gene_length; j++) {
121             elite[j] = individual[MAX_IND_NUM][j];      // 遺伝子情報保存
122         }
123         all_fitness[g] = MAX_fitness;            // 適合度保存
124
```

```
125
126         /*****************************************************************************************/
127         /***** 3-2. パレート解の探索 *****/
128         PARETO_NUM = 0;
129         inferiority_flag = 0;
130
131         for (i = 0; i < IND_NUM; i++) {
132             // i番目の個体と全ての(j番目の)個体の目的関数値を比較
133             for (j = 0; j < IND_NUM; j++) {
134                 if (i != j && speed_sum[i] <= speed_sum[j] && -speed_change[i] <= -speed_change[j]) {
135                     inferiority_flag = 1;
136                     break;
137                 }
138             }
139             if (inferiority_flag == 0) {
140                 // パレート解の遺伝子情報を保存
141                 for (j = 0; j < gene_length; j++) {
142                     pareto[PARETO_NUM][j] = individual[i][j];      // 遺伝子情報保存
143                 }
144                 PARETO_NUM++;           // パレート解の個数更新
145             }
146             inferiority_flag = 0;      // 劣等解のフラグを初期化
147         }
148         /*****************************************************************************************/
149
150
151         /***** 3-3. トーナメント選択 *****/
152         // エリートを含む計 IND_NUM個(51個)の個体individual[][]から、TEMP_NUM個（IND_NUM - 1個）(50個)をトーナメント方式で選択する
153         for (i = 0; i < EVOL_NUM; i++) {      // EVOL_NUM 回トーナメントを行う
154             rnd1 = rand() % IND_NUM;      // ランダムに2ペア選ぶ(indivisualの個体番号)（エリートを含む）
155             rnd2 = rand() % IND_NUM;
156
157             if (fitness[rnd1] >= fitness[rnd2]) {      // rnd1の個体番号の適応度が大きい、もしくは適応度が等しいまたは rnd1 = rnd2の場合
158                 for (j = 0; j < gene_length; j++) {
159                     tournament[i][j] = individual[rnd1][j];
160                 }
161             }
162             else if (fitness[rnd1] < fitness[rnd2]) {
163                 for (j = 0; j < gene_length; j++) {
164                     tournament[i][j] = individual[rnd2][j];
165                 }
166             }
167         }
168
169
170         /*************** 4. 交叉 ***************/
171         for (i = 0; i < EVOL_NUM; i += 2) {      // 交叉対象のEVOL_NUM 個（偶数個）の個体間で、EVOL_NUM / 2 回交叉させる
172             rnd1 = rand() % EVOL_NUM;
173             rnd2 = rand() % EVOL_NUM;           // 交叉させる2組を決定
174
175             for (j = 0; j < gene_length; j++) {
176                 cross[i][j] = tournament[rnd1][j];          // 交叉後の配列に交叉前の遺伝子をコピー
177                 cross[i + 1][j] = tournament[rnd2][j];
178
179                 if ( (double)rand() / RAND_MAX < CROSS_RATE ) {                 // 0.0～1.0の乱数が交叉率より大きいか
180                     temp = cross[i][j];                         // 交叉させる
181                     cross[i][j] = cross[i + 1][j];
182                     cross[i + 1][j] = temp;
183                 }
184             }
185
186         }
187
188
189         /*************** 5. 突然変異 ***************/
190         for (i = 0; i < EVOL_NUM; i++) {                // 全ての交叉済みの個体に対して突然変異を行う
191             for (j = 1; j < gene_length - 1; j++) {                     // 初期速度と最終速度は変化させない
192                 if ( (double)rand() / RAND_MAX < MUT_RATE ) {
193                     cross[i][j] = rand() % 100 + 1;          // 突然変異させる
194                 }
195             }
196         }
197         for (i = 0; i < EVOL_NUM; i++) {
198             for (j = 0; j < gene_length; j++) {
199                 mutation[i][j] = cross[i][j];      // 突然変異後のcrossを突然変異済み配列に格納
200             }
201         }
202
```

```
203
204         /********** 6. 最終的な子世代の作成 **********/
205         for (i = 0; i < IND_NUM; i++) {
206             if (i == 0) {
207                 for (j = 0; j < gene_length; j++) {
208                     individual[i][j] = elite[j];            // individual の先頭にエリートを格納
209                 }
210             }
211             else if (i <= PARETO_NUM) {                         // 各世代によって、パレート最適解の個数は異なるが、大体50個程度
212                 for (j = 0; j < gene_length; j++) {
213                     individual[i][j] = pareto[i - 1][j];      // individual の1～PARETO_NUM番目にパレート解を格納
214                 }
215             }
216             else {
217                 for (j = 0; j < gene_length; j++) {
218                     individual[i][j] = mutation[i - 1][j];      // 残りは進化後の個体に更新
219                 }
220             }
221
222         }
223
224     }
225     /**********************************************************/
226     /********** 遺伝的アルゴリズム終了 **********/
227     /**********************************************************/
228
229
230     /***** エリート個体の情報出力 *****/
231
232     // 最終的なエリートの遺伝子情報を表示
233     printf("¥n¥n○last elite:¥n");
234     for (j = 0; j < gene_length; j++) {
235         printf("%d¥t", elite[j]);      // individual[0][j]
236     }
237
238     // 最終的なエリートの適応度を表示
239     printf("¥n○last MAX_fitness:¥t%d¥n¥n", MAX_fitness);
240
241     // 最終的なエリートの速度の合計値を表示
242     printf("¥n○last speed_sum:¥t%d¥n¥n", speed_sum[MAX_IND_NUM]);
243
244     // 最終的なエリートの速度変化の二乗の合計値を表示
245     printf("¥n○last speed_change:¥t%d¥n¥n", speed_change[MAX_IND_NUM]);
246
247
248     /*****************************************************************************/
249     /***** パレート解の探索 *****/
250     PARETO_NUM = 0;
251     Pareto_speed_sum[IND_NUM] = {};
252     Pareto_speed_change[IND_NUM] = {};
253     inferiority_flag = 0;
254
255     for (i = 0; i < IND_NUM; i++) {
256         for (j = 0; j < IND_NUM; j++) {
257             if (i != j && speed_sum[i] <= speed_sum[j] && -speed_change[i] <= -speed_change[j]) {
258                 inferiority_flag = 1;
259                 break;
260             }
261         }
262         if (inferiority_flag == 0) {
263             Pareto_speed_sum[PARETO_NUM] = speed_sum[i];
264             Pareto_speed_change[PARETO_NUM] = speed_change[i];
265             PARETO_NUM++;
266         }
267         inferiority_flag = 0;
268     }
269     /*****************************************************************************/
270
```

```
271
272     // 遺伝的アルゴリズムによる最終的な最適速度をelite.txtに書き込み
273     FILE *f_elite;
274     fopen_s(&f_elite, "elite.txt", "w");
275     for (j = 0; j < gene_length; j++) {
276         fprintf(f_elite, "%d¥n", individual[0][j]);
277     }
278     fclose(f_elite);
279
280     // 遺伝的アルゴリズムによる各世代での適応度をfitness.txtに書き込み
281     FILE *f_fitness;
282     fopen_s(&f_fitness, "fitness.txt", "w");
283     for (g = 0; g < generation; g++) {
284         fprintf(f_fitness, "%d¥n", all_fitness[g]);
285     }
286     fclose(f_fitness);
287
288
289
290     /***** 最終世代の各目的関数の値 *****/
291     // 最終世代の適応度をlast_generation_fitness.txtに書き込み
292     FILE *f_last_fitness;
293     fopen_s(&f_last_fitness, "last_generation_fitness.txt", "w");
294     for (i = 0; i < IND_NUM; i++) {
295         fprintf(f_last_fitness, "%d¥n", fitness[i]);
296     }
297     fclose(f_last_fitness);
298
299     // 最終世代の各個体の速度合計をspeed_sum.txtに書き込み
300     FILE *f_speed_sum;
301     fopen_s(&f_speed_sum, "speed_sum.txt", "w");
302     for (i = 0; i < IND_NUM; i++) {
303         fprintf(f_speed_sum, "%d¥n", speed_sum[i]);
304     }
305     fclose(f_speed_sum);
306
307     // 最終世代の各個体の速度変化の二乗の合計をspeed_change.txtに書き込み
308     FILE *f_speed_change;
309     fopen_s(&f_speed_change, "speed_change.txt", "w");
310     for (i = 0; i < IND_NUM; i++) {
311         fprintf(f_speed_change, "%d¥n", -speed_change[i]);
312     }
313     fclose(f_speed_change);
314
315
316
317     /***** パレート最適解の各目的関数の値 *****/
318     // パレート最適解の個体の速度合計をPareto_speed_sum.txtに書き込み
319     FILE *f_Pareto_speed_sum;
320     fopen_s(&f_Pareto_speed_sum, "Pareto_speed_sum.txt", "w");
321     for (i = 0; i < PARETO_NUM; i++) {
322         fprintf(f_Pareto_speed_sum, "%d¥n", Pareto_speed_sum[i]);
323     }
324     fclose(f_Pareto_speed_sum);
325
326     // パレート最適解の個体の速度変化の二乗の合計をPareto_speed_change.txtに書き込み
327     FILE *f_Pareto_speed_change;
328     fopen_s(&f_Pareto_speed_change, "Pareto_speed_change.txt", "w");
329     for (i = 0; i < PARETO_NUM; i++) {
330         fprintf(f_Pareto_speed_change, "%d¥n", -Pareto_speed_change[i]);
331     }
332     fclose(f_Pareto_speed_change);
333
334     return 0;
335 }
336
```

ソースコード 4. パレートフロントを求めるソースコード（多目的 GA（Multi-Objective-GA：moGA））

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>      // msleepを使うために必要

//パラメータ宣言
#define IND_NUM     251             // 個体数（個体番号）
#define gene_length 30                  // 遺伝子長（遺伝子番号）
#define EVOL_NUM      (IND_NUM - 1)          // エリートを除いた残りの進化個体数（トーナメント選択・交叉を行う個体数）（偶数）

#define generation  10000           // 世代数
#define CROSS_RATE      0.5                         // 交叉率
#define MUT_RATE    0.01            // 突然変異率

#define First_Speed 20              // 初期速度固定
#define Last_Speed  0               // 最終速度固定

#define Restriction 1800                    // 目的関数の制約条件（速度合計値の下限値）
#define Penalty 10000                       // ペナルティ

#define SPEED_SUM_COEFFICIENT 4                 // 目的関数の速度合計値の係数

int main(void)
{
    srand((unsigned)time(NULL));

    int individual[IND_NUM][gene_length];     // 各個体の遺伝子情報（遺伝子[個体番号][遺伝子番号]）
    int fitness[IND_NUM] = {};                  // 各個体の適合度
    int all_fitness[generation] = {};           // 全世代における各エリート個体の適合度
    int speed_sum[IND_NUM] = {};                // 遺伝子の値の合計
    int speed_change[IND_NUM] = {};                 // 速度変化の2乗の合計

    int elite[gene_length] = {};            // エリート個体用配列
    int elite_number = 0;                       // エリート個体の遺伝子番号（0に初期化）
    int MAX_fitness = 0;                        // 個体の適応度の最大値
    int MAX_IND_NUM = 0;                        // 0番目の個体の遺伝子がエリートであると仮定

    int PARETO_NUM = 0;                             // 各世代におけるパレート解の個数(配列paretoのインデックス)
    int Pareto_speed_sum[IND_NUM] = {};             // パレート解の目的関数値（速度変化の合計）
    int Pareto_speed_change[IND_NUM] = {};      // パレート解の目的関数値（速度変化の二乗の合計）
    int inferiority_flag = 0;                   // 劣等解である場合、1を立てるフラグ

    int tournament[EVOL_NUM][gene_length];      // individual[][]から、トーナメント方式で選択した個体
    int cross[EVOL_NUM][gene_length];           // トーナメント選択した個体間で交叉させた新たな個体
    int temp = 0;                               // 交叉用データー時保管変数
    int mutation[EVOL_NUM][gene_length];      // 交叉した個体を突然変異させた個体

    int rnd1 = 0;      // 乱数格納変数
    int rnd2 = 0;

    int i, j, g;      // for文カウンタ


    /*************** 1. 初期個体群生成 ***************/

    for (i = 0; i < IND_NUM; i++) {
        for (j = 0; j < gene_length; j++) {
            if (j == 0) {
                individual[i][j] = First_Speed;
            }
            else if (j == gene_length - 1) {
                individual[i][j] = Last_Speed;
            }
            else {
                individual[i][j] = rand() % 100;      // 1 〜 100までの乱数
            }
        }
    }

```

```
71
72      /*****************************************************/
73      /************** 遺伝的アルゴリズム開始 ***************/
74      /*****************************************************/
75      for (g = 0; g < generation; g++) {      // generation回進化する
76
77          printf("generation : %d世代¥t", g + 1);          // 世代の表示
78          printf("○ MAX_fitness:%7d¥r", MAX_fitness);     // 現世代の適合度
79
80          /*************** 2. 評価 ***************/
81          // 初期化
82          for (i = 0; i < IND_NUM; i++) {
83              speed_sum[i] = 0;
84              speed_change[i] = 0;
85          }
86
87          // 遺伝子の値の合計
88          for (i = 0; i < IND_NUM; i++) {
89              for (j = 0; j < gene_length; j++) {
90                  speed_sum[i] += individual[i][j];
91              }
92          }
93
94          // 速度変化の二乗の合計
95          for (i = 0; i < IND_NUM; i++) {
96              for (j = 0; j < gene_length - 1; j++) {
97                  speed_change[i] += (int)pow((individual[i][j + 1] - individual[i][j]), 2.0);
98              }
99          }
100
101         // 適合度の計算
102         for (i = 0; i < IND_NUM; i++) {
103             fitness[i] = SPEED_SUM_COEFFICIENT * speed_sum[i] - speed_change[i];
104         }
105
106
107         /*************************************************************************************/
108         //ペナルティチェック
109         for (i = 0; i < IND_NUM; i++) {
110             if (speed_sum[i] < Restriction) {
111                 fitness[i] -= Penalty;
112             }
113         }
114         /*************************************************************************************/
115
116
117         /*************** 3.選択 ***************/
118         /***** 3-1. エリート選択 *****/
119         // 初期化
120         MAX_fitness = fitness[0];     // 0番目の個体の適応度を最大値に設定
121         MAX_IND_NUM = 0;              // 0番目の個体の遺伝子をエリートに設定
122
123         // 適応度の最大値を求める
124         for (i = 1; i < IND_NUM; i++) {             // 0番目をエリートに初期設定しているので、i = 1からスタート
125             if (MAX_fitness < fitness[i]) {
126                 MAX_fitness = fitness[i];
127                 MAX_IND_NUM = i;
128             }
129         }
130
131         // エリート個体の遺伝子情報・適応度を保存
132         for (j = 0; j < gene_length; j++) {
133             elite[j] = individual[MAX_IND_NUM][j];     // 遺伝子情報保存
134         }
135         all_fitness[g] = MAX_fitness;          // 適合度保存
136
```

```
137
138         /***** 3-2. トーナメント選択 *****/
139         // エリートを含む計 IND_NUM個(51個)の個体individual[][]から、TEMP_NUM個（IND_NUM - 1個）(50個)をトーナメント方式で選択する
140         for (i = 0; i < EVOL_NUM; i++) {      // EVOL_NUM 回トーナメントを行う
141             rnd1 = rand() % IND_NUM;      // ランダムに2ペア選ぶ(indivisualの個体番号)（エリートを含む）
142             rnd2 = rand() % IND_NUM;
143
144             if (fitness[rnd1] >= fitness[rnd2]) {      // rnd1の個体番号の適応度が大きい、もしくは適応度が等しいまたは rnd1 = rnd2の場合
145                 for (j = 0; j < gene_length; j++) {
146                     tournament[i][j] = individual[rnd1][j];
147                 }
148             }
149             else if (fitness[rnd1] < fitness[rnd2]) {
150                 for (j = 0; j < gene_length; j++) {
151                     tournament[i][j] = individual[rnd2][j];
152                 }
153             }
154         }
155
156
157         /********** 4. 交叉 **********/
158         for (i = 0; i < EVOL_NUM; i += 2) {      // 交叉対象のEVOL_NUM 個（偶数個）の個体間で、EVOL_NUM / 2 回交叉させる
159             rnd1 = rand() % EVOL_NUM;
160             rnd2 = rand() % EVOL_NUM;           // 交叉させる2組を決定
161
162             for (j = 0; j < gene_length; j++) {
163                 cross[i][j] = tournament[rnd1][j];          // 交叉後の配列に交叉前の遺伝子をコピー
164                 cross[i + 1][j] = tournament[rnd2][j];
165
166                 if ( (double)rand() / RAND_MAX < CROSS_RATE ) {                 // 0.0～1.0の乱数が交叉率より大きいか
167                     temp = cross[i][j];                             // 交叉させる
168                     cross[i][j] = cross[i + 1][j];
169                     cross[i + 1][j] = temp;
170                 }
171             }
172
173         }
174
175
176         /********** 5. 突然変異 **********/
177         for (i = 0; i < EVOL_NUM; i++) {                // 全ての交叉済みの個体に対して突然変異を行う
178             for (j = 1; j < gene_length - 1; j++) {                     // 初期速度と最終速度は変化させない
179                 if ( (double)rand() / RAND_MAX < MUT_RATE ) {
180                     cross[i][j] = rand() % 100 + 1;          // 突然変異させる
181                 }
182             }
183         }
184         for (i = 0; i < EVOL_NUM; i++) {
185             for (j = 0; j < gene_length; j++) {
186                 mutation[i][j] = cross[i][j];      // 突然変異後のcrossを突然変異済み配列に格納
187             }
188         }
189
190
191         /********** 6. 最終的な子世代の作成 **********/
192         for (i = 0; i < IND_NUM; i++) {
193             if (i == 0) {
194                 for (j = 0; j < gene_length; j++) {
195                     individual[i][j] = elite[j];          // individual の先頭にエリートを格納
196                 }
197             }
198             else {
199                 for (j = 0; j < gene_length; j++) {
200                     individual[i][j] = mutation[i - 1][j];      // 残りは進化後の個体に更新
201                 }
202             }
203
204         }
205
206     }
207     /*************************************************/
208     /********** 遺伝的アルゴリズム終了 **********/
209     /*************************************************/
210
```

```
211
212     /***** エリート個体の情報出力 *****/
213
214     // 最終的なエリートの遺伝子情報を表示
215     printf("¥n¥n〇last elite:¥n");
216     for (j = 0; j < gene_length; j++) {
217         printf("%d¥t", elite[j]);      // individual[0][j]
218     }
219
220     // 最終的なエリートの適応度を表示
221     printf("¥n〇last MAX_fitness:¥t%d¥n¥n", MAX_fitness);
222
223     // 最終的なエリートの速度の合計値を表示
224     printf("¥n〇last speed_sum:¥t%d¥n¥n", speed_sum[MAX_IND_NUM]);
225
226     // 最終的なエリートの速度変化の二乗の合計値を表示
227     printf("¥n〇last speed_change:¥t%d¥n¥n", speed_change[MAX_IND_NUM]);
228
229
230     /*****************************************************************************/
231     /***** パレート解の探索 *****/
232     for (i = 0; i < IND_NUM; i++) {
233         for (j = 0; j < IND_NUM; j++) {
234             if (i != j && speed_sum[i] <= speed_sum[j] && -speed_change[i] <= -speed_change[j]) {
235                 inferiority_flag = 1;
236                 break;
237             }
238         }
239         if (inferiority_flag == 0) {
240             Pareto_speed_sum[PARETO_NUM] = speed_sum[i];
241             Pareto_speed_change[PARETO_NUM] = speed_change[i];
242             PARETO_NUM++;
243         }
244         inferiority_flag = 0;
245     }
246     /*****************************************************************************/
247
248
249     // 遺伝的アルゴリズムによる最終的な最適速度をelite.txtに書き込み
250     FILE *f_elite;
251     fopen_s(&f_elite, "elite.txt", "w");
252     for (j = 0; j < gene_length; j++) {
253         fprintf(f_elite, "%d¥n", individual[0][j]);
254     }
255     fclose(f_elite);
256
257     // 遺伝的アルゴリズムによる各世代での適応度をfitness.txtに書き込み
258     FILE *f_fitness;
259     fopen_s(&f_fitness, "fitness.txt", "w");
260     for (g = 0; g < generation; g++) {
261         fprintf(f_fitness, "%d¥n", all_fitness[g]);
262     }
263     fclose(f_fitness);
264
265
```

```
266
267     /***** 最終世代の各目的関数の値 *****/
268     // 最終世代の適応度をlast_generation_fitness.txtに書き込み
269     FILE *f_last_fitness;
270     fopen_s(&f_last_fitness, "last_generation_fitness.txt", "w");
271     for (i = 0; i < IND_NUM; i++) {
272         fprintf(f_last_fitness, "%d¥n", fitness[i]);
273     }
274     fclose(f_last_fitness);
275
276     // 最終世代の各個体の速度合計をspeed_sum.txtに書き込み
277     FILE *f_speed_sum;
278     fopen_s(&f_speed_sum, "speed_sum.txt", "w");
279     for (i = 0; i < IND_NUM; i++) {
280         fprintf(f_speed_sum, "%d¥n", speed_sum[i]);
281     }
282     fclose(f_speed_sum);
283
284     // 最終世代の各個体の速度変化の二乗の合計をspeed_change.txtに書き込み
285     FILE *f_speed_change;
286     fopen_s(&f_speed_change, "speed_change.txt", "w");
287     for (i = 0; i < IND_NUM; i++) {
288         fprintf(f_speed_change, "%d¥n", -speed_change[i]);
289     }
290     fclose(f_speed_change);
291
292
293
294     /***** パレート最適解の各目的関数の値 *****/
295     // パレート最適解の個体の速度合計をPareto_speed_sum.txtに書き込み
296     FILE *f_Pareto_speed_sum;
297     fopen_s(&f_Pareto_speed_sum, "Pareto_speed_sum.txt", "w");
298     for (i = 0; i < PARETO_NUM; i++) {
299         fprintf(f_Pareto_speed_sum, "%d¥n", Pareto_speed_sum[i]);
300     }
301     fclose(f_Pareto_speed_sum);
302
303     // パレート最適解の個体の速度変化の二乗の合計をPareto_speed_change.txtに書き込み
304     FILE *f_Pareto_speed_change;
305     fopen_s(&f_Pareto_speed_change, "Pareto_speed_change.txt", "w");
306     for (i = 0; i < PARETO_NUM; i++) {
307         fprintf(f_Pareto_speed_change, "%d¥n", -Pareto_speed_change[i]);
308     }
309     fclose(f_Pareto_speed_change);
310
311     return 0;
312 }
313
```

ソースコード 5. 制約条件及びペナルティを設け、速度合計値の目的関数に係数を付与したソースコード

13. 調査

調査 1.

経済産業省は国内の組込みソフトウェア産業の実態を把握するために、組み込みソフトウェアに係る全ての企業を対象として、組み込みソフトウェア産業実態調査を 2003 年度に毎年行っている。2010 年度の技術者個人向け調査報告書において、半数以上の技術者が向上させたいと思うスキル及び、そのスキルが必要だと考えられる理由を述べる。

2010 年度の技術者個人向け調査報告書 [3]によると、50%以上の技術者が今後伸ばしたいスキルとして、「通信技術」と「リーダーシップ」の 2 つのスキルを回答していることが分かる。
2009 年度の調査報告書では、「通信技術」スキルを今後伸ばしたいと思う技術者は 40%弱であるが、2010 年度では 50%以上に増加していることから、通信技術の技術向上・時代背景に要因があるのではないかと考察できる。

技術者が「通信技術」のスキルを伸ばしたいと考える理由として、情報社会の進展に伴い、携帯電話・スマートフォン、インターネットなどの情報通信技術（ICT）とそれらをプラットフォームとして展開される新たな生産・サービス・企業活動形態 [4]が、分野課題を超えた共通のインフラ、課題解決の有効なツールとして注目されている時代背景が考えられる。さらに、プロセス効率化による時間短縮・高速通信も実現することができる。
実用例として、中央省庁の業務のコンピュータ化（電子政府化）、インターネットを利用した教育（e-ラーニング）、電子商取引（e-コマース）等がある。

また、無線通信技術を応用し、家電製品等をインターネットに接続する「モノのインターネット」（IoT）の爆発的な利用増加も理由の一つであると考えられる。
IoT によって、モノに対し各種センサーを付け、インターネットを介することで、離れたモノの状態を知ることや、離れたモノを操作することが可能となり、安全で快適な生活を実現できる。[5]さらに、膨大なデータを蓄積できるため、インターネットに接続されたモノがデータを収集・分析し、利用者のニーズに合わせた結果を提供することが可能である。
実用例として、健康管理のツールや自動車のカーナビゲーションサービス、遠隔操作可能な家電製品（エアコン、照明、洗濯機）等がある。

以上より、ICT や IoT による無線高速・高品質な通信を行うためにも「通信技術」を向上させることが求められると考察することができる。

次に、「リーダーシップ」のスキルを伸ばしたいと考える理由として、システム開発を行う際は、必ずチームを構成して開発を進めるため、チームリーダーが率先し、責任感を持ってマネージャーやメンバーとの橋渡しができるスキルが求められることが考えられる。[6)]
いくらチーム内の個人の能力が高くても、メンバーとのコミュニケーションを取れなければ、プロジェクトは成功しない。また、チームには自分の知っている人間だけが所属するわけではなく、初対面の外部の社員や、外国の異文化の人間も所属することも考えられる。
したがって、各個人が、メンバーとして指示待ちになるのではなく、チームリーダーとして「今は何をやるべきなのか」を自主的に考えて行動し、メンバーと連携することが重要である。そして、「チームリーダー」は、プロジェクトマネージャーよりも、より細かいレベルでの作業の割り振りや、問題・課題の潰しこみを行い、チームが能動的に動くことのできる環境を作ることができれば、質の高いアウトプットを出すことができ、プロジェクトを成功へと導くことができる。
しかし、広い視野で現状を把握し、チーム全体の進め方を考え、細かい作業に落とし込み、その作業をできる IT エンジニアが少ないのが現状である。
このため、プロジェクトを円滑に進める能力として、チームのメンバーに的確に指示を出し、かつ、広い視野での現状把握や、積極的にメンバーと連携をとるためにも、「リーダーシップ」の能力向上を希望する技術者が多いのではないかと考えられる。

調査 2.

大学卒業後、組み込みソフトウェア技術者となることを想定した場合、本学科で開講している講義・演習で、学ぶことができる知識やスキルを述べる。また、本学科の講義・演習で学ぶことが出来ないもので、組み込みソフトウェア技術者として必要度の高い知識やスキルにはどのようなものがあるかを述べる。

本学科で開講している講義・演習の中で、組み込みソフトウェア技術者になった際に役に立つものとして、２年・３年時の「情報工学実験Ⅰ・Ⅱ」や「ものづくり創成実習Ⅰ・Ⅱ」がある。

「情報工学実験」においては、全員に共通して与えられた問題を解決することで、講義で学んだプログラミング言語の理解を深めることや、ポインタや構造体、アルゴリズムとデータ構造についての基礎的事項の総復習をすることができた。
また、実験の終わりでは、グループを作り、共通の目的を解決するために意見交換をしあったり、作成したプログラムを見せ合ったりすることで、自分一人では思いつかない斬新なアイディアや、コミュニケーションをとることでより良いプログラムを作成したりすることができた。
以上より、「情報工学実験」では、「基本的なプログラミングスキル」や「グループ演習における協調性・コミュニケーション能力」を学ぶことができたと考えられる。

「ものづくり創成実習」においては、実験の最初から「チーム開発」を行った。達成すべき目標に対して、問題を分割し、分担作業を行うことで、本格的な「チーム開発」をすることができた。そして、分担したプログラムを結合し、それをチームでデバッグする経験もできた。さらに、与えられた問題の難易度も高かったため、「プログラミングに必要な高度の論理的思考」も養うことができた。
また、「ものづくり創成実習Ⅰ」では、チーム開発を期日までに行うために「計画表」を作成することで、チーム開発における「スケジューリングスキル」（円滑に物事を進める力）を身に付けることができた。。
また、「ものづくり創成実習Ⅱ」では、プロジェクトを行う際に、実際に「要求仕様書」「外部設計書」「内部設計書」を作成することで、「設計書の作成スキル」を学ぶことができた。
最後に、実験の最後の時間には自分たちが作成したものを制限時間内にプレゼンテーションをするという時間が設けられており、これも実際の現場に出た際に役に立つのではないかと考えられる。

以上より、「ものづくり創成実習」では、チーム開発に必須なスキルである、「問題分割」「分担作業」「プログラミングの高度な論理的思考」「コミュニケーション能力」「リーダーシップ」「スケジューリングスキル」「設計書作成スキル」「プレゼンテーション能力」を学ぶことができたと考えられる。

次に、本学科の講義・演習で学ぶことができないもので、必要度の高い知識やスキルには「関数型プログラミング」がある。具体的には、情報工学実験やものづくり創成実習で学んだ「手続き型プログラミング」の考え方とは異なる、プログラミングパラダイムに基づく関数型プログラミング言語 Scheme や Haskell、Scala、Standard ML、OCaml を学ぶことができない。

上記に挙げた関数型プログラミング言語は、コンパイラやインタプリタ、静的アナライザを書く言語として優れており、人工知能や自然言語処理、機械学習の研究で使用される言語である。[8)]

関数型プログラミング言語と、オブジェクト指向プログラミング言語 Java を用い、実際にプログラムを作成し、動作させることによって、これらのプログラミングパラダイムの基本的な考え方やその原理となっている計算のメカニズムについて理解を深めることができると考えられる。[7)]
解決しようとしている問題に応じて適切なプログラミングパラダイムを見極め，それに応じたプログラミング言語を用いてシステムを記述することは，システムの記述容易性，理解容易性，保守性などの観点から重要である。

以上より、「手続き型プログラミングとは異なるプログラミングパラダイムに共通の概念や相違点を理解し、問題に応じて適切なプログラミング言語を利用する能力」を、本学科において学ぶことができないと言える。

他に、本学科にない講義として、「外国人講師による英語の専門科目の授業」や「基本情報技術者試験・応用情報技術者試験」等の資格の取得を目標とした実用的な講義がある。グローバル化が進む情報社会において、英語で専門知識を学び、英語で専門的なコミュニケーションをとるスキルはチーム開発において必要なスキルであると言える。また、情報技術者資格も、講義で学んだ知識の定着を促せるとともに、就職活動においても、組み込みソフトウェア技術者としても、有利なスキルであると言える。

調査 3.

現在、ニューラルネットワーク、ファジィ推論、強化学習、遺伝的アルゴリズムなどのソフトコンピューティング技術、機械学習技術は様々な実世界の問題へ応用されている。近年の応用例を調査し参考文献を挙げつつその内容を記述する。

□ニューラルネットワークの応用例「道路の自動走行システム ALVNN」（ 参考文献:9））

ニューラルネットワークによる教師付き学習では、未体験の入力に対する適切な出力として、学習済みの出力からの内挿や外挿を期待する。すなわち、未体験でも何らかの形で出力し、適切に振る舞うことが望まれ、この能力は「汎化」と呼ばれている。この意味するところは、出力の表現が本質的に分散的であるという点である。

ニューラルネットワークの教師付き学習の応用例として、道路の自動走行を実現したシステム「ALVNN」がある。米国カーネギーメロン大学のロボット研究所で開発されたシステムで、人の運転を教師データとして、操舵角を学習する。入力はビデオ画像(30×32 画素)で、隠れ層は 1 層で 4 素子であり、入出力層に全結合している。人間が 5 分間運転し、訓練データを蓄積して、10 分間の BP 法（誤差逆伝搬法）で学習後、自動運転を実現した。それまで時速 4 マイルがせいぜいであったが、高速道路を時速 70 マイルで 90 マイル走行した。

このシステムの出力ノードは、30 種類の操舵角をそれぞれ表す 30 個のユニットである。訓練中は、人間の運転者の教師データに従って、最も近い 1 つのユニットのみを活性化し、その他は活性化しない状態を BP 学習する。
学習後は、誤差が全くなくなることは稀であり、また学習後の入力パタンにもノイズが加わり、類似しているものの、学習時と完全には一致しない。したがって、出力は 30 個のユニットがそれぞれある値を持った状態となる。ALVNN では、出力パタンのガウス関数近似によるピーク検出により、システムの誤差を定義しており、「未経験を含む」という意味で、多様な最終出力を 30 個のノードで分散表現していると言える。[9)]

□ファジィ推論の応用例「家電製品（洗濯機、掃除機）」　（ 参考文献：11)、12) ）

ファジィの大きな特徴は、ノウハウなどの経験的な知識をルールとして記述しやすく、その管理も比較的容易なことが挙げられる。したがって、応用の際の重要な点として、以下のことが挙げられる。

・問題がファジィによる解法に適しているか
・ルールがファジィ処理を行うのに十分なだけあるか
・実際に用いる場合、ハードウェアなどの性能との問題

上記の点を満たすものとして「家電製品」がある。本来、家電は人が行っていた仕事を機械にやらせるものである。したがって、掃除にしても調理にしても、色々なノウハウがある。このような知識は、言語的に表現しやすいのでファジィの絶好の応用分野となる。家電への応用の多くは、ファジィ制御を用いており、ここでは用いられるファジィルールを中心に説明する。

〇洗濯機
洗濯機の場合は、できるだけ短い時間でよく洗浄し、布傷みは少なく、という相反する要求がある。洗浄力を強くすると布の傷みが大きくなることからも、設計条件が厳しいことが理解できる。そこで、洗濯機では、次のようなルールを用いている。[11)]

・もし布量が多く、かつ、布質がごわごわならば、水流を強くし、洗い時間を長くする
　（ジーンズをたくさん洗っているような場合に相当）
・もし布量が少なく、かつ、布質が柔らかならば、水流を弱くし、洗い時間を短くする
　（ワイシャツやブラウスを少量洗っているような場合に相当）

上記のルールにおける「布量」や「布質」などは、センサを用いて検知し、水流の強さや洗い時間をファジィ推論で推定して、時間の無駄を省くと共に、洗濯物を傷めず、きれいに洗うことを目指している。

また、その他には、洗濯機の汚れの質（脂汚れ、泥汚れの程度）や汚れの量（軽い汚れ、ひどい汚れなど）をセンサで検知し，これらの概括的なデータから洗い時間をファジィ推論で推定して、時間の無駄を省くと共に、洗濯物のいたみを防ぐ応用例もある。[12)]

〇掃除機

掃除機を畳や木床にかけると、ゴミはほとんど瞬時に吸い込まれる。しかし、カーペットにかけると、最初は多くのゴミが取れ、その量は次第に減る。また、畳や木床で吸引力を強くして使おうとすると、床ノズルが床面に吸い付いて操作性が悪くなってしまう。そこで、ゴミの量と床面の状態をセンサで推定して、

・もしゴミの量が多く、かつ、床面がカーペット状ならば、吸引力を大きくする

などのルールで吸引力を制御している。[11)]

□強化学習の応用例「囲碁プログラム：AlphaGo」　（ 参考文献：13)、14)、15) ）

強化学習とは、エージェントがある環境において経験から蓄積される報酬を最大にする行動を自動的に学習する機械学習の一種である。この形式は、汎用的であるため制御工学やマルチエージェントの分野でも用いられる。近年では、強化学習を用いることで強いゲーム AI が提案されている。

2016 年 1 月に Google の「AlphaGo」というプログラムが二段のプロ棋士に互先で 5 戦 5 勝の成績を上げたと発表して大ニュースになった。
AlphaGo は

・深層学習（ディープラーニング）
・モンテカルロ木探索
・強化学習

という 3 つの手法をうまく組み合わせている。[13)]

なお、AlphaGo では、評価関数の作成に強化学習を利用している。AlphaGo において評価関数は、policy networks と value networks と呼ばれる deep convolutional neural networks（畳み込みネットワーク）を用いて、概ね以下のように作成されている。[14)]

(1) policy networks をプロ棋士の棋譜データを教師として学習させて P_q とする。
(2) P_q を自己対戦させることで強化学習をして P_p とする。
(3) 強化学習をした P_p を最適方策の近似として利用し、状態 s の最適方策に従った場合の状態価値 v^*(s)の近似として方策 P_p に従った場合の価値 v^P_p(s)を求める。

ここで、最適方策の近似として強化学習により学習した方策が用いられており、AlphaGo は policy networks の方策下で評価関数を学習する。[14)]

次に、AlphaGo の強化学習における学習手法を説明する。AlphaGo は、畳み込みネットワークで構成される方策ネットワークと線形で構成されるロールアウトポリシーを用いてランダムに局面を生成し、その局面の勝率を予測するモデルを構築している。まず、学習に用いる局面のステップ数をランダムに決める。ステップ数の一つ前まではロールアウトポリシーによって手を決める。次に完全ランダムな手を決め、その手を着手した局面を学習に用いる。最終結果はその局面から方策ネットワークを用いてゲームを進めることで得る。このように、AlphaGo は 3000 万の局面を生成し学習を行うことで局面の勝率を予測する評価

値ネットワークを構築し、世界チャンピオンに匹敵するレベルに達した。[13)]

つまり、大量のプロ棋士の棋譜をデータとして，深層学習によってある程度の強さのプログラムを作り，そのプログラム同士の強化学習によってさらに強くしているのである。
従来と異なり、これまでコンピューター囲碁で成功しなかった評価関数を実質的に作ったことが AlphaGo の大きな特徴である。手を決める部分では従来手法であるモンテカルロ木探索を使っている。2016 年 3 月には世界のトップクラスのイ・セドルという韓国人のプロ棋士と対戦して勝ち越して大きな話題になった。事実上囲碁でもコンピューターが人間を抜いたことになる。[15)]

□遺伝的アルゴリズムの応用例「NASA の進化型アンテナ」 （ 参考文献：2） ）

遺伝的アルゴリズムの応用の際の重用点は以下の通りである。

・問題が遺伝的アルゴリズムによる解法に適しているかの判断
・解の明確な評価法
・解を遺伝子表現できるか
・他の技術との組み合わせ
・実際のシステムで解が求められなかった場合の対応策

上記の点を満たす応用例として、「設計問題」がある。ここでは、「進化型アンテナ」について説明する。

NASA では、従来のアンテナ専門家では思いつかないような形状でありながら優れた性能や利点（小型化、設計期間の短縮）を持つアンテナを、GA を用いて設計し、実際に人工衛星に搭載している。これまでに八木・宇田アンテナの最適化、NASA のミッションである火星探査機マース・オデッセイにおける UHF アンテナ、Space Technology 5 プロジェクトにおけるアンテナ（以下 ST5 アンテナと略す）、TDRS-C 衛星用 S バンドアンテナの開発を行っている。ST5 アンテナは極めてユニークな形状で、かつ小型であるが、その高利得による低消費電力化、仰角の変動時における信頼性、設計期間の短縮（従来手法の 5 人月に比べて 3 人月に短縮）などの利点が GA 設計により得られた。[2)]

その他にも、近年、GA を用いた新幹線 700 系の設計が話題となったように、現実世界の様々な分野において、GA の応用が進んでいる。また、製品設計や、最適化、計画、予測、制御、信号処理、知識獲得など、幅広い問題への GA の適用が試みられている。これは、計算機の高速化というハード面での発展とアルゴリズムの高度化というソフト面での発展の相乗効果によるものである。さらに、今後、複雑で大規模な社会問題や環境問題への GA の適用も期待されている。[2)]

14.参考文献

1) 北野宏明　「遺伝的アルゴリズム」
産業図書株式会社（平成 5 年 6 月 3 日　発行）［135 ページ ］[136 ページ ］

2) 電気学会　進化技術応用調査専門委員会編　「進化技術ハンドブック」
株式会社　近代科学社（2010 年 1 月 31 日　発行）
［19 ページ ］[216 ページ ］[99 – 113 ページ]

3) 経済産業省「2010 年版組込みソフトウェア産業実態調査報告書」
＜ http://www.meti.go.jp/policy/mono_info_service/joho/downloadfiles/2010software_research/10gijutusya_houkokusyo.pdf ＞
（2017 年 12 月 30 日閲覧）

4) JICA 研究所「 「オープン・イノベーションと開発」研究会実施結果報告書 」
＜ https://www.jica.go.jp/jica-ri/ja/publication/booksandreports/l75nbg0000037wt6-att/open_innovation_report.pdf ＞
（2017 年 12 月 30 日閲覧）

5) 「MONOWIRELESS」
＜ https://mono-wireless.com/jp/tech/Internet_of_Things.html ＞
（2017 年 12 月 30 日閲覧）

6) 「Crowdtech」
＜ https://crowdtech.jp/blog/?p=392 ＞
（2017 年 12 月 30 日閲覧）

7) 東京工業大学　講義シラバス「H28 年度　情報実験第二」
＜ http://www.ocw.titech.ac.jp/index.php?module=General&action=T0300&JWC=201600496 ＞
（2017 年 12 月 30 日閲覧）

8)「Haskell、Scala、ML、Scheme：あなたが次に学ぶ関数型言語」
＜ http://postd.cc/best-programming-languages/ ＞
（2017 年 12 月 30 日閲覧）

9) 浅田稔、國吉康夫　「ロボットインテリジェンス」
株式会社岩波書店（2006 年 3 月 24 日　発行）［ 161 ~ 162 ページ ］

11) 萩原将文　「ニューロ・ファジィ・遺伝的アルゴリズム」
産業図書株式会社（1994 年 9 月 12 日　発行）［ 136 ~ 141 ページ ］

12) アキューム「ファジィ理論と応用」
＜ http://www.accumu.jp/back_numbers/vol3/%E3%83%95%E3%82%A1%E3%82%B8%E3%82%A3%E7%90%86%E8%AB%96%E3%81%A8%E5%BF%9C%E7%94%A8.html ＞
（2017 年 12 月 30 日閲覧）

13) 水上直紀、鶴岡慶雅「強化学習を用いた効率的な和了を行う麻雀プレイヤ」
＜ https://ipsj.ixsq.nii.ac.jp/ej/?action=pages_view_main&active_action=repository_view_main_item_detail&item_id=175342&item_no=1&page_id=13&block_id=8 ＞
（2017 年 12 月 31 日閲覧）

14) 獄俊太郎、金子知適　「強化学習を用いた評価関数の作成手法の信頼性の分析」
＜ https://ipsj.ixsq.nii.ac.jp/ej/?action=repository_action_common_download&item_id=183866&item_no=1&attribute_id=1&file_no=1 ＞
（2017 年 12 月 31 日閲覧）

15) 松原仁　「ゲーム情報学 コンピューター将棋を超えて」
＜ https://www.jstage.jst.go.jp/article/johokanri/59/2/59_89/_pdf/-char/ja ＞
（2017 年 12 月 31 日閲覧）