卒業研究報告書

題目

EQALINE の解探索

掛け算に着目した解探索

指導教員

石水 隆 講師

報告者

21–1–037–0118

武本 芳樹

近畿大学理工学部情報学科

令和 7 年 2 月 2 日提出

概要

EQUALINE はゲーム開発チーム「丸ダイス」が制作した計算パズルゲームであり, 3×3 中で, 数値と算術演算記号が配置された 3×3 の盤面から定められた値を一筆書きの要領で探し出す計算パズルである. EQUALINE は比較的新しいゲームであるため, このゲームに関する既知の結果はあまりなく, EQUALINE の与えられた盤面に対し, 解を効率よく求めることができる手法は知られていない. 本研究では,EQUALINE の難易度は NP 完全であると証明した. したがって解を得るには本質的には虱潰しに全パターンを試さなければならないと考えられる. また本研究では, EQUALINE の出題盤面の作成およびその盤面の解を求めるプログラムを作成し, また, 積算記号に着目して解く順番に工夫をもたすことで, 少ない手数で解を得ようとするプログラムの作成を行い, 虱潰しに探索を行うものとそれに枝刈りの概念を可能な範囲で適応させたものと比較する. 比較には, ルートを選択し, そのルートの計算結果と答え合わせをした回数を比較する.

目次

1　序論

1.1　本研究の背景

EQUALINE はゲーム開発チーム「丸ダイス」が制作した計算パズルゲームであり, 3×3 中で, 数値と記号が配置された 3×3 の盤面から定められた値を一筆書きの要領で探し出す計算パズルである. 「通った順番で計算される」というルールのもとでゲームを行う [1]. 図 1 に EQUALINE のプレイ画面を示す. 図 1 では, 盤面の左上を起点に $6 \times 2 \times 4 + 8$ と一筆書きの経路で辿っており, その計算結果が右の CALCULATION に表示されている. この値が左の TARGET の値に一致させるのが目的である.

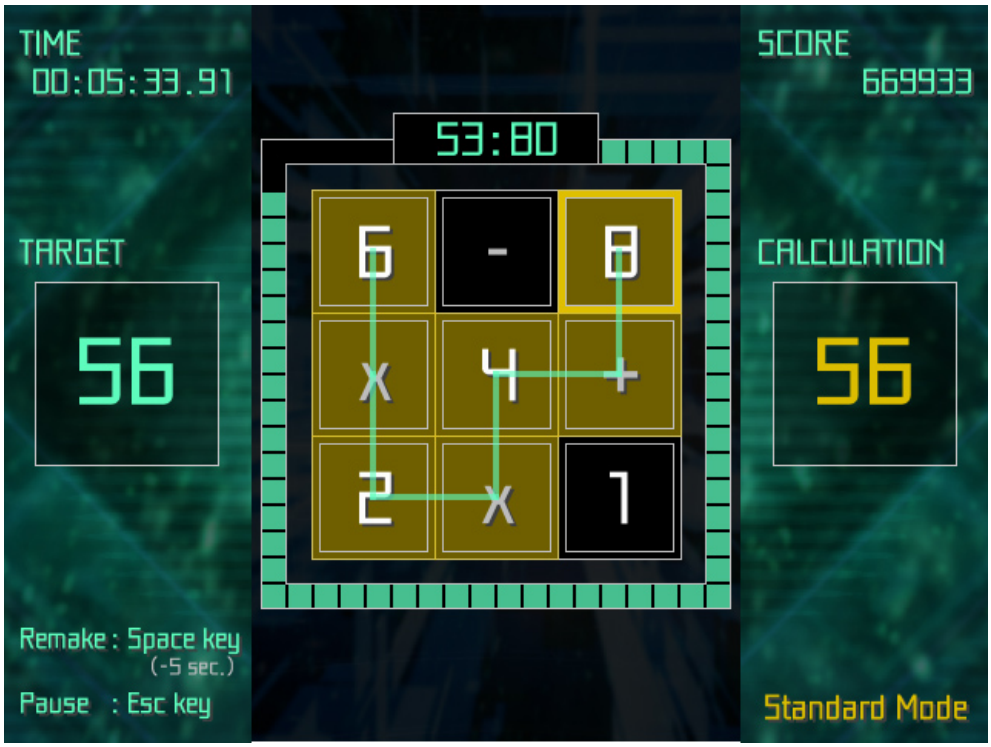

図 1　EQUALINE のプレイ画面 [1]

1.2　EQUALINE に関する既知の結果

EQUALINE は比較的新しいゲームであるため, このゲームに関する既知の結果はあまりない. 既知の結果としては一木が提案した与えられた盤面に対して, その解集合を表現する ZDD(Zero-suppressed Binary Decision Diagram) を効率的に構成する手法などがある [2].

1.3　本研究の目的

EQUALINE の与えられた盤面に対し, 解を効率よく求めることができる手法は知られていない. 本研究では, まず EQUALINE は, 部分和問題に帰着できるため NP 完全であることを証明する. EQUALINE は NP 完全であるため, 解を得るには本質的には虱潰しに全パターンを試さなければならないと考えられる. そこで本研究では, 積算記号に着目し解となり得ないパターンの枝刈りを行うことで, 効率の良い解発見アルゴリズムの構築を目指す.

1.4　本報告書の構成

第 2 章では,EQUALINE について記述し, 第 3 章では NP 完全性について証明する. そして, 第 4 章では, プログラムの構成などについて記述し, 第 5 章では, 結果と考察について記述している. 最後に, 第 6 章で結論と今後の課題について記述している.

2　EQUALINE

本章では,EQUALINE について説明をする.

2.1　EQUALINE について

EQUALINE は Puzzler K によって制作され,2019 年にリリースされた, 一筆書き計算パズルといったジャンルのゲームである.

例外はあるが,3×3 の盤面が示され, そこから TARGET という定められた数値になるよう一筆書きで盤面の経路を辿り, 答えを探し出すといったゲームである. 一般は, 盤面と TARGET が与えられたとき正解となるルートが存在しない, あるいは正解のルートが複数あることがありえるが, ゲームとして成立させるためには, 出題される盤面と TARGET は解答となる式が 1 つだけとなるものでなければならない. また, このゲームの経路の計算結果は前から計算される. 例えば,$3 + 8 \times 4$ の経路を辿った場合本来の計算結果は 35 となるが, このゲームでは,$(3 + 8) \times 4$ となり, 計算結果は 44 として扱われる.

EQUALINE のゲームモードは制限時間内で限界まで解く Endless モード, 盤面上に足し算だけ登場するといったものから, 2×2 や 5×5 の盤面で解くなどといった特殊な条件下で, ノルマの問題数を解く Mission モード, 対戦形式で遊べる Calculation Battle モードといった 3 種のモードがある. 今回の研究対象としては, 基本的に Endless や Calculation Battle モードといった一般的な盤面を想定したものを取り扱う. 例外的に,Mission モードから出題される 5×5 の盤面のものも取り扱う.

公開されているルールではないが,Mission モードの一部を除いて, 解答を得られた経路のあとに ×1 を通る経路や +4 して直後に-4 など結果的に無駄なルートを通っても解を得られるような TARGET は生成されないことや計算途中で負の値および 0 となるルートが存在する TARGET は生成されないといったルールがある.

2.2　本研究に関するルール

2.1 節では, このゲームについて説明した. しかし, 執筆者の技術力などで本研究で作成したプログラムでは, 実際の EQUALINE と比べていくつか異なる点が存在する. この節では, プログラムの仕様から実験時に定めたルールについて記述する.

2.2.1　問題条件

本研究で採用している EQUALINE の問題条件は以下の通りである. ここからは,EQUALINE における TARGET は, ターゲットと記載する.

- 盤面生成はランダムに生成する.
- 式の長さは 5,7,9 マスよりランダムに出題する.
- ターゲットの値は 10 から 100 までとする
- 計算途中で負の値にならない.
- 答えは複数解存在する場合もある.

盤面生成は, 基本的にはランダムだが, 符号に重み付をしてなるべく同じ符号が出ないようにしている. 式の長さは 5,7,9 マスよりランダムに出題と記述しているが, 3×3 の場合は特に 9 マスの生成できるパターンが少なく, 10 から 100 までのターゲットに収まらないことや, 計算途中で負の値を取る可能性も高いため, 生成に失敗することが多い. 2.2.2 節で後述するが, その際に再度プログラムを実行するので, 実質的に 5,7,9 マスには, 生成率の偏りが存在する.

2.2.2　ルール

本研究で採用している EQUALINE のルールは以下の通りである.

- まれに盤面から問題条件を満たすターゲットが生成できないことがあるが, その際は, 結果の対象外とし, 無かったことにして再度プログラムの実行をする.
- 答えは, なにかしら一つの解を出せば良いものとする.
- ルートの正誤判定をした回数を数える。

プログラム中, 枝刈りの範囲内かどうか, およびターゲットからの近似値かどうかの確かめを行なっているが、これは試行回数にカウントしていない。この確かめについては, 事前にそのルートの答え合わせをしているか, 解答にはなり得ないとわかっているものである.

3　EQUALINE の NP 完全性

EQUALINE は問題のサイズが大きくなると, 解を得るための計算量が増加する. 本章では,EQUALINE の計算量について述べる.

3.1 NP 完全とは

クラス NP とは多項式時間以内で、問題の解が本当に合っているかの検証を判定できる問題のことを表す. また, 多項式時間還元とは, 問題 A,B があり, 多項式時間で計算可能な問題 A の問題例から問題 B の問題例への関数を f とし, 問題 B の問題例が受理できるとき, 問題 A の問題例が受理されるような関数 f が存在することとなる. 言い換えると, 問題 A を問題 B に出力が変わらないように変換するような関数があるとして, 問題 A, 問題 B も出力は変わらないため, 問題 B を解くと自動的に問題 A も答えが出せるといったものである. なお、この多項式時間還元には, 推移性があり, 問題 A から問題 B への多項式時間還元ができ, 問題 B から問題 C への多項式時間還元が可能である場合, 問題 A から問題 C への多項式時間還元もできる.

そして, 問題 A において,NP 完全というためには,

- 問題 A がクラス NP に属している
- すべての NP 問題が問題 A へ多項式還元可能

以上 2 つの条件を満たしている必要がある. この証明にはチューリング機械を使う必要があるが, 困難である. そこで, 多項式時間還元の推移性を用いることで, NP 完全だと判明している問題 A があれば, 問題 A からクラス NP の問題を問題 B を多項式還元可能であることを示すことで, 全てのクラス NP 問題から問題 B が多項式還元可能となるため, NP 完全であることを示すことができる [3].

3.2 部分和問題

部分和問題は, 正の整数の集合 X と整数 T が与えられ, X の部分集合で要素の和が T となるものが存在するかどうかを判定するものである. これは最小頂点被覆問題が NP 困難であることを前提とし, そこからの帰着で考える. まず任意の無向グラフ G と整数 T を構築し, G に大きさ k の頂点被覆が存在するときに限って和が T の X の部分集合であるようにする.

G の辺に適当な順番で 0 から $m-1$ の番号を振る. こうしたとき X の各要素は各辺 i に対する $b_i :- 4^i$ という整数と, 各頂点 v に対する

$$a_v :- 4^m + \sum_{i \in \Delta(v)} 4^i$$

という整数である. ここで $\Delta(v)$ は v を端点に持つ辺の集合を表す. そしてターゲットとなる整数 T は次のように定める

$$T :- k \cdot 4^m + \sum_{i=0}^{m-1} 2 \cdot 4^i$$

この帰着が正しいことを示す.

- ($\Rightarrow$)G に大きさ k の頂点被覆があると仮定する. $X_C \subseteq X$ を
  1. 大きさ k の頂点被覆に含まれる頂点 v に対する a_v を全て含み,
  2. 大きさ k の頂点被覆に含まれる頂点を端点のちょうど一方に持つ辺 i に対する b_i を全て含む

  ものとして定義する. このとき X_c に含まれる整数の和を四進法で書くと, 最初の m 桁には全て 2 が並び,m 桁より上位の桁は k となる. つまり,X_c の和は T に等しい.

- ($\Leftarrow$)$X' \subseteq X$ の和が T になると仮定する. このとき

$$\sum_{v \in V'} a_v + \sum_{i \in E'} b_i = T$$

となる $V' \subseteq V$ と $E' \subseteq E$ を取ることができる. ここでも T を四進法で表したとする.X には $i < m$ 桁が 1 の整数が三つしかないので, T を計算するときには最初の m 桁で繰り上がりは起きない. また各辺 b_i は i 桁目に 1 を持ち,T の i 桁目は 2 なことから, V' には G の各辺の少なくとも一方の端点が含まれる. つまり,V' は頂点被覆である.
また,4^m より大きいのは頂点に対応する整数だけであり,$\lfloor T/4^m \rfloor = k$ であることから,V' の大きさは k 以下である.

この帰着は明らかに多項式時間で実行できる. 最小頂点被覆問題が NP 困難であることは前提としているため, 部分和問題は NP 困難である [4].

3.3 EQUALINE の NP 完全性の証明

EQUALINE の難易度については, 以下の定理が成り立つ.

定理 3.1 2 行 2 n 列の EQUALINE は NP 完全である.

(証明) $X = \{a, b, c, d, ...\}$(要素数 n) とする. このとき、以下のような 2 行 $\times$2n 列の EQUALINE 問題を考える.

0	+	0	+	0	+	0	+	0	...
+	a	+	b	+	c	+	d	+	...

このとき, 与えられた整数 k となる経路が存在すれば, それは部分和問題の解となる.

部分和問題は NP 困難であることは,3.2 節で示したため, 部分和問題に帰着できる EQUALINE の解を得る難易度は NP 困難となる. また,EQUALINE の解が正しいかどうかの検証は明らかに多項式時間以内で検証可能であるため, クラス NP に属するため,NP 完全である.

□

4 プログラム

本研究では,EQUALINE の出題盤面の作成およびその解を求めるプログラムを作成する. 本章では, 本研究で作成した EQUALINE プログラムについて述べる. 本研究では,Python を用いて EQUALINE の問題作成およびその解を求めるプログラムを作成した. 本研究で作成したプログラムは,EQUALINE の盤面およびターゲットの数値, 解答と試行回数と枝刈りを行えた回数を出力する. 付録に本研究で作成したプログラムのソースを示す。

第 3 章で証明した通り,EQUALINE は NP 完全であり, サイズが大きくなった場合解を得るための効率的な手法は存在しないと考えられる. そこで本研究では積算記号に着目して解く順番に工夫をもたすことで, 少ない手数で解を得ようとするプログラムの作成を行う. また, 虱潰しに探索を行うものとそれに枝刈りの概念を可能な範囲で適応させたものと比較し, 枝刈りによる探索効率の効果を検証する. 比較には, ルートを選択し, そのルートの計算結果と答え合わせをした回数を比較する.

以降では作成したプログラムの各クラスについて述べる.

4.1 eq2.py

eq2 クラスはゲームの核となる部分を記述している. 本研究で扱う 3×3 の盤面と対応する番号を図 4.1 に示す. 左上を 1 とし, 右にいくにつれて数字を増やし, 端に到達すると一段下の左からカウントを再開している. これは, 解答を示すときに利用している.

1	2	3
4	5	6
7	8	9

図 2　盤面の番号の対応

eq2 クラスはメイン関数を実装しており, これを呼び出すことでゲームを行うことが可能である. ゲーム開始時に難易度 E(Easy),N(Normal),H(Hard) を選択すると, 出題の盤面と TARGET が表示される. プレイヤーは, 式の値が TARGET になるように図 4.1 の番号を用いて解答を入力する. メイン関数を呼び出し, ゲームを完了させた様子を図 3,4 に示す.

図 3 では, 盤面のサイズを 3, 難易度を式の長さが 7,9 マスである “H” を選択し正解のルートを入力した際の出力を示している. また, 図 4 では, 盤面のサイズを 5, 難易度を式の長さが 5,7 マスである “N” を選択し不正解のルートを入力した際の出力を示している. このとき, ターゲット生成時のルートを表示してくれる.

なお, 不正解時の出力文は, 解を得るプログラムで解を探索する際には, 出力文が冗長となってしまうため, コメントアウトしている.

```
盤面のサイズを指定しください。(奇数限定)
3
難易度を設定してください。(E/N/H)
H
8-3
+1-
9*7
target:  67
答えを入力してください(1,2,3...といった形式で)
9,8,7,4,1,2,3,6,5
correct
```

図 3　eq2 の実行画面 1

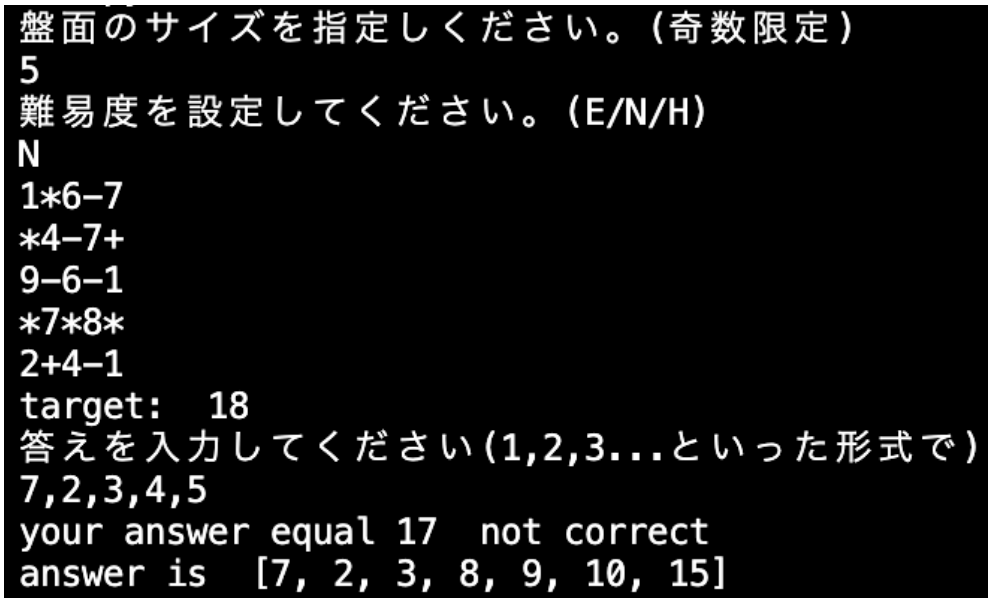

図 4　eq2 の実行画面 2

4.2　tool.py

tool クラスは解探索をする際に共通して使うコードを記述している．主に経路探索における, 現在位置を判別するメソッドとその判別値に応じて, 今までで探索した経路などを記憶しておいた変数と比較して, 移動できる場所を探し出すメソッドをここに記述している．

4.3　brute.py

brute クラスは虱潰しに探索を行うコードを記述している．計算式の長さを 1 マスから判定を行い,3,5 マスと徐々に拡大して, 漏れなく全ての通りを探索していくコードである．図 5 に, 一辺の数の cell num を 3 として実行したものを示す．図 5 は探索によって, 得られた解と試行回数を表示している．

4.4　impbrute.py

impbrute クラスは brute から改良したコードを記述している．具体的には, 枝刈りの概念を取り入れ, 問題条件より 1 マス計算の試行を除去したものである．枝刈りは理想的なものではなく, あくまで既に通っている経路も含めて盤面の数値をソートして, 最大値から順にマイナスの数だけ足し合わせたものをターゲットに加算したものを枝刈りをする上限値として定め, それを超えた経路の探索をやめるといったものである．図 6 にて, 一辺の数の cell num を 5 として実行したものを示す．図 6 は探索によって, 得られた解と枝刈りできた回数と試行回数を表示している．

枝刈りした回数は, 枝刈りによって省略できた回数を示しているのではなく, 上限値を上回って, その経路の続きの探索をしないと決めた回数を示している．したがって,3×3 の 9 マス計算の際には, 探索木における葉の部分にも枝刈りした回数に含まれており, 定義上正しい値とはなっていない．よって, この値自体に関しての考

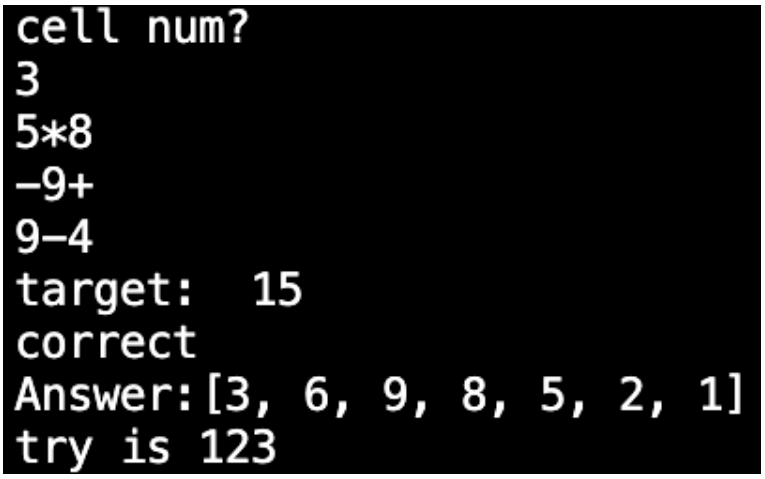

図 5　brute の実行画面

察などはしない.

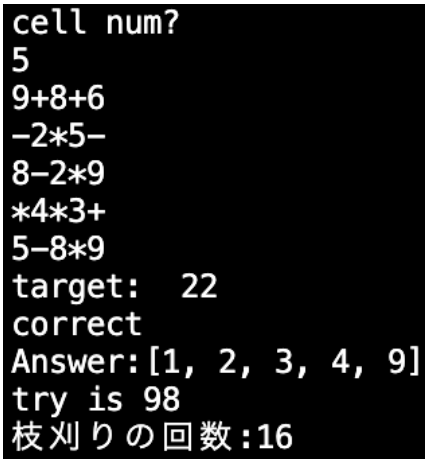

図 6　impbrute の実行画面

4.5　mysolv.py

mysolv クラスは第 4 章冒頭で記述した, 積算記号に着目して解く順番に工夫をもたすといったことを実装したクラスである. 具体的には, まず最後に掛け算で終わる式が解答になるか確認することを最優先で行う. この確認の際は, 終点を先に決めて探索を行うため, 式の逆から経路を探索することになる. その都合上, 枝刈りはここでの探索の際には利用できない. その後, 最初に掛け算で始まる経路について, ターゲットの近似値であれば経路探索を行い, 答えになる経路が存在するか確認する. 近似値というのは,4.4 節で述べた, 枝刈りの範囲内の際に近似値としている. ここでは, プラスとマイナスを含まないまでの間の経路, つまり掛け算だけで続いている経路について, 再帰的に確認している. その後, その他の経路についての探索を行っている.

図 7 にて, 一辺の数の cell num を 5 として実行したものを示す. 図 7 は探索によって, 得られた解と枝刈り

できた回数と試行回数を表示している.

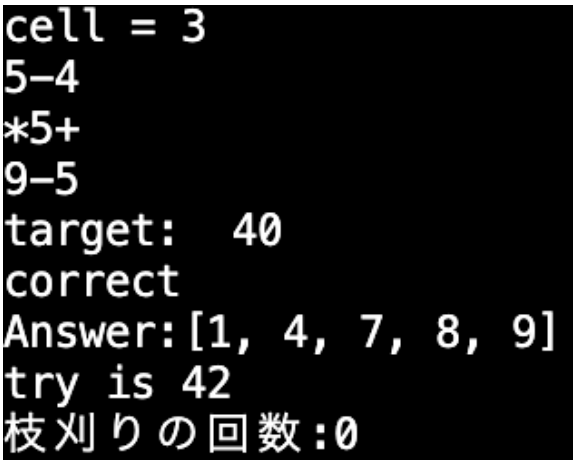

図 7　mysolv の実行画面

5　結果と考察

5.1　結果

4 章で説明した 3 つの解を得るプログラムを盤面サイズ 3×3 および 5×5 に対して, 各 100 回試行した結果を表 1 に示す. 3×3 では, 平均と中央値をみると mysolve が良い値を得られたが, 最悪値はそれほど変わらず,

表 1　結果

		平均	中央値	標準偏差	最悪値
3	brute	75.3	58	52.8	191
×	impbrute	59.4	45	42.9	155
3	mysolv	30.0	15	35.4	150
5	brute	396.1	160.5	660.2	3947
×	impbrute	387.8	139	852.7	7617
5	mysolv	2046	53	4499.5	21625

標準偏差を見てもばらつきは一定程度あることがわかる. brute と impbrute のみで比較した際には,impbrute がすべての値で良い値を出しており, ばらつきも抑えられている。一方,5×5 では,mysolve は中央値は良い値を得たが, いくつかの試行で非常に効率の悪い探索をしているため, 最悪値が impbrute の 3 倍近くとなっており, ばらつきも非常に大きくなっている. brute と impbrute のみで比較した際には, 平均値と中央値は impbrute の方が優れているが, 最悪値に関しては,brute の方が良い値を出した.

5.2 考察

5.1 節のような結果となったのは, 5×5 では,3×3 と比較して別解が多く, ターゲット生成時のルートよりも短いルートが虱潰しの探索で見つかりやすいからだと考えられる.

これは, そもそも mysolv が 3×3 の想定で作成したものであるということにも原因はあるが, 4.5 節でも説明した通り, 積算記号で終わる探索を最優先して行っているため, 約数と積算記号が隣同士にあり, かつその終点の解が存在しない時, 盤面の全てを探索してから, 次の探索をしている. 3×3 では, 終点を除く 7 マスまでの計算を試行するだけだが, 5×5 では,23 マス分の計算の試行をするため, 膨大な量の試行回数となる.

図 8 に長くなった解の例を出す. これの解は,[17,18,19,24,25] の $9 \times 6 - 8 = 46$ というわずか 5 マスの解が存在するが, 約数である 9 マス目の “2” と 14 マス目の積算記号が終点となる全ルートを試していくため,19 マスの経路の解を得ている. 表 1 では, 最悪値は 21625 となっているが, より最悪時には 10 万を超えると考えて

```
cell = 5
6+2-2
+5+2+
1+5*2
+9*6*
1-6-8
target:  46
correct
Answer:[25, 20, 15, 10, 5, 4, 3, 2, 7, 8, 13, 12, 17, 22, 23, 24, 19, 14,
 9]
try is 2412
```

図 8　長い解の例

いる. これは, 積算記号がいくつか存在し, まわりにターゲットの約数があるにも関わらず, 最後が掛け算で終われる解の経路が一切存在しない場合にこういった結果になり得ると考えられる. ただし, 滅多にこういったことは起きないと考えられる. また, 図 9 に impbrute の最悪時について示す. 9 マス計算となっているが, 始

```
cell num?
5
1+9-4
+6+4*
7+2+2
+5+7-
7*8+8
target:  73
correct
Answer:[21, 22, 23, 24, 19, 14, 9, 8, 7]
try is 7617
枝刈りの回数:637
```

図 9　impbrute の最悪時

点が 21 であり,1-19 までを始点とした 9 マス計算や全ての 3,5,7 マス計算でターゲットの値を得られなかったことになる. それに加えて, 掛け算記号が少なくターゲットの値を大きく上回る枝刈りの条件を満たすことも少なかったという, 極めて稀な最悪時の盤面が生成されたため, 最悪値の値が大きくなってしまった. こういったことは, 滅多に起こり得ないと考えられるため, データ数を 1000 にすることや, もう一度データを取り直す

といったことを行えば, 平均の差はもう少し大きくなると考えられる.

6　結論・今後の課題

本研究では,EQUALINE が NP 完全であることを証明し, また,EQUALINE の模倣プログラムおよび解探索を行うプログラムの作成を行なった. 3×3 の盤面においては, 多少の枝刈りを行なった虱潰しの探索と比較しても良い結果を得られることができたが, 5×5 の盤面では, 良い結果を得られることはできなかった. しかし, 単純な枝刈り自体にはある程度の効果が得られるということが分かった.

今後の課題としては, まず,5×5 の盤面では, 掛け算に関わる優先探索は, 9 マス程度のところで打ち切りにするといった, 深く探索しすぎないことで改善できるかもしれない. しかし, 終点が掛け算ではないとわかったが故の探索を後に施しているため, そこを改変する必要が出てくるため, うまく塩梅をとらないといけないと思われる. また, この研究では, 自身の経験に基づいた解き方の一部を再現したものであるため, 機械学習などを用いてみれば, 人間の思考では考えづらい画期的なプロセスが出現して, より良い成果が出せるかもしれない. 他には,3×3 の際には, 複数の解が存在しないなどといった, EQUALINE をより再現できた環境であれば, 違ったアルゴリズムを適用できるといった, 異なる結果が得られると考えられる.

謝辞

本研究を行うにあたり, 石水隆講師には, ゲームの研究に関することをこ指導いただき, 卒業論文の方向性の相談や概要集原稿や卒業論文の添削をしていただきました. また,EQUALINE が NP 完全であることを理解および証明することは, 私だけでは到底できず, 石水隆講師の助言によるものでした. この場をお借りしてお礼申し上げます.

参考文献

[1] Puzzler K:EQUALINE, ゲーム開発チーム「丸ダイス」 (2019) https://marudice.com/equaline/

[2] 一木輝:ZDD による Equaline 解列挙, 情報処理学会 第 85 回全国大会 ポスター発表 (2023) https://www.ipsj.or.jp/event/taikai/85/ipsj_web2023/html/event/B-14.html

[3] 工業大学生ももやまのうさぎ塾:うさぎでもわかる P vs NP 問題（NP 完全、NP 困難の違い）https://www.momoyama-usagi.com/entry/info-p-np

[4] Jeff Erickson Subset Sum (from Vertex Cover), NP-Hardness, Algorithms, Independently published, pp.402-403 (2019) https://jeffe.cs.illinois.edu/teaching/algorithms/book/12-nphard.pdf

付録 A　ソースコード

以下に本研究で作成したプログラムのソースコードを示す.

eq2.py を示す.

```
1  import random
2
3  class Board:
4      """盤面に関するクラス"""
5
6      #target = -1 #ターゲットで初期値で-1
7      #FP = -1 #初期位置
8      #calc = -1 #計算値
9      #cell = -1 #n*n の盤面の n の値
10
11     def __init__(self,cell):
12         """コンストラクタ"""
13         self.cell = cell #n*n の盤面の n の値, 奇数推奨
14         self.board = [] #board を表す変数
15         self.target = -1 #ターゲットで初期値で-1
16         self.FP = -1 #初期位置
17         self.calc = -1 #計算値
18         self.move = [] #回答参照用
19         self.numbers = [] #数値のみを保存（解探索に用いる）
20         self.minusVol = 0 #マイナスの数（解探索に用いる）
21         self.plusVol = 0 #プラスの数（解探索に用いる）
22
23     def createBoard(self):
24         """ランダムで盤面生成する関数"""
25         #盤面の初期化
26         self.board = [-1] #最初の要素は念のため空けておく（要素 1 から始める）
27         #盤面の生成
28         for n in range(int(self.cell * self.cell / 2)):
29             self.board.append(random.randrange(1,10,1))
30             #operator: 0 -> +, 1 -> -, 2 -> *
31             self.board.append(random.choice(["+", "-", "*"]))
32         if(self.cell % 2 == 1): #マスが奇数の時だけもう一度数字を生成する
33             self.board.append(random.randrange(1,10,1))
34         self.board.append(-1) #最後の要素にも空白を入れておく
```

```
35
36      def createBoard2(self):
37          """同一の符号が出にくい盤面生成する関数"""
38          #盤面の初期化
39          self.board = [-1] #最初の要素は念のため空けておく（要素 1 から始める）
40          plus = 1 #+ に関する重み付け
41          minus = 1 #-に関する重み付け
42          cross = 1 #*に関する重み付け
43          #盤面の生成
44          for n in range(int(self.cell * self.cell / 2)):
45              self.board.append(random.randrange(1,10,1))
46              self.numbers.append(self.board[-1])
47              sum = plus + minus + cross
48              decide = random.randrange(1, sum + 1, 1)
49              if(decide <= plus):
50                  self.board.append("+")
51                  self.plusVol += 1
52                  minus += 1
53                  cross += 1
54              elif(decide <= plus + minus):
55                  self.board.append("-")
56                  self.minusVol += 1
57                  plus += 1
58                  cross += 1
59              #else でいいけど、デバック用にここも書いておく。
60              elif(decide <= plus + minus + cross):
61                  self.board.append("*")
62                  plus += 1
63                  minus += 1
64              #デバック用
65              else:
66                  print("盤面生成において、不適切な重み付の発生。")
67                  exit(1)
68
69              #self.board.append(random.choice(["+", "-", "*"]))
70          if(self.cell % 2 == 1): #マスが奇数の時だけもう一度数字を生成する
71              self.board.append(random.randrange(1,10,1))
72          self.board.append(-1) #最後の要素にも空白を入れておく
73
74      def customBoard(self, path):
```

```
75          """指定した盤面を作成する関数 (テスト用)"""
76          for n in path:
77              self.board.append(n)
78              if(len(self.board) % 2 == 0):
79                  self.numbers.append(n)
80
81      def dpBoard(self):
82          """盤面を表示させる関数"""
83          for n in range(self.cell):
84              for m in range(self.cell):
85                  print(self.board[self.cell * n + m + 1], end="")
86              print()
87
88      def calcurate(self, *args):
89          """計算を行う関数"""
90          calc = -1
91          if(len(args) <= 2):
92              return args[0]
93          #誤って偶数で計算する際最後の演算子を無視するために-1 する。
94          for n in range(int((len(args) - 1) / 2)):
95              #最初だけ処理を変えている。
96              if(n == 0):
97                  if(args[1] == "+"):
98                      calc = args[0] + args[2]
99                  elif(args[1] == "-"):
100                     calc = args[0] - args[2]
101                 elif(args[1] == "*"):
102                     calc = args[0] * args[2]
103                 else:
104                     print("undefined oprator in calc")
105                     exit(1)
106                 continue
107
108             if(args[2 * n + 1] == "+"):
109                 calc = calc + args[2 * n + 2]
110             elif(args[2 * n + 1] == "-"):
111                 calc = calc - args[2 * n + 2]
112             elif(args[2 * n + 1] == "*"):
113                 calc = calc * args[2 * n + 2]
114             else:
```

```
115                 print("undefined oprator in calc2")
116                 exit(1)
117         return calc
118
119     def checkRoute(self, *args):
120         """ルートが一筆書きにから逸脱していないか確かめる関数"""
121         prenum = 0 #一つ前に確認した場所
122         already = []
123         for n in range(len(args)):
124             #print("試行:", n)
125             if(args[n] < 1 or args[n] > self.cell * self.cell):
126                 print("枠をはみ出しています。")
127                 return False
128             #print(args[n],already)
129             if(args[n] in already):
130                 print("同じ道を通っています。")
131                 return False
132             if(n == 0):
133                 #もし偶数の盤面を作成する際はここの訂正を必要とする。
134                 if(args[0] % 2 == 0):
135                     print("演算子から始まっています。")
136                     return False
137                 prenum = args[0]
138                 already.append(args[n])
139                 continue
140             #縦横に動いているかチェック
141             #print(abs(args[n] - prenum))
142             if(abs(args[n] - prenum) == 1 or abs(args[n] - prenum) == self.cell):
143                 #setTarget の 16 行目-21 行目とやっていることは同じ。
144                 if(prenum % self.cell == 0):
145                     if(args[n] - prenum == 1):
146                         print("盤面の右端から左下に行っています。")
147                         return False
148                 if(prenum % self.cell == 1):
149                     if(args[n] - prenum == -1):
150                         print("盤面の左端から右上に行っています。")
151                         return False
152                 prenum = args[n]
153                 already.append(args[n])
154             else:
```

```
155                 print("移動先が不適です。")
156                 return False
157         return True
158
159     def setTarget(self, squares):
160         """ターゲットを決める関数"""
161         frag = True #break の時、外のコードが実行されてしまうためこれで制御する。
162         if(squares > self.cell * self.cell):
163             print("盤面のマス数を越える数の計算はできません")
164             exit(1)
165         #まれにどう動かしても負の数になりそうなので 50 回の試行で強制終了。
166         for n in range(50):
167             move = [] #マス目をどのように動かすかここに保存する。
168             self.FP = random.randrange(1, self.cell * self.cell, 2)
169             move.append(self.FP)
170             for m in range(squares - 1):
171                 frag = True
172                 choice = [2,4,6,8] #2:下、4:左、6:右、8:上
173                 while(len(choice) != 0):
174                     direction = choice.pop(random.randrange(0,len(choice),1))
175                     #1 番右の列の時に右に動こうとした時に再度数字を選ばせる。
176                     if(move[-1] % self.cell == 0 and direction == 6):
177                         continue
178                     #1 番左の列の時に左に動こうとした時に再度数字を選ばせる。
179                     if(move[-1] % self.cell == 1 and direction == 4):
180                         continue
181                     match direction:
182                         case 2:
183                             candidate = move[-1] + self.cell
184                         case 4:
185                             candidate = move[-1] - 1
186                         case 6:
187                             candidate = move[-1] + 1
188                         case 8:
189                             candidate = move[-1] - self.cell
190                         case _:
191                             print("ルート決定時にエラー")
192                             exit(1)
193                     #行動先が枠内で、すでに行ったところでなければ、行動先として保存する。
194                     if(candidate > 0 and candidate <= self.cell * self.cell
```

```
                        and candidate not in move):
                         move.append(candidate)
                         #print(move)
                         break
                #上下左右行動不可の場合 break して初期位置からやり直す
                else:
                    frag = False
                    break
            if(frag):
                equation = []
                for l in range(squares):
                    equation.append(self.board[move[l]])
                #print(equation)
                self.target = self.calcurate(*equation)
                #print(self.target)
                if(self.target > 0 and self.target < 100):
                    self.move = move
                    #print("target 生成成功")
                    return
                else:
                    pass
                    #print("target マイナスか 100 越え 再生成", self.target)
        #現状 50 回ループで出来なかったら実行を停止している。
        else:
            print("target ターゲット生成失敗。やり直してください。")
            exit(0)

    def setTarget2(self, squares):
        """若干の改良をしたターゲットを決める関数"""
        frag = True #break の時、外のコードが実行されてしまうためこれで制御する。
        if(squares > self.cell * self.cell):
            print("盤面のマス数を越える数の計算はできません")
            exit(1)
        #まれにどう動かしても負の数になりそうなので 100 回の試行で強制終了。(テスト中 20 に)
        for n in range(20):
            move = [] #マス目をどのように動かすかここに保存する。
            FPnumber = self.cell * self.cell #もし偶数盤面を作成した際に、範囲に影響するので変数を加える
            if(FPnumber % 2 == 0):
                FPnumber += 1
```

```
234             self.FP = random.randrange(1, FPnumber + 1, 2)
235             move.append(self.FP)
236             deletion = 0 #選択した方向の逆側をあらかじめ消しておく変数
237             for m in range(squares - 1):
238                 frag = True
239                 choice = [2,4,6,8] #2:下、4:左、6:右、8:上
240                 if(deletion != 0):
241                     choice.remove(deletion)
242                 while(len(choice) != 0):
243                     direction = choice.pop(random.randrange(0,len(choice),1))
244                     #1 番右の列の時に右に動こうとした時に再度数字を選ばせる。
245                     if(move[-1] % self.cell == 0 and direction == 6):
246                         continue
247                     #1 番左の列の時に左に動こうとした時に再度数字を選ばせる。
248                     if(move[-1] % self.cell == 1 and direction == 4):
249                         continue
250                     match direction:
251                         case 2:
252                             candidate = move[-1] + self.cell
253                             deletion = 8
254                         case 4:
255                             candidate = move[-1] - 1
256                             deletion = 6
257                         case 6:
258                             candidate = move[-1] + 1
259                             deletion = 4
260                         case 8:
261                             candidate = move[-1] - self.cell
262                             deletion = 2
263                         case _:
264                             print("ルート決定時にエラー")
265                             exit(1)
266                     #行動先が枠内で、すでに行ったところでなければ、行動先として保存する。
267                     if(candidate > 0 and candidate <= self.cell * self.cell
268                        and candidate not in move):
269                         move.append(candidate)
270                         #計算式の途中でも値の確認を行う。
271                         if(m % 2 == 1):
272                             preequation = []
273                             for l in range(m + 2):
```

```
                                preequation.append(self.board[move[l]])
                            prevalue = self.calcurate(*preequation)
                            if(prevalue < 0 and prevalue > 130):
                                move.pop(-1)
                                continue
                        break
                #上下左右行動不可の場合 break して初期位置からやり直す
                else:
                    #print("行き詰まり", move)
                    frag = False
                    break
            if(frag):
                self.target = prevalue
                if(self.target >= 10 and self.target < 100):
                    self.move = move
                    return
                else:
                    pass
                    #print("target 10 以下または、100 越え 再生成", self.target)
        #現状 100 回ループで出来なかったら実行を停止している。
        else:
            print("target ターゲット生成失敗。やり直してください。")
            exit(0)

    def customTarget(self,path):
        """自分でターゲットを指定するメソッド (テスト用)"""
        self.move = path[:]
        answer = []
        for l in range(len(path)):
            answer.append(self.board[path[l]])
        self.target = self.calcurate(*answer)
        #自分で変える
        self.minusVol = 3

    def getTarget(self):
        """ターゲットのゲッタ"""
        return self.target

    def getMove(self):
        """ムーブ (解答) のゲッタ"""
```

```
314             return self.move
315
316         def getBoard(self):
317             """盤面のゲッタ"""
318             return self.board
319
320         def getBoard2(self, num):
321             """盤面中指定した番号の値を取ってくれるメソッド"""
322             return self.board[num]
323
324         def getnumbers(self):
325             """盤面の数字だけを返すもの"""
326             return self.numbers
327
328         def getMinus(self):
329             """マイナスの数を返すメソッド"""
330             return self.minusVol
331
332         def getPlus(self):
333             """プラスの数を返すメソッド"""
334             return self.plusVol
335
336         def answerCheck(self, *args):
337             """引数にルートを取り、その結果が target に一致しているか確かめる関数"""
338             answer = []
339             for l in range(len(args)):
340                 answer.append(self.board[args[l]])
341             self.calc = self.calcurate(*answer)
342             if(self.calc == self.target):
343                 print("correct")
344                 return True
345             else: #ここの print 文は, 実験時には出力文が長くなるためコメントアウトする.
346                 print("your answer =", self.calc, " not correct")
347                 print("answer is ", self.move)
348                 return False
349
350         def answerCheck2(self, ans, *args):
351             """引数に必要値とルートを取り、その結果が必要値に一致しているか確かめる関数"""
352             answer = []
353             for l in range(len(args)):
```

```
354                 answer.append(self.board[args[l]])
355             self.calc = self.calcurate(*answer)
356             if(self.calc == ans):
357                 print("correct")
358                 return True
359             else:
360                 #print("your answer =", self.calc, " not correct")
361                 #print("answer is ", self.move)
362                 return False
363 
364     def main():
365         print("盤面のサイズを指定してください。(奇数限定)")
366         cell = input()
367         board = Board(int(cell))
368         print("難易度を設定してください。(E/N/H)")
369         diff = input()
370         match diff:
371             case "E":
372                squares = random.randrange(3,6,2)
373             case "N":
374               squares = random.randrange(5,8,2)
375             case "H":
376                squares = random.randrange(7,10,2)
377             case "Ex":
378               squares = 9
379             case _:
380                 print("normal で実行します。")
381                 squares = random.randrange(5,8,2)
382         board.createBoard2()
383         board.dpBoard()
384         board.setTarget2(squares)
385         print("target: ", board.getTarget())
386         print("答えを入力してください (1,2,3... といった形式で)")
387         while True:
388             answer = input()
389             number = answer.split(",")
390             ans = []
391             for n in range(len(number)):
392                 ans.append(int(number[n]))
393             if(board.checkRoute(*ans)):
```

```
394                 break
395         board.answerCheck(*ans)
```

tool.py を示す.

```
1   class tool:
2       """道具を置いとくクラス"""
3       def __init__(self,cell):
4           """コンストラクタ"""
5           self.cls = -1 #クラス
6           self.cell = int(cell) #縦横の数
7           self.log = [] #通った場所を明白にするためのログ
8           self.pLog = [] #枝刈りなどで行けないルートのログを消さずに残すためのもの
9
10      def classification(self, num):
11          """引数からその番号の位置を定めて返すメソッド"""
12          #左上の時
13          if(num == 1):
14              return 1
15          #右上の時
16          elif(num == self.cell):
17              return 2
18          #左下の時
19          elif(num == self.cell * (self.cell - 1) + 1):
20              return 3
21          #右下の時
22          elif(num == self.cell * self.cell):
23              return 4
24          #上段の時
25          elif(num < self.cell):
26              return 5
27          #下段の時
28          elif(num > self.cell * (self.cell - 1)):
29              return 6
30          #左行の時
31          elif(num % self.cell == 1):
32              return 7
33          #右行の時
34          elif(num % self.cell == 0):
35              return 8
36          #それ以外（真ん中らへん）のとき
```

```
37          else:
38              return 9
39
40      def numGenerator(self, path2):
41          """数字を生成するメソッド"""
42          #以下 3 行必要か不明。(色がつかないので)
43          path = []
44          for n in path2:
45              path.append(n)
46          num = path[-1]
47          self.cls = self.classification(num)
48          match self.cls:
49              case 1: #左上の時
50                  if(2 not in path):
51                      if(3 not in path): #右右
52                          path.append(2)
53                          path.append(3)
54                          if(path in self.log):
55                              del path[-2:]
56                          else:
57                              return path
58                      if(2 + self.cell not in path): #右下
59                          path.append(2)
60                          path.append(2 + self.cell)
61                          if(path in self.log):
62                              del path[-2:]
63                          else:
64                              return path
65
66                  if(1 + self.cell not in path):
67                      if(2 + self.cell not in path): #下右
68                          path.append(1 + self.cell)
69                          path.append(2 + self.cell)
70                          if(path in self.log):
71                              del path[-2:]
72                          else:
73                              return path
74                      if(1 + (self.cell * 2) not in path): #下下
75                          path.append(1 + self.cell)
76                          path.append(1 + (self.cell * 2))
```

```
77                          if(path in self.log):
78                              del path[-2:]
79                          else:
80                              return path
81                  return path
82
83              case 2: #右上の時
84                  if(self.cell - 1 not in path): #式が簡単になるので、cell を使用
85                      if(self.cell - 2 not in path): #左左
86                          path.append(self.cell - 1)
87                          path.append(self.cell - 2)
88                          if(path in self.log):
89                              del path[-2:]
90                          else:
91                              return path
92                      if((self.cell * 2) - 1  not in path): #左下
93                          path.append(self.cell - 1)
94                          path.append((self.cell * 2) - 1)
95                          if(path in self.log):
96                              del path[-2:]
97                          else:
98                              return path
99
100                 if(self.cell * 2 not in path):
101                     if(self.cell * 2 - 1 not in path): #下左
102                         path.append(self.cell * 2)
103                         path.append(self.cell * 2 - 1)
104                         if(path in self.log):
105                             del path[-2:]
106                         else:
107                             return path
108                     if(self.cell * 3 not in path): #下下
109                         path.append(self.cell * 2)
110                         path.append(self.cell * 3)
111                         if(path in self.log):
112                             del path[-2:]
113                         else:
114                             return path
115                 return path
116
```

```
117                 case 3: #左下の時
118                     if(num + 1 not in path):
119                         if(num + 2 not in path): #右右
120                             path.append(num + 1)
121                             path.append(num + 2)
122                             if(path in self.log):
123                                 del path[-2:]
124                             else:
125                                 return path
126                         if(num + 1 - self.cell not in path): #右上
127                             path.append(num + 1)
128                             path.append(num + 1 - self.cell)
129                             if(path in self.log):
130                                 del path[-2:]
131                             else:
132                                 return path
133 
134                     if(num - self.cell not in path):
135                         if(num - self.cell + 1 not in path): #上右
136                             path.append(num - self.cell)
137                             path.append(num - self.cell + 1)
138                             if(path in self.log):
139                                 del path[-2:]
140                             else:
141                                 return path
142                         if(num - (self.cell * 2) not in path): #上上
143                             path.append(num - self.cell)
144                             path.append(num - (self.cell * 2))
145                             if(path in self.log):
146                                 del path[-2:]
147                             else:
148                                 return path
149                     return path
150 
151                 case 4: #右下の時
152                     if(num - 1 not in path):
153                         if(num - 2 not in path): #左左
154                             path.append(num - 1)
155                             path.append(num - 2)
156                             if(path in self.log):
```

```
157                             del path[-2:]
158                         else:
159                             return path
160                     if(num - 1 - self.cell not in path): #左上
161                         path.append(num - 1)
162                         path.append(num - 1 - self.cell)
163                         if(path in self.log):
164                             del path[-2:]
165                         else:
166                             return path
167 
168                 if(num - self.cell not in path):
169                     if(num - self.cell - 1 not in path): #上左
170                         path.append(num - self.cell)
171                         path.append(num - self.cell - 1)
172                         if(path in self.log):
173                             del path[-2:]
174                         else:
175                             return path
176                     if(num - (self.cell * 2) not in path): #上上
177                         path.append(num - self.cell)
178                         path.append(num - (self.cell * 2))
179                         if(path in self.log):
180                             del path[-2:]
181                         else:
182                             return path
183                 return path
184 
185             case 5: #上行の時
186                 if(num - 1 not in path):
187                     if(num - 2 not in path): #左左
188                         path.append(num - 1)
189                         path.append(num - 2)
190                         if(path in self.log):
191                             del path[-2:]
192                         else:
193                             return path
194                     if(num - 1 + self.cell not in path): #左下
195                         path.append(num - 1)
196                         path.append(num - 1 + self.cell)
```

```
197                         if(path in self.log):
198                             del path[-2:]
199                         else:
200                             return path
201 
202                 if(num + 1 not in path):
203                     if(num + 2 not in path): #右右
204                         path.append(num + 1)
205                         path.append(num + 2)
206                         if(path in self.log):
207                             del path[-2:]
208                         else:
209                             return path
210                     if(num + 1 + self.cell not in path): #右下
211                         path.append(num + 1)
212                         path.append(num + 1 + self.cell)
213                         if(path in self.log):
214                             del path[-2:]
215                         else:
216                             return path
217 
218                 if(num + self.cell not in path):
219                     if(num + self.cell - 1 not in path): #下左
220                         path.append(num + self.cell)
221                         path.append(num + self.cell - 1)
222                         if(path in self.log):
223                             del path[-2:]
224                         else:
225                             return path
226                     if(num + self.cell + 1 not in path): #下右
227                         path.append(num + self.cell)
228                         path.append(num + self.cell + 1)
229                         if(path in self.log):
230                             del path[-2:]
231                         else:
232                             return path
233                     if(num + (self.cell * 2) not in path): #下下
234                         path.append(num + self.cell)
235                         path.append(num + (self.cell * 2))
236                         if(path in self.log):
```

```
                            del path[-2:]
                        else:
                            return path
                return path

            case 6: #下行の時
                if(num - 1 not in path):
                    if(num - 2 not in path): #左左
                        path.append(num - 1)
                        path.append(num - 2)
                        if(path in self.log):
                            del path[-2:]
                        else:
                            return path
                    if(num - 1 - self.cell not in path): #左上
                        path.append(num - 1)
                        path.append(num - 1 - self.cell)
                        if(path in self.log):
                            del path[-2:]
                        else:
                            return path

                if(num + 1 not in path):
                    if(num + 2 not in path): #右右
                        path.append(num + 1)
                        path.append(num + 2)
                        if(path in self.log):
                            del path[-2:]
                        else:
                            return path
                    if(num + 1 - self.cell not in path): #右上
                        path.append(num + 1)
                        path.append(num + 1 - self.cell)
                        if(path in self.log):
                            del path[-2:]
                        else:
                            return path

                if(num - self.cell not in path):
                    if(num - self.cell - 1 not in path): #上左
```

```
277                         path.append(num - self.cell)
278                         path.append(num - self.cell - 1)
279                         if(path in self.log):
280                             del path[-2:]
281                         else:
282                             return path
283                     if(num - self.cell + 1 not in path): #上右
284                         path.append(num - self.cell)
285                         path.append(num - self.cell + 1)
286                         if(path in self.log):
287                             del path[-2:]
288                         else:
289                             return path
290                     if(num - (self.cell * 2) not in path): #上上
291                         path.append(num - self.cell)
292                         path.append(num - (self.cell * 2))
293                         if(path in self.log):
294                             del path[-2:]
295                         else:
296                             return path
297                 return path
298 
299             case 7: #左列の時
300                 if(num - self.cell not in path):
301                     if(num - (self.cell * 2) not in path): #上上
302                         path.append(num - self.cell)
303                         path.append(num - (self.cell * 2))
304                         if(path in self.log):
305                             del path[-2:]
306                         else:
307                             return path
308                     if(num - self.cell + 1 not in path): #上右
309                         path.append(num - self.cell)
310                         path.append(num - self.cell + 1)
311                         if(path in self.log):
312                             del path[-2:]
313                         else:
314                             return path
315 
316                 if(num + self.cell not in path):
```

```
317                     if(num + (self.cell * 2) not in path): #下下
318                         path.append(num + self.cell)
319                         path.append(num + (self.cell * 2))
320                         if(path in self.log):
321                             del path[-2:]
322                         else:
323                             return path
324                     if(num + self.cell + 1 not in path): #下右
325                         path.append(num + self.cell)
326                         path.append(num + self.cell + 1)
327                         if(path in self.log):
328                             del path[-2:]
329                         else:
330                             return path
331 
332                 if(num + 1 not in path):
333                     if(num + 1 - self.cell not in path): #右上
334                         path.append(num + 1)
335                         path.append(num + 1 - self.cell)
336                         if(path in self.log):
337                             del path[-2:]
338                         else:
339                             return path
340                     if(num + 1 + self.cell not in path): #右下
341                         path.append(num + 1)
342                         path.append(num + 1 + self.cell)
343                         if(path in self.log):
344                             del path[-2:]
345                         else:
346                             return path
347                     if(num + 2 not in path): #右右
348                         path.append(num + 1)
349                         path.append(num + 2)
350                         if(path in self.log):
351                             del path[-2:]
352                         else:
353                             return path
354                 return path
355 
356             case 8: #右列の時
```

```
357                 if(num - self.cell not in path):
358                     if(num - (self.cell * 2) not in path): #上上
359                         path.append(num - self.cell)
360                         path.append(num - (self.cell * 2))
361                         if(path in self.log):
362                             del path[-2:]
363                         else:
364                             return path
365                     if(num - self.cell - 1 not in path): #上左
366                         path.append(num - self.cell)
367                         path.append(num - self.cell - 1)
368                         if(path in self.log):
369                             del path[-2:]
370                         else:
371                             return path
372 
373                 if(num + self.cell not in path):
374                     if(num + (self.cell * 2) not in path): #下下
375                         path.append(num + self.cell)
376                         path.append(num + (self.cell * 2))
377                         if(path in self.log):
378                             del path[-2:]
379                         else:
380                             return path
381                     if(num + self.cell - 1 not in path): #下左
382                         path.append(num + self.cell)
383                         path.append(num + self.cell - 1)
384                         if(path in self.log):
385                             del path[-2:]
386                         else:
387                             return path
388 
389                 if(num - 1 not in path):
390                     if(num - 1 - self.cell not in path): #左上
391                         path.append(num - 1)
392                         path.append(num - 1 - self.cell)
393                         if(path in self.log):
394                             del path[-2:]
395                         else:
396                             return path
```

```
397                     if(num - 1 + self.cell not in path): #左下
398                         path.append(num - 1)
399                         path.append(num - 1 + self.cell)
400                         if(path in self.log):
401                             del path[-2:]
402                         else:
403                             return path
404                     if(num - 2 not in path): #左左
405                         path.append(num - 1)
406                         path.append(num - 2)
407                         if(path in self.log):
408                             del path[-2:]
409                         else:
410                             return path
411                 return path
412 
413             case 9: #それ以外の時
414                 if(num - self.cell not in path):
415                     if(num - self.cell - 1 not in path): #上左
416                         path.append(num - self.cell)
417                         path.append(num - self.cell - 1)
418                         if(path in self.log):
419                             del path[-2:]
420                         else:
421                             return path
422                     if(num - self.cell + 1 not in path): #上右
423                         path.append(num - self.cell)
424                         path.append(num - self.cell + 1)
425                         if(path in self.log):
426                             del path[-2:]
427                         else:
428                             return path
429                     if(num - (self.cell * 2) > 0
430                        and num - (self.cell * 2) not in path): #上上
431                         path.append(num - self.cell)
432                         path.append(num - (self.cell * 2))
433                         if(path in self.log):
434                             del path[-2:]
435                         else:
436                             return path
```

```
437
438                 if(num + self.cell not in path):
439                     if(num + self.cell - 1 not in path): #下左
440                         path.append(num + self.cell)
441                         path.append(num + self.cell - 1)
442                         if(path in self.log):
443                             del path[-2:]
444                         else:
445                             return path
446                     if(num + self.cell + 1 not in path): #下右
447                         path.append(num + self.cell)
448                         path.append(num + self.cell + 1)
449                         if(path in self.log):
450                             del path[-2:]
451                         else:
452                             return path
453                     if(num + (self.cell * 2) <= self.cell ** 2
454                        and num + (self.cell * 2) not in path): #下下
455                         path.append(num + self.cell)
456                         path.append(num + (self.cell * 2))
457                         if(path in self.log):
458                             del path[-2:]
459                         else:
460                             return path
461
462                 if(num + 1 not in path):
463                     if(num + 1 - self.cell not in path): #右上
464                         path.append(num + 1)
465                         path.append(num + 1 - self.cell)
466                         if(path in self.log):
467                             del path[-2:]
468                         else:
469                             return path
470                     if(num + 1 + self.cell not in path): #右下
471                         path.append(num + 1)
472                         path.append(num + 1 + self.cell)
473                         if(path in self.log):
474                             del path[-2:]
475                         else:
476                             return path
```

```
477                     if(num % self.cell != (self.cell - 1)
478                        and num + 2 not in path): #右右
479                         path.append(num + 1)
480                         path.append(num + 2)
481                         if(path in self.log):
482                             del path[-2:]
483                         else:
484                             return path
485
486                 if(num - 1 not in path):
487                     if(num - 1 - self.cell not in path): #左上
488                         path.append(num - 1)
489                         path.append(num - 1 - self.cell)
490                         if(path in self.log):
491                             del path[-2:]
492                         else:
493                             return path
494                     if(num - 1 + self.cell not in path): #左下
495                         path.append(num - 1)
496                         path.append(num - 1 + self.cell)
497                         if(path in self.log):
498                             del path[-2:]
499                         else:
500                             return path
501                     if(num % self.cell != 2 and num - 2 not in path): #左左
502                         path.append(num - 1)
503                         path.append(num - 2)
504                         if(path in self.log):
505                             del path[-2:]
506                         else:
507                             return path
508                 return path
509             case _:
510                 print("classification error")
511                 exit(1)
512
513     def registerLog(self, log):
514         """ログを登録するメソッド"""
515         self.log.append(log)
516
```

```
517         def registerpLog(self, plog):
518             """消さないログを登録するメソッド"""
519             self.pLog.append(plog)
520 
521         def clearLog(self):
522             """ログをクリアするメソッド"""
523             self.log = self.pLog[:]
524 
525         def getpLog(self):
526             """消さないログのゲッタ"""
527             return self.pLog
528 
```

brute.py を示す

```
 1  import eq2 # type: ignore
 2  import tool # type: ignore
 3  import random
 4 
 5  print("cell num?")
 6  cell = int(input())
 7  board = eq2.Board(cell)
 8  board.createBoard2()
 9  board.dpBoard()
10  randNum = random.choice([5,7,9])
11  board.setTarget2(randNum)
12  print("target: ", board.getTarget())
13  tol = tool.tool(cell)
14  tryn = 1 #試行回数
15  flag = True #次の数字に遷移するためのもの
16  path = [] #道の生成
17  path2 = [] #委譲先で返ってきた道をここに代入する。
18  copy = [] #ポインタ参照を回避するためにここに複製する
19  for n in range(int(cell * cell / 2 + 1)): #演算対象の数
20      for m in range(1,cell * cell + 1, 2): #初期位置の決定
21          flag = True
22          while(flag): #フラグがあるうちは初期値は変わらない。
23              path.append(m)
24              l = 0 #現在の式の長さを示すカウンタ用変数
25              while(l < n):
26                  path2 = tol.numGenerator(path)
```

```
27                  #経路があった場合.
28                  if(path != path2):
29                      path = path2[:]
30                      l += 1
31                      continue
32                  #経路がなく行き詰まった場合
33                  else:
34                      #長さが 1 の時は、探索打ち止め。
35                      if(len(path) == 1):
36                          path = []
37                          flag = False
38                          break
39                      #長さが 1 より大きい場合は、ログを取って巻き戻す.
40                      else:
41                          copy = []
42                          for k in path:
43                              copy.append(k)
44                          tol.registerLog(copy)
45                          del path[-2:]
46                          l -= 1
47                          continue
48              if(len(path) != 0):
49                  #このコメントアウトを外すと, デバック用に探索した経路が表示される.
50                  #print(path)
51                  if(board.answerCheck(*path)): #答え合わせする。
52                      print("Answer:",end="")
53                      print(path)
54                      print("try is " + str(tryn))
55                      exit(0)
56                  tol.registerLog(path)
57                  tryn += 1 #答え合わせをしたタイミングで増やす。
58              path = []
59              #n が 0 の時長さが 1 で終わらせるので、flag を切る。
60              if(n == 0):
61                  flag = False
62              #無限ループ防止。
63              if(tryn == 10000):
64                  print("規定回数を超えました。")
65                  exit(1)
66      tol.clearLog()
```

impbrute.py を示す

```
1  import eq2 # type: ignore
2  import tool # type: ignore
3  import random
4
5  print("cell num?")
6  cell = int(input())
7  board = eq2.Board(cell)
8  board.createBoard2()
9  board.dpBoard()
10 randNum = random.choice([5,7,9])
11 board.setTarget2(randNum)
12 print("target: ", board.getTarget())
13 tol = tool.tool(cell)
14 tryn = 1 #試行回数
15 flag = True #次の数字に遷移するためのもの
16 path = [] #道の生成
17 path2 = [] #委譲先で返ってきた道をここに代入する。
18 copy = [] #ポインタ参照を回避するためにここに複製する
19 upperpruning = 0 #枝刈りの上界値
20 pruningNum = 0 #枝刈り
21 for n in range(1, board.getMinus() + 1):
22     upperpruning += sorted(board.getnumbers())[-n]
23 upperpruning += board.getTarget()
24 for n in range(1, int(cell * cell / 2 + 1)): #演算対象の数
25     for m in range(1,cell * cell + 1, 2): #初期位置の決定
26         flag = True
27         while(flag):
28             path.append(m)
29             l = 0
30             while(l < n):
31                 path2 = tol.numGenerator(path)
32                 if(path != path2):
33                     path = path2[:]
34                     l += 1
35                     continue
36                 else:
37                     #長さが 1 の時は、探索打ち止め。
38                     if(len(path) == 1):
```

```
39                          path = []
40                          flag = False
41                          break
42                      #長さが 1 より大きい場合は、
43                      else:
44                          copy = []
45                          for k in path:
46                              copy.append(k)
47                          tol.registerLog(copy)
48                          del path[-2:]
49                          l -= 1
50                          continue
51              if(len(path) != 0):
52                  #print(path)
53                  if(board.answerCheck(*path)): #答え合わせする。
54                      print("Answer:",end="")
55                      print(path)
56                      print("try is " + str(tryn))
57                      print("枝刈りの回数:",end="")
58                      print(pruningNum)
59                      exit(0)
60                  #枝刈りをする
61                  answer = []
62                  for l in range(len(path)):
63                      answer.append(board.getBoard2(path[l]))
64                  if(board.calcurate(*answer) > upperpruning
65                     or board.calcurate(*answer) < 0):
66                      tol.registerpLog(path)
67                      #print("e:",path)
68                      pruningNum += 1
69                  tol.registerLog(path)
70                  tryn += 1 #答え合わせをしたタイミングで増やす。
71              path = []
72              #n が 0 の時長さが 1 で終わらせるので、flag を切る。
73              if(n == 0):
74                  flag = False
75              #無限ループ防止。
76              if(tryn == 10000):
77                  print("規定回数を超えました。")
78                  exit(1)
```

```
79      tol.clearLog()
```

mysolv.py を示す.

```
1   import eq2 # type: ignore
2   import tool # type: ignore
3   import random
4
5   print("cell = ", end="")
6   cell = int(input())
7   board = eq2.Board(cell)
8   tol = tool.tool(cell)
9   board.createBoard2()
10  board.dpBoard()
11  randNum = random.choice([5,7,9])
12  board.setTarget2(randNum)
13  print("target: ", board.getTarget())
14  factor = [] #約数を格納する。
15  path = [] #道の生成
16  path2 = [] #委譲先で返ってきた道をここに代入する。
17  subTarget = -1 #一部分を除いた新しいターゲット
18  tryn = 1 #試行回数
19  pruningNum = 0 #枝刈りした回数
20  upperpruning = 0 #枝刈りの上界
21  lowerpruning = 0 #枝刈りの下界
22
23  def lastMulCheck(lastPos):
24      """約数があった際にまわりに掛け算記号があるか確認する。"""
25      Pos = tol.classification(lastPos)
26      conform = [] #返り値のための変数
27      match Pos:
28          case 1: #左上
29              if(board.getBoard2(2) == '*'):
30                  conform.append([1, 2, 3])
31                  conform.append([1, 2, 2 + cell])
32              if(board.getBoard2(1 + cell) == '*'):
33                  conform.append([1, 1 + cell, 2 + cell])
34                  conform.append([1, 1 + cell, 1 + (cell * 2)])
35              return conform
36
37          case 2: #右上
```

```
38              if(board.getBoard2(lastPos - 1) == '*'):
39                  conform.append([lastPos, lastPos - 1, lastPos - 2])
40                  conform.append([lastPos, lastPos - 1, lastPos - 1 + cell])
41              if(board.getBoard2(lastPos + cell) == '*'):
42                  conform.append([lastPos, lastPos + cell, lastPos + (cell * 2)])
43                  conform.append([lastPos, lastPos + cell, lastPos + cell - 1])
44              return conform
45
46          case 3: #左下
47              if(board.getBoard2(lastPos - cell) == '*'):
48                  conform.append([lastPos, lastPos - cell, lastPos - (cell * 2)])
49                  conform.append([lastPos, lastPos - cell, lastPos - cell + 1])
50              if(board.getBoard2(lastPos + 1) == '*'):
51                  conform.append([lastPos, lastPos + 1, lastPos + 2])
52                  conform.append([lastPos, lastPos + 1, lastPos + 1 - cell])
53              return conform
54
55          case 4: #右下
56              if(board.getBoard2(lastPos - cell) == '*'):
57                  conform.append([lastPos, lastPos - cell, lastPos - (cell * 2)])
58                  conform.append([lastPos, lastPos - cell, lastPos - cell - 1])
59              if(board.getBoard2(lastPos - 1) == '*'):
60                  conform.append([lastPos, lastPos - 1, lastPos - 2])
61                  conform.append([lastPos, lastPos - 1, lastPos - 1 - cell])
62              return conform
63
64          case 5: #上行
65              if(board.getBoard2(lastPos - 1) == '*'):
66                  conform.append([lastPos, lastPos - 1, lastPos - 2])
67                  conform.append([lastPos, lastPos - 1, lastPos - 1 + cell])
68              if(board.getBoard2(lastPos + 1) == '*'):
69                  conform.append([lastPos, lastPos + 1, lastPos + 2])
70                  conform.append([lastPos, lastPos + 1, lastPos + 1 + cell])
71              if(board.getBoard2(lastPos + cell) == '*'):
72                  conform.append([lastPos, lastPos + cell, lastPos + (cell * 2)])
73                  conform.append([lastPos, lastPos + cell, lastPos + cell + 1])
74                  conform.append([lastPos, lastPos + cell, lastPos + cell - 1])
75              return conform
76
77          case 6: #下行
```

```
78              if(board.getBoard2(lastPos - 1) == '*'):
79                  conform.append([lastPos, lastPos - 1, lastPos - 2])
80                  conform.append([lastPos, lastPos - 1, lastPos - 1 - cell])
81              if(board.getBoard2(lastPos + 1) == '*'):
82                  conform.append([lastPos, lastPos + 1, lastPos + 2])
83                  conform.append([lastPos, lastPos + 1, lastPos + 1 -cell])
84              if(board.getBoard2(lastPos - cell) == '*'):
85                  conform.append([lastPos, lastPos - cell, lastPos - (cell * 2)])
86                  conform.append([lastPos, lastPos - cell, lastPos - cell - 1])
87                  conform.append([lastPos, lastPos - cell, lastPos - cell + 1])
88              return conform
89
90          case 7: #左列
91              if(board.getBoard2(lastPos - cell) == '*'):
92                  conform.append([lastPos, lastPos - cell, lastPos - (cell * 2)])
93                  conform.append([lastPos, lastPos - cell, lastPos - cell + 1])
94              if(board.getBoard2(lastPos + cell) == '*'):
95                  conform.append([lastPos, lastPos + cell, lastPos + (cell * 2)])
96                  conform.append([lastPos, lastPos + cell, lastPos + cell + 1])
97              if(board.getBoard2(lastPos + 1) == '*'):
98                  conform.append([lastPos, lastPos + 1,lastPos + 2])
99                  conform.append([lastPos, lastPos + 1,lastPos + 1 - cell])
100                 conform.append([lastPos, lastPos + 1,lastPos + 1 + cell])
101             return conform
102
103         case 8: #右列
104             if(board.getBoard2(lastPos - cell) == '*'):
105                 conform.append([lastPos, lastPos - cell, lastPos - (cell * 2)])
106                 conform.append([lastPos, lastPos - cell, lastPos - cell - 1])
107             if(board.getBoard2(lastPos + cell) == '*'):
108                 conform.append([lastPos, lastPos + cell, lastPos + (cell * 2)])
109                 conform.append([lastPos, lastPos + cell, lastPos + cell - 1])
110             if(board.getBoard2(lastPos - 1) == '*'):
111                 conform.append([lastPos, lastPos - 1, lastPos - 2])
112                 conform.append([lastPos, lastPos - 1, lastPos - 1 - cell])
113                 conform.append([lastPos, lastPos - 1, lastPos - 1 + cell])
114             return conform
115
116         case 9: #その他
117             if(board.getBoard2(lastPos - cell) == '*'):
```

```
118                     if(lastPos - cell >= cell):
119                         conform.append([lastPos, lastPos - cell, lastPos - (cell * 2)])
120                     conform.append([lastPos, lastPos - cell, lastPos - cell - 1])
121                     conform.append([lastPos, lastPos - cell, lastPos - cell + 1])
122                 if(board.getBoard2(lastPos + cell) == '*'):
123                     if(lastPos + (cell * 2) <= cell * cell):
124                         conform.append([lastPos, lastPos + cell, (lastPos + cell * 2)])
125                     conform.append([lastPos, lastPos + cell, lastPos + cell - 1])
126                     conform.append([lastPos, lastPos + cell, lastPos + cell + 1])
127                 if(board.getBoard2(lastPos - 1) == '*'):
128                     if((lastPos - 1) % cell != 1):
129                         conform.append([lastPos, lastPos - 1, lastPos - 2])
130                     conform.append([lastPos, lastPos - 1, lastPos - 1 - cell])
131                     conform.append([lastPos, lastPos - 1, lastPos - 1 + cell])
132                 if(board.getBoard2(lastPos + 1) == '*'):
133                     if((lastPos + 1) % cell != 0):
134                         conform.append([lastPos, lastPos + 1, lastPos + 2])
135                     conform.append([lastPos, lastPos + 1, lastPos + 1 - cell])
136                     conform.append([lastPos, lastPos + 1, lastPos + 1 + cell])
137                 return conform
138 
139             #デバッグ用
140             case _:
141                 print("error in Pos(classification)")
142                 exit(1)
143 
144     def fluctuation(cPath):
145         """増減させて答えか確かめる関数"""
146         global tryn
147         global pruningNum
148         path = cPath[:]
149         for t in range(int((cell * cell - len(cPath)) / 2)): #演算対象の数
150             flag = True
151             while(flag):
152                 s = -1 #そのままの数値が答えであることはないので、-1 スタートとする。
153                 while(s < t): #現在の演算対象の数だけ経路選択をする
154                     #print(path)
155                     path2 = tol.numGenerator(path)
156                     #1 をかける場合を除いて、最初から続いて掛け算になる際は別の処理に任せるので、
```

ここでは排除する。

```
157                 if(s == -1 and path2[-2] == "*" and path2[-1] != 1):
158                     tol.registerLog(path2)
159                     break
160                 #未探索の経路があった場合
161                 if(path != path2):
162                     path = path2[:]
163                     s += 1
164                     continue
165                 #新たに経路を得られなかった場合
166                 else:
167                     #長さが 3 の時は、探索打ち止め。
168                     if(len(path) == 3):
169                         #del path[-3:]
170                         flag = False
171                         break
172                     #長さが 3 より大きい場合は、ログを取って、巻き戻して続行。
173                     else:
174                         copy = path[:]
175                         tol.registerLog(copy)
176                         #log.append(copy)
177                         del path[-2:]
178                         s -= 1
179                         continue
180             #現在の t の演算対象分が終わった時にここを通るので、条件を加えておく。
181             if(len(path) > 3):
182                 #解答チェック
183                 if(board.answerCheck(*path2)):
184                     print("Answer:",end="")
185                     print(path)
186                     print("try is " + str(tryn))
187                     print("枝刈りの回数:",end="")
188                     print(pruningNum)
189                     #exit(0)
190                     quit(0)
191                 #枝刈りをする
192                 answer = []
193                 for l in range(len(path)):
194                     answer.append(board.getBoard2(path[l]))
195                 if(board.calcurate(*answer) > upperpruning):
196                     tol.registerpLog(path)
```

```
197                         pruningNum += 1
198                     tol.registerLog(path)
199                     path = path[:3]
200                     tryn += 1
201                 #デバッグ用無限ループ防止。
202                 if(tryn == 1000000):
203                     print("規定回数を超えました。")
204                     exit(1)
205             tol.clearLog()
206 
207     def MulMul(value, root):
208         """数値とルートを指定し、範囲内なら答えを確かめ、低ければ、再帰処理を行う。"""
209         check = lastMulCheck(root[-1]) #root の最後のマスから掛け算記号を探す。
210         for n in check:
211             if(n[-1] in root or n[-2] in root):
212                 continue
213             ans = value * board.getBoard2(n[-1])
214             if(ans < upperpruning):
215                 root2 = root[:] + n[-2:]
216                 if(ans > lowerpruning):
217                     fluctuation(root2)
218                 MulMul(ans, root2)
219 
220 
221     def difMultiple(pos1, pos2, pos3):
222         """場所を指定して掛け算し、枝刈りされない範囲なら答えを確かめ、数値が低い場合は再度もう一
    度掛けさせる関数"""
223         result = board.getBoard2(pos1) * board.getBoard2(pos2)
224         if(result < board.getTarget() + upperpruning):
225             if(result > board.getTarget() - lowerpruning):
226                 fluctuation([pos1, pos3, pos2])
227                 fluctuation([pos2, pos3, pos1])
228             MulMul(result,[pos1, pos3, pos2])
229             MulMul(result,[pos2, pos3, pos1])
230         else:
231             return -1
232 
233     def MulType(n, *args):
234         """掛け算できる候補を列挙しておき、引数でもらった場所をあてに difMultiple に飛ばし真偽を
    もとに返り値を送る"""
```

```
235         #上左
236         if(1 in args):
237             difMultiple(n - cell, n - 1, n)
238         #上右
239         if(2 in args):
240             difMultiple(n - cell, n + 1, n)
241         #上下
242         if(3 in args):
243             difMultiple(n - cell, n + cell, n)
244         #左下
245         if(4 in args):
246             difMultiple(n - 1, n + cell, n)
247         #左右
248         if(5 in args):
249             difMultiple(n - 1, n + 1, n)
250         #下右
251         if(6 in args):
252             difMultiple(n + cell, n + 1, n)
253         #return result
254 
255     #約数抽出
256     for n in range(2, 10):
257         if(board.getTarget() % n == 0):
258             factor.append(n)
259 
260     #最後が掛け算で終わる際の探索（逆に探索してるので枝刈りはできない。）
261     for n in range(len(factor)):
262         last = factor.pop()
263         for m in range(1, cell * cell + 1, 2):
264             #約数と一致する際
265             if(board.getBoard2(m) == last):
266                 subTarget = int(board.getTarget() / last)
267                 lastPaths = lastMulCheck(m)
268                 if(len(lastPaths) == 0):
269                     continue
270                 for l in lastPaths:
271                     path = l[:]
272                     for t in range(int(cell * cell / 2)): #演算対象の数
273                         flag = True
274                         while(flag):
```

```
275                         s = 0
276                         while(s < t): #現在の演算対象の数だけ経路選択をする
277                             #print(path)
278                             path2 = tol.numGenerator(path)
279                             #未探索の経路があった場合
280                             if(path != path2):
281                                 path = path2[:]
282                                 s += 1
283                                 continue
284                             #新たに経路を得られなかった場合
285                             else:
286                                 #長さが 3 の時は、探索打ち止め。
287                                 if(len(path) == 3):
288                                     flag = False
289                                     break
290                                 #長さが 1 より大きい場合は、ログを取って、巻き戻して続行。
291                                 else:
292                                     copy = path[:]
293                                     tol.registerLog(copy)
294                                     del path[-2:]
295                                     s -= 1
296                                     continue
297                         #t=0 の時 path2 が生成されないので別処理にする。
298                         if(t == 0):
299                             #解答チェック
300                             if(board.answerCheck(*path)):
301                                 print("Answer:",end="")
302                                 print(path)
303                                 print("try is " + str(tryn))
304                                 print("枝刈りの回数:",end="")
305                                 print(pruningNum)
306                                 #exit(0)
307                                 quit()
308                             tryn += 1
309                         #現在の t の演算対象分が終わった時にここを通るので、条件を加えておく。
310                         if(len(path) > 3):
311                             path2.reverse()
312                             #解答チェック 2
313                             if(board.answerCheck2(subTarget, *path2[:-2])):
```

```
                                    print("Answer:",end="")
                                    print(path2)
                                    print("try is " + str(tryn))
                                    print("枝刈りの回数:",end="")
                                    print(pruningNum)
                                    exit(0)
                                tol.registerLog(path)
                                path = path[:3]
                                tryn += 1
                            #t が 0 の時長さが 1 で終わらせるので、flag を切る。
                            if(t == 0):
                                flag = False
                            #デバッグ用無限ループ防止。
                            if(tryn == 1000000):
                                print("規定回数を超えました。")
                                exit(1)
                        #log = []
                        tol.clearLog()

#近似値（掛け算）からの探索
for n in range(1, board.getMinus() + 1):
    upperpruning += sorted(board.getnumbers())[-n]
upperpruning += board.getTarget()
for n in range(1, board.getPlus() + 1):
    lowerpruning += sorted(board.getnumbers())[-n]
lowerpruning = board.getTarget() - lowerpruning

ooc = [] #object of calculation 計算対象
#掛け算記号を探して、探索させる。
for n in range(2, cell * cell, 2):
    if(board.getBoard2(n) == "*"):
        #左
        if(n % cell == 1):
           ooc = MulType(n,2,3,6)
        #右
        elif(n % cell == 0):
            ooc = MulType(n,1,3,4)
        #上
        elif(n <= cell):
            ooc = MulType(n,4,5,6)
```

```
354             #下
355             elif(n > cell * (cell - 1)):
356                 ooc = MulType(n,1,2,5)
357             #他
358             else:
359                 ooc = MulType(n,1,2,3,4,5,6)
360 
361     for n in range(1, int(cell * cell / 2 + 1)): #演算対象の数
362         for m in range(1,cell * cell + 1, 2): #初期位置の決定
363             flag = True #現在の m の値で、行けるところがなくなったらフラグを折る。
364             while(flag):
365                 flag2 = True #無意味な解答確認を避けるための変数
366                 path.append(m)
367                 s = 0
368                 while(s < n):
369                     path2 = tol.numGenerator(path)
370                     #インデックスエラー吐くので。
371                     if(len(path2) > 2):
372                         #最初が掛け算であれば、前にやった試行と重複するため除去する
373                         if(board.getBoard2(path2[1]) == "*"):
374                             tol.registerLog(path2)
375                             tol.registerpLog(path2)
376                             path = []
377                             flag2 = False
378                             break
379                     if(path != path2):
380                         path = path2[:]
381                         s += 1
382                         continue
383                     else:
384                     #長さが 1 の時は、探索打ち止め。
385                         if(len(path) == 1):
386                             path = []
387                             flag = False
388                             break
389                         #長さが 1 より大きい場合は、
390                         else:
391                             #print(path2)
392                             copy = path[:]
393                             #for k in path:
```

```
                            #copy.append(k)
                        tol.registerLog(copy)
                        del path[-2:]
                        s -= 1
                        continue
            if(len(path) != 0 and flag2):
                if(board.answerCheck(*path)):
                    print("Answer:",end="")
                    print(path)
                    print("try is " + str(tryn))
                    print("枝刈りの回数:",end="")
                    print(pruningNum)
                    #exit(0)
                    quit()
                #枝刈りをする
                answer = []
                for l in range(len(path)):
                    answer.append(board.getBoard2(path[l]))
                if(board.calcurate(*answer) > upperpruning
                   or board.calcurate(*answer) < 0):
                    tol.registerpLog(path)
                    pruningNum += 1
                tol.registerLog(path)
                tryn += 1 #答え合わせをしたタイミングで増やす。
            path = []
            #n が 0 の時長さが 1 で終わらせるので、flag を切る。
            if(n == 0):
                flag = False
            #デバッグ用無限ループ防止。
            if(tryn == 1000000):
                print("規定回数を超えました。")
                exit(1)
    tol.clearLog()
```