プログラム・コードの並べ替えパズルにおける正解との距離の変化

山口 琢[1]　中村 陽太[2]　大場 みち子[3]

概要：学習者の演習など知的プロセスを測定・分析・評価する研究では，「適切なプロセスでは，プロセスの進展に伴って正解との編集距離が単調に減少する」ことを前提とする場合がある．この前提は妥当だろうか?われわれは，プログラム・コードの並べ替えパズル「ジグソー・コード」を題材に，これを確かめた．まずコンピュータによるシミュレーションで，全ての局面を生成し，適切と考えられる並べ替え操作をした前後の正解との編集距離の変化を調べた．また人を対象としたプログラミング実験で，受講生がコードを並べ替えるたびに並びを記録し，途中の並びと正解の並びとの編集距離を計算して変化を調べた．その結果，シミュレーションでも実験でも，適切な操作によって正解との距離が遠ざかる場合があることが分かった．適切なプロセスだからといって正解との距離が単調減少するとは限らない．学習分析研究では当面，知的なプロセスにおける適切な操作と編集距離との関係を調べて，知見を蓄える必要があると考える．

The Change of Distance to the Correct Answer in Processes of Solving Reordering Puzzles of Programming Codes

Taku Yamaguchi[1]　Yota Nakamura[2]　Michiko Oba[3]

1. 背景

特にプログラミング教育において，学習者の演習プロセスを測定・分析して，学習者のつまづきや理解を測ろうという研究が行われるようになってきた [2][3][4][5][6][7].

このような人の知的プロセスを測定・分析・評価する研究では，「適切なプロセスでは，プロセスの進展に伴って正解との編集距離が単調に減少する」ことを前提とする場合がある．

2. 目的

「適切なプロセスでは，プロセスの進展に伴って正解との編集距離が単調に減少する」という前提は妥当だろうか?．本研究の目的は，並べ替え問題を題材に，この前提の妥当性を検証することである．

ここで並べ替え問題とは，数の整列 [8] 問題や，プログラミングであれば「ランダムに並んだプログラムの行を，プレイヤーが適切と考える順序に並べ替える」[2](**図 1**)，「カードを並べ替える操作によってプログラムを組み上げる」[3][4]，あるいは「reassemble the lines in their correct order」(コード行を正しい順序に再構築する)[6] といった問題である．

本稿で単調減少とは，i 番目の状態での正解との距離 $d(i)$ について，$i < j$ のとき $d(i) \geq d(j)$ であることを意味する．広義の単調減少，または狭義に単調非増加である．

3. 関連する研究

3.1 正解との距離の変化に基づく推定

Perkins らは，初心者 (novice programmer) のプログラミングを観察して，tinkering というタイプの行動を見出した．Tinker する学生は，まず，ある程度コードを書いて，

[1] フリー
Independent Researcher
[2] 公立はこだて未来大学大学院 システム情報科学研究科
Graduate School of Systems Information Science, Future University Hakodate
[3] 公立はこだて未来大学システム情報科学部
Faculty of Systems Information Science, Future University Hakodate

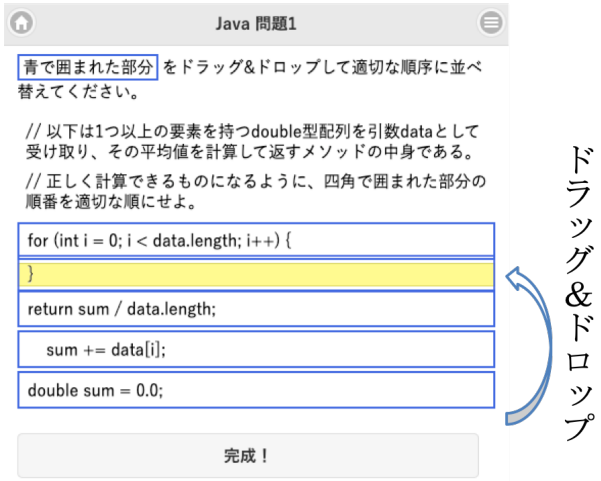

図 1　プログラム・コードの並べ替えパズル

それから上手く動くように期待して少し直す，というやり方でプログラミング問題を解く．このとき「問題を検討し直したり，自分が構文を理解してるか自問することなく，ただ小さな修正が必要なだけという想定のもと修正を繰り返す」[1]．ただし，Perkins らは，これが必ずしも誤った行動であるとは言っていない．

最近の約 1 年間に次の 3 つの論文が発表されて，いずれも正解との編集距離に基づいて，プログラミングの得手/不得手，解答の正しさの度合いを推定している:

森永らは，開発したプログラミング演習アプリケーションの操作に合うようにレーベンシュタイン距離を修正した距離を考案し，その距離を使って問題を解くプロセス (過程) を分析すること提案した．その中で「 正解の並びを得たにも関わらず回答を送信していない状態がみられる．これは正解に気付けていない状態であると考えられる．その理由として，Tinkering を行っていた可能性が考えられる．具体的には，先入観や思い込みで学習者が正解だと考えたパターンがすでにあり，その配置に変更する過程で正解が得られていた場合が考えられる．あるいは要件 (問題) に対する解について，要件 (問題) とそれに対応する解をプログラムの形 (パターン) で憶えており，仮説的な考えを試しながら『建設的な試行錯誤』でアルゴリズムを設計せずパターンで回答しようとしていた過程が時系列変化に現れていた可能性が考えられる．いずれにせよ，このような振る舞いが可視化された点はプログラミングを不得意とする学習者の特徴をより深く知るうえで意義がある 」と述べている [3][4].

Maharjan らは，プログラム行を取捨選択・並べ替えて完成させる Parsons Puzzles について，「正解との編集距離は，解答の正しさの度合いを示す」(“The edit distance of a student’s solution from the correct solution of a Parsons puzzle gives the degree of correctness of the student’s solution.”) と主張して，レーベンシュタイン距離を修正した編集距離に基づく分析手法を提案した．完成形で n 行となるプログラムの並べ替えパズル (問題) を解くプロセスを，nPn = n!個の状態の変化として分析するのに比べて，正解との距離は n に比例する状態しかなく，状態数が少なくてすむ利点を主張した [6]．そして「正解との編集距離の列が非単調で頻繁に上下に振れるのは，試行錯誤を示唆している」(“A non-monotonic trail with frequent up- and down-swings may suggest the use of trial-and-error approach.”) と述べている．さらに「編集距離を使うことでパズル (問題) 固有の詳細を排除できる」(”using edit distances eliminates puzzle-specific details”) ことが、編集距離による分析の利点である主張している．

伊東らは PAD 形式でプログラムを記述するオリジナルの Web アプリケーションを開発した．このアプリケーションは，回答者がブロックを配置するふるまいを記録する．記録の分析について「 正否とその時間的推移各ブロックの正否と模範プログラムとの編集距離を収集する．…模範プログラムとの編集距離と近ければ，一般に論理的にブロックを組み立てられていると考えられる.」と述べている [7].

これら 3 つの研究は，適切な/熟達したプロセスでは正解との距離が単調に減少することを前提とし，途中で距離が増加する場合は不適切な/未熟な操作が行われていると推定している．

また，これらの研究では，正解との距離が遠ざかるときに不適切な (sub-optimal[6]) ことが起きていると推定するが，その不適切な操作を具体的に示さない．また適切さ (optimal[6]) の定義も示されない．むしろ，そのような個別の詳細を示さずに済むことが，距離に基づく分析の利点だと主張している [6].

3.2　コンピュータによる整列での正解との距離

データを一定のルールに従って並べ替えることを整列，またはソート (sort) といい [8]，シンプルだが知的なプロセスである．コンピュータによる整列は，クイックソートなどさまざまなアルゴリズムが研究され実装/採用されてきた．これら成果によって，コンピュータによる整列プロセスは，一つ一つの操作もプロセス全体も適切 (optimal) であると考えられる．クイックソートのプロセスは試行錯誤ではない．

われわれはコンピュータによる整列を題材に，処理の進展に伴う距離の変化を調べた [9]．1 から 10 の 10 個の数をクイックソートなどの整列アルゴリズムでコンピュータに整列させ，途中の並びを記録した．そして，途中の並びと正解の並びとの距離を計算して変化を調べた．距離としてはレーベンシュタイン距離やケイリー距離などを採用した．コンピュータによる整列では，プロセスの途中で正解との距離が遠ざかることがあるし，距離 0 すなわち正解に達しても，さらにプロセスが続いて遠ざかることがあると

分かった．**表 1** は 10!個の整列問題のうち途中で距離が増加した問題の数を，整列アルゴリズムと距離ごとに整理したものである．

コンピュータによる整列プロセスは一つ一つの操作もプロセス全体も適切なので，この結果によって，一般論として「適切なプロセスでは正解との距離が単調に減少する」という前提は誤りであることが示された．コンピュータによる整列では，ある操作で距離が増加する場合も減少する場合も，実行されたコードは同じである．「距離が増加してるので，クイックソートは tinker している」といった推定は誤りである．さらに，バブルソートであれば人にも簡単に真似できるであろう．バブルソートを真似た人による整列プロセスも，同じ結果になる．

3.3 人による並べ替え操作の測定・分析

我々はこれまで，並べ替えプログラミング・パズルのジグソー・コードを開発して，学習者が並べ替えプログラミングするとき，並べ替え操作を測定した．操作の傾向を分析して，学習者へアドバイスすべき内容を明らかにした [2][5].

本稿は，これら人による操作のデータを正解との距離の観点から分析するものである．

4. アプローチ

単調減少の前提の妥当性を，プログラム・コードの並べ替えパズル「ジグソー・コード」を題材に，確かめる．まず適切な操作を確認する．続いて，採用する距離/編集距離を検討し確認する．コンピュータによるシミュレーションで，全ての局面を生成し，適切と考えられる並べ替え操作をした前後の正解との編集距離の変化を調べる．さらに，人を対象としたプログラミング実験で，被験者がコードを並べ替えるたびに並びを記録し，途中の並びと正解の並びとの編集距離を計算して変化を調べる．ある適切な操作の前後で正解との距離が増加するケースが 1 つでもあれば，単調減少の前提は誤りと分かる．全ての適切な操作，あるいはプロセス全体を網羅する必要はない．

(1) 本稿での適切な操作を，シミュレーションと実験のそれぞれについて検討して定義する．
(2) 本稿で採用する距離/編集距離を，シミュレーションと実験のそれぞれについて検討し決める．
(3) コンピュータによるシミュレーションで確認する．全ての局面を生成し，適切と考えられる並べ替え操作をした前後で，正解との編集距離の変化を調べる．
(4) 人を対象としたプログラミング実験で，被験者がコードを並べ替えるたびに並びを記録し，途中の並びと正解の並びとの編集距離の変化を調べる．

4.1 適切な操作

4.1.1 シミュレーション

プログラムのコード行の並べ替え問題で，for 文や if 文のカッコ「{」,「}」の対応を取るのは適切な操作と考えられる．図 1 では，学習者は開き括弧「{」の直後/直下に閉じ括弧「}」を移動しようとしている．この操作は for 文の構文を先ず完成させようとしていると考えられる [2]．そして，これは実際，適切な操作である．なぜならば，ソースコードエディタの入力補完機能は，開き括弧「{」を入力すると閉じ括弧「}」も自動的に入力してくれるからである．多くのプログラミング関係者が括弧の対応を重要視しているからこそ，このような機能が実装されているのであろう．図 1 の正解は**図 3** であり，最終的には「{」と「}」の間に別の行が入っているにかかわらず，である．

4.1.2 実験

人による並べ替え実験では，文献 [2] の操作記録データを検証対象とする．このデータにおいて，発生頻度の多い操作パターンについて，われわれが学習者の意図を説明できた操作を，本稿では適切な操作とみなす．その操作パターンは tinker ではない，すなわち「問題を検討し直したり，自分が構文を理解してるか自問することなく，ただ小さな修正が必要なだけという想定のもと修正を繰り返」[1] したものではないと判断する．

4.2 距離

学習分析の研究では，学習アプリケーションで可能な操作に応じて，編集距離を定義する場合がある [3][4]．本稿では，シミュレーションではいくつかの距離を採用して分析することにし，実験ではレーベンシュタイン距離を採用することにする．

距離としてはレーベンシュタイン距離 (Levenshtein distance)，ハミング距離 (Hamming distance)，ケイリー距離 (Cayley distance) を採用する．レーベンシュタイン距離以外は，同じ文字数の文字などに適用される距離である．

レーベンシュタイン距離は，1 要素の挿入・削除・置換 (その要素を任意の別の要素に置き換える) 操作を何回繰り返すと，一方の並びから他方の並びへ変わるかの操作回数である．断りなく編集距離というとレーベンシュタイン距離を指すことが多い．

ハミング距離は，2 つの並びの同じ位置にあって異なる要素の個数である．(2,5,3,1,4) と (1,2,3,4,5) の距離は，3 だけが同じで他は異なるので，4 である．

ケイリー距離は「交換」操作によってのみ並びを変換して，一方の並びから他方の並びにいたるまでの変換の回数を距離とするものである．**図 4** はケイリー距離の例である．destination と start との距離を測っている．swap(1, 4) は交換操作を表し，1 や 4 は各数の添字である．前の行に交換操作を行った結果が swap() の行に表示されてい

表 1　コンピュータによる整列プロセスの途中で距離が増加した問題の数 [9]

	Levenshtein 距離		Hamming 距離		Caley 距離		Ulam 距離	
	増加した問題数	割合	増加した問題数	割合	増加した問題数	割合	増加した問題数	割合
バブルソート	2,743,264	75.6%	3,512,825	96.8%	3,624,619	99.9%	0	0.0%
ヒープソート	3,628,800	100.0%	3,628,800	100.0%	3,628,800	100.0%	3,628,800	100.0%
挿入ソート	1,339,176	36.9%	1,964,215	54.1%	3,443,878	94.9%	0	0.0%
マージソート	1,852,755	51.1%	2,432,789	67.0%	3,426,453	94.4%	0	0.0%
クイックソート	1,034,478	28.5%	0	0.0%	1,089,872	30.0%	872,908	24.1%
選択ソート	0	0.0%	0	0.0%	0	0.0%	0	0.0%

```
swap-vl                                levenshtein    hamming      cayley       ulam
start    1, 2, 3, 4, 5, 7, 8,10, 6, 9  4 ####        5 #####      4 ####       2 ##
( 9,10)  1, 2, 3, 4, 5, 7, 8, 9, 6,10  2 ##          4 ####       3 ###        1 #
( 6, 8)  1, 2, 3, 4, 5, 7, 6, 9, 8,10  3 ###         4 ####       2 ##         2 ##
( 6, 7)  1, 2, 3, 4, 5, 6, 7, 9, 8,10  2 ##          2 ##         1 #          1 #
( 8, 9)  1, 2, 3, 4, 5, 6, 7, 8, 9,10  0             0            0            0
answer   1, 2, 3, 4, 5, 6, 7, 8, 9,10
```

図 2　クイックソートの距離の変化 [9]

1	double sum = 0.0;
2	for (int i = 0; i < data.length; i++) {
3	sum += data[i];
4	}
5	return sum / data.length;

図 3　正解の順序と行の ID

destination	2, 5, 3, 1, 4
start	1, 2, 3, 4, 5
swap(1, 4) *	4, 2, 3, 1, 5
swap(1, 5)	5, 2, 3, 1, 4
swap(1, 2)	2, 5, 3, 1, 4
number of cycles	3
	(* swapの引数は添字)

図 4　ケイリー距離の例

る．swap(1, 4) は，添字 1 と添字 4 を交換，すなわち添字 1=1，添字 4=4 を交換することを表す．交換操作で交換された数に下線が引かれている．start から 3 回の交換で destination と同じになったので，ケイリー距離は 3 である．

ケイリー距離は，Python ならば SymPy ライブラリの Permutations[10] を使って，R であれば PerMallows パッケージ [11] を使って計算できる．

4.3　シミュレーション

シミュレーションで，全ての局面を生成し，あるいはランダムに生成し，適切と考えられる並べ替え操作をした前後の正解との編集距離の変化を調べる．

計算のため，図 3 のように，各行に ID を割り当てる．図では，1 行目に 1，2 行目に 2，…最終行に 5 を割り当てている．図の問題を解く途中で現れる局面は，1 から 5 の

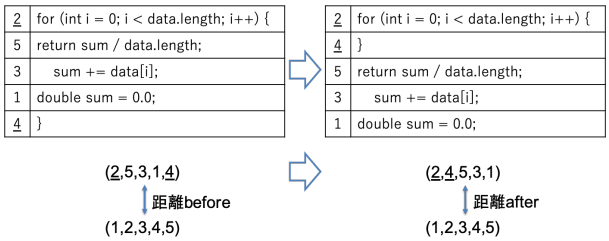

図 5　適切な操作の前後での，正解との距離

並びの変化	正解との距離の変化
(1,2,3,4,5) → (1,2,4,3,5)	0 → 2
(1,2,3,5,4) → (1,2,4,3,5)	2 → 2
(1,2,4,3,5) → (1,2,4,3,5)	2 → 2
(1,2,4,5,3) → (1,2,4,5,3)	2 → 2
(1,2,5,3,4) → (1,2,4,5,3)	2 → 2
(1,2,5,4,3) → (1,2,4,5,3)	2 → 2
(1,3,2,4,5) → (1,3,2,4,5)	2 → 2
(1,3,2,5,4) → (1,3,2,4,5)	3 → 2
(1,3,4,2,5) → (1,3,2,4,5)	2 → 2
…	…

図 6　正解との距離の変化の例

5 個の数から 5 個を取り出した順列に対応する．図 1 でドロップ後の並びには，(2,4,5,3,1) が対応する．図 3 は正解で，(1,2,3,4,5) が対応する．

全局面の数，すなわち全順列の数は ${}_5P_5 = 120$ である．

シミュレーションでは，この 120 個の並びそれぞれについて，まず正解との距離を計算する．続いて，その並びから適切な操作をした後の並びについて，正解との距離を計算する．適切な操作とは，4.1 で取り上げた「括弧の対応を取る」操作である．具体的には，「4」を「2」の直後に移動する．操作後にも正解との距離を計算し，操作の前後で距離が小さくなったのか大きくなったのかを調べる．**図 4.3** に例を示す．

このように，5 行のプログラムの行の並べ替えは，1,

n \ n+1	s9	s10	s11	s12	s13	s14	s15	s16	s17	s18	s19	s20	s21	s22	s23
s9	6	11	18	2	0	0	1	0	1	0	1	0	1	2	7
s10	3	4	6	5	1	1	0	1	0	1	3	0	4	10	1
s11	1	2	9	23	4	6	1	0	2	0	1	1	1	0	1
s12	0	0	1	2	25	13	1	1	1	0	2	2	0	0	1
s13	1	1	1	2	5	28	3	2	4	0	0	0	0	0	1
s14	0	2	1	2	1	0	25	17	3	1	0	1	0	1	1
s15	0	2	0	1	3	0	2	23	6	2	0	1	0	0	0
s16	0	2	0	4	1	2	2	3	23	12	2	2	0	0	0
s17	2	2	0	3	2	1	3	1	1	19	7	1	1	1	1
s18	3	1	0	0	2	1	0	1	1	0	17	6	0	2	2
s19	0	5	0	0	2	0	0	0	1	0	0	18	6	2	2
s20	2	8	0	2	0	0	0	2	1	1	5	0	4	6	2
s21	2	1	1	2	1	0	2	0	0	0	0	0	2	3	5
s22	0	3	3	0	0	2	0	1	0	0	2	1	3	4	0
s23	2	7	5	0	0	0	1	1	1	0	0	2	1	2	3

図 7　「Java 問題 1-2 成績判定 1」の共起行列
s13 の次に s14 を動かすことが多い

2，…5 個の数の並べ替えに還元される．「if 文の開き括弧」といった行の具体的な内容に関係なく，全ての局面や，その局面での操作を同等に扱っている．

図 5 では，正解の並びでは (1,2,3,4,5) と「2」が前で「4」が後に位置する．図では，距離を調べるのは「2」の直後に「4」を置くという，「2」と「4」の順序関係が正解と同じ場合となっている．そこで，図 5 の行の内容的には適切とは思えないが，別の問題でそのようなものがあるとして，「3」を「5」の直後へ移動する場合，すなわち正解と順序関係が逆になる場合についても調べることにする．

4.4　実験

プログラミング実験で，被験者がコードを並べ替えるたびに並びを記録し，途中の並びと正解の並びとの編集距離を計算して変化を調べる．データは，文献 [2] のものを使う．

実験の対象者は情報系大学の学部 2 年生を対象とした情報処理演習を履修している学生である．「情報処理演習は，Java 言語を題材とした課題を通してソフトウェア開発プロセスにおける基本技能を習得することを目的とした授業である」[2].

検証対象としては，文献 [2] のデータのうち，「Java 問題 1-2 成績判定 1」と「Java 問題 7-1 可変な値の格納」を採用する．

図 8 は「Java 問題 1-2 成績判定 1」の問題，**図 7** は，その共起行列．**図 9** は「Java 問題 7-1 可変な値の格納」の問題，**図 10** は，その共起行列である．

適切な操作を見つけるために，まず発生頻度の高い操作パターンを見つける．発生頻度の多い操作パターンは，共起行列から判断する．共起行列とは，プログラムの各コード行について，その行が動かされた次にどの行が動かされることが多いかを示した表である．図 7 では，「s11 の行を動かした次に，s12 の行を動かした回数が 23 回であった」

```
s1   // 評点(score)に基づいて評定(rating)を出力するプログラムを完成させてください.
s2   // 90点以上でS，80点以上90点未満でA，70点以上80点未満でB，60点以上70点未満でC，60点未満でFの順に判定します.
s3   // ただし，評点の初期値は85とし，最高で100とします.
s4   public class Main {
s5       public static void main(String[] args) throws Exception {
s6           int score = 85;
s7           System.out.println("評定: " + judgeRating(score));
s8       }
s9       public static char judgeRating(int score) {
s10          char rating;
s11          if (90 <= score) {
s12              rating = 'S';
s13          } else if (80 <= score && score < 90) {
s14              rating = 'A';
s15          } else if (70 <= score && score < 80) {
s16              rating = 'B';
s17          } else if (60 <= score && score < 70) {
s18              rating = 'C';
s19          } else {
s20              rating = 'F'
s21          }
s22          return rating;
s23      }
s24  }
s25  // コンソールの出力
s26  // > 評定: A
```

図 8　「Java 問題 1-2 成績判定 1」の問題

```
s1   // ランダムな整数値を10回生成し，10より小さいものを全て出力するプログラムを完成させてください.
s2   public class Main {
s3       static ArrayList numbers = new ArrayList();
s4       public static void main(String[] args) {
s5           Random random = new Random();
s6           for (int n = 0; n < 10; n++) {
s7               int i = random.nextInt(20);
s8               if (i < 10) numbers.add(new Integer(i));
s9           }
s10          for (Integer num: numbers) {
s11              System.out.print(num.intValue() + " ");
             }
s12      }
s13  }
s14  // コンソールの出力
s15  // > 2 4 6 8
```

図 9　「Java 問題 7-1 可変な値の格納」の問題

などと読む．セルの背景の赤色 (濃い背景) は，回数が平均 + 標準偏差 x2 よりも大きい，つまり特に頻繁にその順序で操作されたことを示している．

「Java 問題 1-2 成績判定 1」は，共起行列の対角線の 1 つ右隣で頻度が大きい．「Java 問題 7-1 可変な値の格納」には，そのような傾向は見られない．

図 7 では，s13 の次に s14 を動かすことが多いと分かる．文献 [2] では「s13 は 1 つ目の else-if 節であり，s14 は s13 の else-if 節に対応する変数の代入処理である．条件式に対応する処理を条件式の次に操作する傾向が見られた．」とあり，この操作をリーズナブルと見なしている．

n \ n+1	s5	s6	s7	s8	s9	s10	s11
s5	3	3	2	1	1	1	1
s6	2	1	1	1	2	6	0
s7	0	3	2	2	0	1	2
s8	0	0	5	2	1	1	0
s9	1	0	0	2	0	2	2
s10	0	1	1	6	2	1	1
s11	1	1	1	0	2	1	0

図 10 「Java 問題 7-1 可変な値の格納」の共起行列
s6 の次に s10 を動かすことが多い

表 2 シミュレーションで適切な操作前後での距離の増減: 2 の直後に 4 を

距離	遠ざかった回数/全数	割合
レーベンシュタイン距離	25 / 120	21%
ハミング距離	27 / 120	23%
ケイリー距離	31 / 120	26%

表 3 シミュレーションで適切な操作前後での距離の増減: 5 の直後に 3 を

距離	遠ざかった回数/全数	割合
レーベンシュタイン距離	27 / 120	23%
ハミング距離	39 / 120	33%
ケイリー距離	48 / 120	40%

図 10 では，s6 の次に s10 を動かすことが多いと分かる．文献 [2] では，「s6 は 1 つ目の for 文の最初の行，s10 は 2 つ目の for 文の最初の行，s8 は 1 つ目の for 文に含まれる if 文であった.for 文の宣言の次に別の for 文の宣言や if 文を操作するといった，構文を連続して操作する傾向が見られた.」とあり，この操作をリーズナブルと見なしている．

以上の 2 つのケースは適切な操作パターンと判定できたので，それらの操作前後での正解との距離の変化を調べる．

5. 結果

シミュレーションおよび実験で，適切な操作の前後で，正解との距離が遠ざかる場合があることが分かった．

5.1 シミュレーション

表 2 に，全順列 120 通りの状態から 4.1 節の「2」の直後に「4」を置く操作を行った結果，正解との距離が広がった/遠ざかった場合の数を示す．3 種類の距離について，21%から 40%の場合で，正解との距離が遠ざかったことが分かる．

表 3 に，全順列 120 通りの状態から「5」の直後に「3」を置く操作を行った結果，正解との距離が広がった/遠ざかった場合の数を示す．3 種類の距離で 23%から 40%の場合で遠ざかったことが分かる．

5.2 実験

表 4 に「Java 問題 1-2」と「Java 問題 7-1」を解くプロセスで，ある適切な操作をした前後の，正解との距離の変化を示す．「Java 問題 1-2」では操作後に正解から遠ざかることはなかったが，「Java 問題 7-1」では 29%が正解から遠ざかった．

表 4 実験で特定の操作前後での距離の増減

距離	操作	遠ざかった回数/全数	割合
Java 問題 1-2	s13 の次に s14	0 / 28	0%
Java 問題 7-1	s6 の次に s10	2 / 7	29%

6. 考察

一方で，コンピュータによるシミュレーションでは，距離/編集距離にかかわらず無視できない割合で，適切な操作の後で正解との距離が広がった，すなわち正解から遠ざかる．他方，人による実験では，適切な操作の後で遠ざからない問題と，無視できない割合で遠ざかる場合がある問題とがある．

この結果から少なくとも，適切なプロセスだからといって正解との距離が単調減少するとは限らないと言える．とはいえ，人による実験で，適切な操作の前後で遠ざかる場合がなかった問題の存在については検討が必要であろう．次に考察する．

6.1 コンピュータによるシミュレーション

表 3 よりも表 2 の方が遠ざかった場合が少ないのは，操作後の「2」と「4」の順序が正解の順序と同じだからであろう．

いずれにしても表 2 や表 3 で，遠ざかる場合の割合が無視できないのは，全順列の局面が起きる確率が等しく，かつその局面でその操作をする確率が等しいとみなすからである．

6.2 人による実験の場合

「Java 問題 1-2」図 7 では，適切な操作の後で正解から遠ざかることがなかったが，「Java 問題 7-1」図 10 では 29%が正解から遠ざかった．

「Java 問題 1-2 成績判定 1」は，共起行列の対角線の 1 つ右隣で頻度が大きい．共起行列の行見出しも列見出しも，正解の順序で並んでいるので，対角線の 1 つ右隣の頻度が大きいということは，コード行が正解の順序で上から下へ配置されていったことを示唆している．並べながら正解を見つけるのでなく，解き始める前にすでに正解を知っていて，その順序に並べていっただけに近い．ここで正解を知っているとは，if 文や for 文の構文を知っているとは限らず，単に正解を覚えているだけの場合も含む．例えば，それがいわゆる「過去問」，過去に出題された問題であって，かつ皆がそれを予習して暗記している状況でその問題を解けば，このような結果になるだろう．

「Java 問題 7-1 可変な値の格納」の共起行列には，そのような傾向が見られない．このような場合，それが適切な操作であっても，その操作によって正解から遠ざかる場合があるのである．

図 10 などの共起行列で赤い背景 (濃い背景) のセルが存在するのは，様々な局面で人が次に行う操作が偏っているからである．ここが，本稿でのコンピュータによるシミュレーションとの違いであろう．

6.3 結論

各局面で適切な操作を繰り返すような適切な問題解決プロセスで，正解との距離が単調減少するという前提を，無条件におくのは妥当ではない．ある操作で正解から遠ざかったからといって，それだけで，それが不適切な操作だったり，迷いの現れであるとは言えない．単調減少するかどうかは，問題と解く人との組み合わせによる．単調減少は，その組み合わせを表現する指標の 1 つとみなすのが妥当であろう．

7. 残された課題

学習分析研究では当面，さまざまな知的なプロセスにおける操作と編集距離との関係を調べて，知見を蓄える必要があると考える．学習者の操作を測定できる新しい学習アプリケーションを開発すると，それまで見たこともない新しい測定データを得られる．このデータを既存の，あるいは新しい手法で分析することで，それまでにない新しい知見を得られる．その知見は，それまでの常識をデータで裏付けるものかもしれないし，それまで思いもかけなかった新しい事実かもしれない．いずれにせよ，そのような知見の報告は有効な研究となるであろう．

本稿はプロセスの分析に編集距離を利用することを検討したが，プロセスのアウトプットである解答の分析に編集距離を利用することも再検討が必要であろう．Maharjan らは「正解との編集距離は，解答の正しさの度合いを示す」と述べている [6]．この主張は，解答の正しさの度合いを「正解に至るプロセスの進捗度合い」と考えてのものかもしれない．本稿は正解に至る適切なプロセスにおいて，正解との距離が単調減少するとは限らないことを示した．すなわち，正解に至る直前で正解から遠ざかることがあるので，「正解に至るプロセスの進捗度合い」を正解との距離で評価するには，そのようなことが起きないという条件が必要となる．

謝辞 本研究は JSPS 科研費 20H01728 の助成を受けたものである．ジグソー・コードを使った実験について，データを提供して頂いた公立はこだて未来大学の伊藤恵准教授に感謝します．

参考文献

[1] D. N. Perkins, Chris Hancock, Renee Hobbs, Fay Martin, Rebecca Simmons, Conditions of Learning in Novice Programmers, Journal of Educational Computing Research, 2, 1986
https://doi.org/10.2190/GUJT-JCBJ-Q6QU-Q9PL

[2] 中村 陽太, 大場 みち子, 山口 琢, 伊藤 恵, 学習進度に対応するパズルを利用したプログラミング思考過程の分析, 情報処理学会研究報告コンピュータと教育 (CE), 2019-CE-151(1), 2019-09-28
http://id.nii.ac.jp/1001/00199559/

[3] 森永 笑子 , 松本 慎平 , 村上 瑠香 , 林 雄介 , 平嶋 宗, カード操作方式によるプログラミング学習支援システムでの学習過程の可視化方法の提案，人工知能学会 先進的学習科学と工学研究会 85, 92-97, 2019-03-07
http://id.nii.ac.jp/1004/00009654/

[4] 森永 笑子, 松本 慎平, 林 雄介, 平嶋 宗, カード操作方式によるプログラミング学習支援システムにおける学習者の活動に基づく学習課題の特徴分析, 電気学会 C 部門 情報システム研究会 (CIS), 2019-11-10
http://id.nii.ac.jp/1031/00126864/

[5] 中村 陽太, 大場 みち子, 山口 琢, 伊藤 恵, 授業進度に対応するパズルを利用したプログラミング思考過程の分析と教育支援システムの開発, 情報処理学会 第 82 回全国大会講演論文集, 2020-02-20
http://id.nii.ac.jp/1001/00205909/

[6] Salil Maharjan and Amruth Kumar, Using Edit Distance Trails to Analyze Path Solutions of Parsons Puzzles, Educational Data Mining 2020 (EDM 2020), 2020-07-10
https://educationaldatamining.org/files/conferences/EDM2020/papers/paper_163.pdf

[7] 伊東大輝, 島川博光, コードパズルによるプログラミング的思考力を考慮した理解度の推定, 情報科学技術フォーラム (FIT2020) 講演論文集, 2020

[8] 奥村晴彦, ［改訂新版］C 言語による標準アルゴリズム事典, 技術評論社, 2018
https://gihyo.jp/dp/ebook/2018/978-4-7741-9787-6

[9] 山口琢, 大場みち子, コンピュータの整列処理で正解との距離は単調に減少するか?, 情報処理学会研究報告コンピュータと教育 (CE), 2020-CE-157, 2020-10-31
http://id.nii.ac.jp/1001/00207490/

[10] SymPy Development Team, SymPy 1.6.2 documentation, 2020-08-09.
https://docs.sympy.org/latest/index.html

[11] CRAN, PerMallows: Permutations and Mallows Distributions, 2017-04-28
https://cran.r-project.org/package=PerMallows