論文

復帰距離による減点法を用いた並べ替え問題の部分採点法
Partial Scoring of Reordering Tasks with Recovery Distance as Penalty

安間 一雄[*1]
AMMA Kazuo

Email: ammakazuo@mac.com

並べ替え問題の簡明な一般的部分採点法として最大項目連鎖法がある（安間，2010a[1], 2010b[2]）．本研究においては，そこで弱点とされた復帰距離を減点要因として算入するアルゴリズムを追加し，自動計算を実現するコードをプログラミング言語 Xojo[3] を用いて作成した．所与の項目連鎖（解答）から目的連鎖（正解）に至る過程では 1 項目の復帰操作の度に配列が変化するので中間出力に応じた動的アルゴリズムが必要で，冗長操作を防止する手段と共にこれを実現した．復帰距離による減点を加味した採点法は中間得点域において最大項目連鎖法による採点を精緻化したものとなり，正解からのずれが大きくなるほど評価を低くするという原則により忠実に沿った採点が可能になった．但し，言語テストなど実際のテストに適用する際には，談話構造など個別の制約を反映させる必要がある．また，全ての項目の組合せ（順列）を生成して総当たりで本プログラムを確認する方法にはパソコン並びにアプリケーションのメモリーの限界があり，8 項目連鎖が検証できた最大の大きさであった．今後はこの範囲を拡大すべく適切なアルゴリズムの開発もしくは数理的な解法が求められる．

Based on Amma's (2010a[1], 2010b[2]) algorithm and corresponding programme of MRS (Maximal Relative Sequence), a general and rational solution to the partial scoring of reordering tasks, I have developed a Xojo programme[3] which calculates recovery distance as penalty — a factor that a simple MRS was hard to consider. The major feature of the present programme includes a dynamic algorithm for adjusting solution tactics in response to individual moves of items, and a manoeuvre that circumvents redundant moves. The result was an elaboration and precision of scoring among mid-range test-takers, reflecting the fundamental requirement that the farther an answer is to the correct answer, the lower the evaluated score should be. In its actual application, we need to consider some such specific local constraints as discourse structure in language proficiency tests. The procedure of measuring recovery distances was tested using artificially generated data of items 4 to 8 in all possible permutations. It was the memory limitation that prevented the confirmation for longer sequences. A further development in algorithm and/or mathematical solution will be necessary to expand the target of validation.

***1：Department of Interdisciplinary Studies,
Faculty of International Liberal Arts**

1. Introduction

The present study elaborates the algorithm of 'Maximal Relative Sequence' (MRS) (Amma, 2010a[1], 2010b[2]) for calculating partial scoring of reordering tasks, and reports the Xojo programming source code with a special focus on the new addition of procedure, namely the recovery distance penalty. The final product has been confirmed to be viable for four- to eight-item sequences using all possible permutations of character elements.

MRS is a compact and convenient tool for calculating partial scoring of reordering tasks, but it fails to take into consideration the distance of the recovery that irrelevant elements incur. As a result, the final score would stay the same whether an irrelevant item was located just next to the right position or at the far end of the sequence. Our revised MRS+Distance programme (Appendix 2) has enabled the calculation of distance by dynamic algorithm, ie., through a procedure which, for each intermediary step of recovery, identifies the top priority item to be displaced and determines the optimal position for it to move to.

This protocol is robust against sequences of missing, duplicate, and irrelevant items. These extraneous cases are checked and corrected before the measurement procedure begins, which makes this programme distinct from a spreadsheet calculation where only relevant answers are properly dealt with.

Our programme being dynamic costs time and memory to process. In the author's computational environment (Mac Pro 2x2.4GHz, 6-Core Intel Xeon, 24GB memory with 1333MHz DDR3; Mac OS 10.12.5), the machine soon runs out of memory as the length of sequence increases. This processing load is caused by the addition of the recovery distance routine. Some techniques were introduced to alleviate the overwork, including copying the results of the past processing history, which proved effective for tests where answer variations occur in a small range. Although the validity of this programme has not been confirmed exhaustively against all possible sequences of all length, the present programme is considered useful for practical purposes.[2]

2. Background

In language testing, item reordering is one of the popular test types for measuring reading comprehension, particularly the ability to understand and construct discourse-level coherence. One example of the National Centre Examination (2013)[4] (Fig. 1) is a simple task of reordering four illustrations according to the story presented. The test-takers were asked to choose the right sequence out of four combinations of items. The scoring method here is 'all or nothing', ie., one will not get a point unless he/she chooses the correct option (2): B-D-C-A; even if part of the answer B-D or C-A is contained distantly in (1): B-C-D-A; or another part of the answer B-A in (4): D-B-A-C. When the number of items to be reordered increases, the task requires a large amount of processing in the working memory. An 'all-or-nothing' scoring would sound unfair to test-takers who have reached an answer close to the correct sequence.

問5 Which of the following shows the order of the scenes as they appear in the movie? [45]

(1) (B) -> (C) -> (D) -> (A)
(2) (B) -> (D) -> (C) -> (A)
(3) (D) -> (A) -> (B) -> (C)
(4) (D) -> (B) -> (A) -> (C)

[Fig. 1: Sample reordering task from the National Centre Examination (2013)]

Alderson, Percsich, and Szabo (2000)[5] proposes four alternative scoring methods: (1) Exact matches, (2) Previous, (3) Next, and (4) Edges, of which (2) and (3) are variations of 'Adjacent matching' method, and (4) is the linear extension of (1). I shall start by briefly reviewing the limitations of these two major methods, using A-B-C-D-E as a correct sample sequence with five constituting elements.

'Exact matching' gives one point to an element if it is placed in the right position, independently of other elements. For example, an answer A-D-C-B-E will have three points for A, C, E because they are located in the right positions. But if the initial item was misplaced at the end (eg., B-C-D-E-A), there is

2 The present study reports the development of the computer programmes as of November 2016. Later revisions and advancements will be recorded in subsequent publications.

no point at all even if the rest (B-C-D-E) is arranged in the correct order.

In 'Adjacent matching', one point is given to two elements properly juxtaposed. For example, an answer D-E-C-A-B will get two points out of four, which is the full score, for the two adjacent pairs D-E and A-B. In the above case in the 'Exact matching', part of the sequence B-C-D-E will now get three points. Here the score is the simple number of adjacent transitions; it does not consider where the adjacent pairs are. Take A-B-D-E-C, for example, where it is only C which is ill-located. The score is still two points, same as D-E-C-A-B, which appears worse considering the overall configuration. Our natural reaction requires us to consider how close a given answer is to the correct answer. Therefore, we need a more rational measurement procedure reflecting the psychological complexity of the task.

Recently a measurement procedure was proposed by Bollegala, Okazaki, and Ishizuka (2010)[6]. They evaluate the closeness of a given sequence to the correct sequence by calculating 'Average Continuity' defined as

$$AC = \exp\left(\frac{1}{k-1}\right)\sum_{n=2}^{k}\log\left(P_n + 0.001\right) \qquad \text{...(1)}$$

where
n is the length of continuous elements, and
k is the maximum number of continuous elements to be considered for calculation

based on their 'precision of n continuous sentences', namely,

$$P_n = m/(N - n + 1) \qquad \text{...(2)}$$

where
m is the number of continuous elements in the correct order, and
N is the number of elements in the correct sequence

Bollegala, et al. handles only a continuous segment such as B-C-D-E in B-C-D-E-A. Our procedure in contrast has a deeper perspective, allowing a sequence with intervening elements in between to be evaluated. Such a possibility of 'leaps' should be handled in a test of recalling the order of chronological events where the relative order of two events — whether consecutive or not — accounts for part of the test-taker's understanding.

'Maximal Relative Sequence' appeared through my attempts to capture the magnitude of relative order of elements (Amma, 2007a[7], 2007b[8], 2010a[1], 2016[9]). It is a vector increment measurement system, picking up pairs of elements retaining the incremental order in the correct sequence. The longest possible combination of pairs in which all member elements are arrayed in the incremental order is the maximal relative sequence. For example, given an answer B-D-C-A-E, we can pick up either B-C-E or B-D-E, ignoring other elements which would interrupt the increment. In either case the score is the same, two points, corresponding to the number of transitions.

There is a similar protocol called 'Minimal Edit Distance' — a computer algorithm used notably for artificial intelligence and genetic engineering — which seeks to find the fewest steps in converting a character sequence to another. The instances we deal with here are special cases of Levenshtein distance[10] in the sense that only replacement — no addition or deletion — of elements within a given sequence is considered. The basic idea, however, that the distance is calculated by the number of replacement remains the same. We count the number of elements displaced from the original position as a penalty score. An answer C-E-A-D-B is considered to have been reached by (1) dislocating C, E, and B, for example, with a penalty score of three points, leaving A-D intact. Alternatively, one can (2) dislocate C, E, and D, leaving A-B; or (3) dislocate E, A, and B, leaving C-D; or (4) dislocate A, D, and B, leaving C-E. In any case the dislocation occurs three times and the stable transition is one. The frequency of dislocation indicates how far the correct sequence has been distorted — an intuitively relevant means of measuring the degree of remoteness. If we trace back the process, we can measure how far a given answer is by counting the number of dislocation until we reach the correct sequence. However, as the number of elements increases, it becomes diabolically difficult to find the right element to relocate and the right position to move it to for each of the steps involved.

A remarkable breakthrough was a discovery that this dislocation procedure in 'Minimal Edit Distance' (MED) is mathematically equivalent to the measurement by 'Maximal Relative Sequence' (MRS) (Haga, 2006[11]; Amma, 2010a[1], 2010b[2]). Take case (1) above, for example, where we ignored C, E, and B to secure the stable sequence A-D by MRS. C, E, and B are considered extra floating elements all of which must have the right position to move back to. In other words, if we relocate these three elements somehow — in whichever elements to start with and in whatever order — they will be settled in as parts of the correct sequence. Thus the number of these floating elements is the penalty score in MED, and the number of transitions in the stable items is the point in MRS. We can generalise this relationship as either

MRS + MED = full score ...(3a)

or

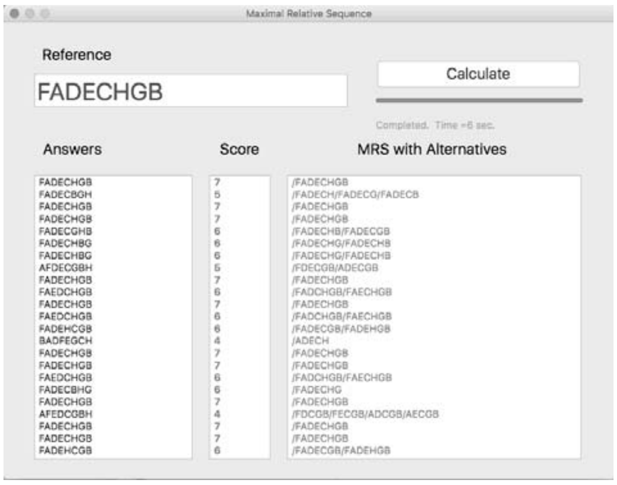
MRS + MED = number of elements - 1 ...(3b)

There are several advantages of MRS over other measurement methods. First of all, it reflects the complexity of cognitive manipulation. The closer the way test-takers consider to be the right sequence to the correct sequence, the less disrupted the correct sequence is in the answer sequence. If the test-taker's knowledge/skill stays far from what he/she is supposed to perform, the answer sequence should be geometrically remote from the correct sequence. The degree of distance between the answer sequence and correct sequence is linearly and uniquely calculable by MED, allowing possible variations of paths. Secondly, it accounts for both local and global configurations. While both D-E-C-A-B and A-B-D-E-C have two points by 'Adjacent matching', D-E-C-A-B has one point (stable elements = D-E or A-B) and A-B-D-E-C has three points (stable elements = A-B-D-E), respectively, by MRS. MRS covers the shortcoming of 'Adjacent matching' which disregards the relative positions of adjacent pairs. Thirdly, it is quicker and lighter to calculate than MED. If the sequence is short enough, the calculation can easily be conducted without a computer. Finally, it is robust against sequences with missing, overlapping, or irrelevant items. For example, if an answer is A-B-C-E when the correct sequence is A-B-C-D-E, MRS secures a stable sequence A-B-C-E (three points). If an answer is A-**C**-B-**C**-E, the longest possible sequence A-B-C-E becomes a stable sequence (three points). If an answer is A-B-C-D-**F**, the stable sequence is A-B-C-D (three points), ignoring the irrelevant element.

3. Elaboration

Despite the advantages stated above, MRS fails to capture how long floating elements have to travel back to the correct position. Two answers B-C-**A**-D-E and B-C-D-E-**A** have equally two points, but A in B-C-**A**-D-E has to go over two preceding elements B and C, whereas A in B-C-D-E-**A** over four elements B, C, D, and E. If the similarity between the correct answer and given answer is the key construct in this method, the distance of recovery must be incorporated.

In addition to the procedure for MRS (Fig. 2) I have added a routine for calculating recovery distance (Fig. 3) in this upgrade.[3] The outline for the entire process is described and illustrated in the following steps (see Fig. 4 for the overview). Refer to Appendix 2 for the line number.



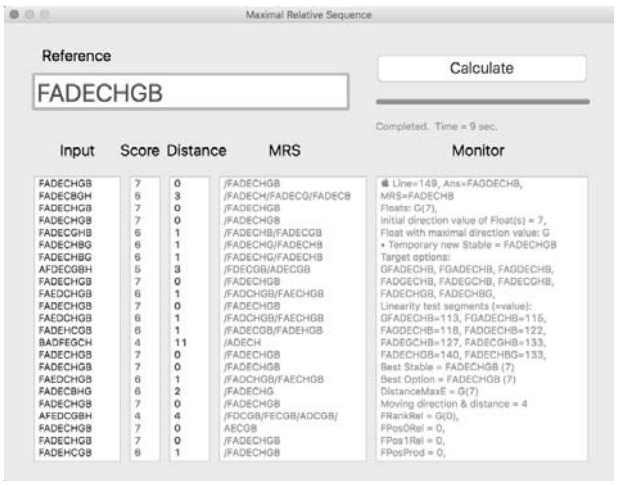
Reference

FADECHGB

Answers	Score	MRS with Alternatives
FADECHGB	7	/FADECHGB
FADECBGH	5	/FADECH/FADECG/FADECB
FADECHGB	7	/FADECHGB
FADECHGB	7	/FADECHGB
FADECGHB	6	/FADECHB/FADECGB
FADECHBG	6	/FADECHG/FADECHB
FADECHBG	6	/FADECHG/FADECHB
AFDECGBH	5	/FDECGB/ADECGB
FADECHGB	7	/FADECHGB
FAEDCHGB	6	/FADCHGB/FAECHGB
FADECHGB	7	/FADECHGB
FAEDCHGB	6	/FADCHGB/FAECHGB
FADEHCGB	6	/FADECGB/FADEHGB
BADFEGCH	4	/ADECH
FADECHGB	7	/FADECHGB
FADECHGB	7	/FADECHGB
FAEDCHGB	6	/FADCHGB/FAECHGB
FADECBHG	6	/FADECHG
FADECHGB	7	/FADECHGB
AFEDCGBH	4	/FDCGB/FECGB/ADCGB/AECGB
FADECHGB	7	/FADECHGB
FADECHGB	7	/FADECHGB
FADEHCGB	6	/FADECGB/FADEHGB

[Fig. 2: Xojo window for MRS]

[3] While MRS leaves overlapping elements intact, MRS+Distance deletes all overlapping elements. This is because (1) leaving all overlapping elements intact causes a malfunction in the distance recovery routine, and (2) leaving one element intact causes difference in recovery distance depending on which element is left undeleted.



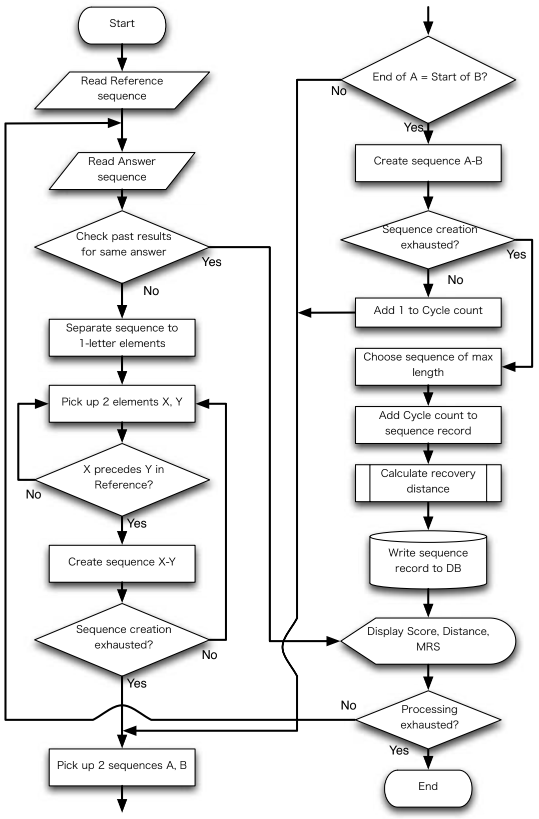
FADECHGB

Completed. Time = 9 sec.

Input	Score	Distance	MRS
FADECHGB	7	0	/FADECHGB
FADECBGH	5	3	/FADECH/FADECG/FADECB
FADECHGB	7	0	/FADECHGB
FADECHGB	7	0	/FADECHGB
FADECGHB	6	1	/FADECHB/FADECGB
FADECHBG	6	1	/FADECHG/FADECHB
FADECHBG	6	1	/FADECHG/FADECHB
AFDECGBH	5	3	/FDECGB/ADECGB
FADECHGB	7	0	/FADECHGB
FAEDCHGB	6	1	/FADCHGB/FAECHGB
FADECHGB	7	0	/FADECHGB
FAEDCHGB	6	1	/FADCHGB/FAECHGB
FADEHCGB	6	1	/FADECGB/FADEHGB
BADFEGCH	4	11	/ADECH
FADECHGB	7	0	/FADECHGB
FADECHGB	7	0	/FADECHGB
FAEDCHGB	6	1	/FADCHGB/FAECHGB
FADECBHG	6	2	/FADECHG
FADECHGB	7	0	/FADECHGB
AFEDCGBH	4	4	/FDCGB/FECGB/ADCGB/
FADECHGB	7	0	AECGB
FADECHGB	7	0	/FADECHGB
FADEHCGB	6	1	/FADECHGB

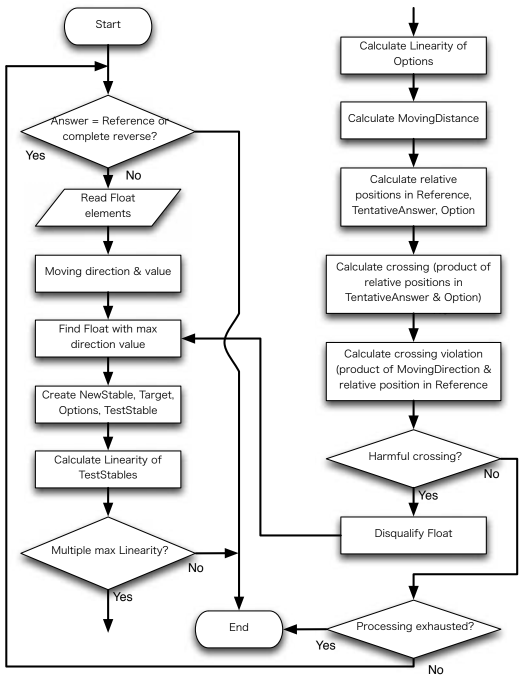
```
 Line=149, Ans=FAGDECHB,
MRS=FADECHB
Floats: G(7),
Initial direction value of Float(s) = 7,
Float with maximal direction value: G
• Temporary new Stable = FADECHGB
Target options:
GFADECHB, FGADECHB, FAGDECHB,
FADGECHB, FADEGCHB, FADECGHB,
FADECHGB, FADECHBG,
Linearity test segments (=value):
GFADECHB=113, FGADECHB=115,
FAGDECHB=118, FADGECHB=122,
FADEGCHB=127, FADECGHB=133,
FADECHGB=140, FADECHBG=133,
Best Stable = FADECHGB (7)
Best Option = FADECHGB (7)
DistanceMaxE = G(7)
Moving direction & distance = 4
FRankRel = G(0),
FPos0Rel = 0,
FPos1Rel = 0,
FPosProd = 0,
```

[Fig. 3: Xojo window for MRS+Distance]



Start
Read Reference sequence
Read Answer sequence
Check past results for same answer
Yes
No
Separate sequence to 1-letter elements
Pick up 2 elements X, Y
X precedes Y in Reference?
No
Yes
Create sequence X-Y
Sequence creation exhausted?
No
Yes
Pick up 2 sequences A, B
End of A = Start of B?
No
Yes
Create sequence A-B
Sequence creation exhausted?
Yes
No
Add 1 to Cycle count
Choose sequence of max length
Add Cycle count to sequence record
Calculate recovery distance
Write sequence record to DB
Display Score, Distance, MRS
No
Processing exhausted?
Yes
End

[Fig. 4: Flow chart of the main code]



Start
Answer = Reference or complete reverse?
Yes
No
Read Float elements
Moving direction & value
Find Float with max direction value
Create NewStable, Target, Options, TestStable
Calculate Linearity of TestStables
Multiple max Linearity?
No
Yes
End
Calculate Linearity of Options
Calculate MovingDistance
Calculate relative positions in Reference, TentativeAnswer, Option
Calculate crossing (product of relative positions in TentativeAnswer & Option)
Calculate crossing violation (product of MovingDirection & relative position in Reference
Harmful crossing?
No
Yes
Disqualify Float
Processing exhausted?
Yes
No

[Fig. 5: Flow chart of the recovery distance code]

(1) Prepare arrays of a size of 26. This size is tentative considering a typical reordering tasks may use character marks from A up to maximum Z (Lines 12-15).

(2) The code for Macintosh platform or an alternative is specified following the Xojo manual. Although it does not seem to affect the performance by the alternative platform machine[4], it remains a mystery that if answers are directly typed with carriage returns input from the keyboard in the 'Answers' box instead of being pasted from a text editor or Excel, the programme does not recognise the carriage return but concatenate multiple answer sequences into one sequence (Line 21).

(3) Read correct answer ('Reference') sequence (Line 24), and dissolve it into character elements ('R') (Lines 37-39). Here we take an example of correct sequence A-B-C-D-E.

(4) Read student answer ('AnsRaw') and dissolve it into character elements. If any element is not a member of the correct answer, exclude it and

[4] In fact, this code runs perfectly on a Windows machine (Windows 7 / Intel Core 32bit i5-3230M, 2.6GHz, 4GBRAM) without changing the code.

reform the tentative answer sequence ('AnsH') (Lines 49-62). We refer to a sample case with an answer B-E-C-D-A throughout this illustration.

(5) If the tentative answer sequence contains multiple elements, delete all of them (Lines 72-95).

(6) If the tentative answer sequence is found in the past record, skip calculation and submit the data for display (Lines 97-110). This dynamic procedure improves the performance by about 30 to 50% (Amma, 2010a[1]).

(7) Pick up single characters from tentative answer sequence ('E') (Lines 115-118). [Sample case: B, E, C, D, A]

(8) Concatenate two characters if they are arrayed in an ascending order in the correct answer (Lines 120-134). [Sample case: B-E, B-C, B-D, C-D]

(9) Concatenate two sequences to create a longer sequence ('Sprout') if the final character of the first one and the initial character of the second one are equivalent. Then repeat this process (Lines 140-170). [Sample case: B-C-D]

(10) Choose the longest Sprout — there can be multiple — as the sequence of Maximal Relative Sequence ('MRSe') (Lines 172-180). [Sample case: B-C-D]

[Calculate recovery distance] (See Fig. 5 for the overview.)

(11) If the tentative answer is correct or in a completely reverse order of the correct answer, skip this routine. Otherwise use the first option of MRS as a stable sequence ('Stable') (Lines 183-193).

(12) Identify individual elements of the floating items ('F') (Lines 201-211). [Sample case: E, A]

(13) Assign the value of moving direction of floating element from the correct sequence to the tentative answer sequence: 1 for moving right, -1 for moving left. Multiply it with its location in the correct sequence ('Direction') (Lines 216-227). [Sample case: A = -1, E = 5]

(14) Starting with the floating element with maximum value of Direction, mix it with stable elements in the correct order ('NewStable') (Lines 233-257). [Sample case: NewStable = B-C-D-**E**]

(15) Create a sequence 'Target' made of tentative answer elements without the floating element in question (Lines 261-270). [Sample case: Target = B-C-D-A]

(16) Create sequences 'Option' made of 'Target' in which the floating element is inserted in all possible combinations (Lines 273-279). [Sample case: Options = **E**-B-C-D-A, B-**E**-C-D-A, B-C-**E**-D-A, B-C-D-**E**-A, B-C-D-A-**E**]

(17) Create sequences 'TestStable' made of stable elements and the floating element in all possible combinations (Lines 283-291). [Sample case: TestStables = **E**-B-C-D, B-**E**-C-D, B-C-**E**-D, B-C-D-**E**, B-C-D-**E**]

(18) Choose 'Option' corresponding to 'TestStable' with maximum 'Linearity' value (Lines 295-313). [Sample case: B-C-D-**E** (fourth TestStable) = 20, B-C-D-**E** (fifth TestStable) = 20]

(19) If there are multiple 'TestStable' with the same maximum 'Linearity' values, choose the 'Option' which has the maximum 'Linearity' value from among the corresponding 'TestStable' (Lines 315-334). [Sample case: B-C-D-**E**-A (fourth Option) = 20, B-C-D-A-**E** (fifth Option) = 21. So element E should move after A.

(20) Check if the floating element in the 'Option' either crosses younger item located left to the floating element or crosses older item located right to the floating element (= 'Crossing Constraint'). If the product of the relative position of a floating element X other than the one to be dislocated in the correct sequence and the relative position of the floating element being dislocated Y from the answer sequence to the Option sequence is positive, the element X should block the floating element Y to cross it over. If the violation is observed (ie., the product is proved to be positive), nullify the 'Direction' of the floating element, and start over from (12) (Lines 339-468). [Sample case: A's product of its relative position to element E in the answer sequence is 3, that in the suggested best Option sequence is -1; the product is -3, which means E crosses over A. But the product of E's MovingDistance (+3) and A's relative position in the correct answer (-4) is negative (= -12), therefore the crossing is not harmful (ie., E moves right, crossing over a young element A). Therefore the dislocation of E from B-**E**-C-D-A to B-C-D-A-**E** does not violate the Crossing Constraint.

(21) Repeat (10)-(18) and accumulate the recovery distance for the MRS (Lines 197-493). [Sample case: B-C-D-A-E now becomes a new temporary answer AnsH, and B-C-D-E becomes a new temporary stable.]

(22) Record and display score, recovery distance ('CumDistance'), and varieties of MRS (Lines 497-516).

'Linearity' in step (17) is a value for representing the degree of similarity of a sequence to the target sequence (correct answer). A point of the number of elements - 1 is given to adjacent elements which are arranged in the ascending order (with reference to the order in the correct sequence); a point of the number of elements - 2 is given to two elements with one item in between if the two elements are arranged in the ascending order; and so on: in general, a point of

(the number of elements) - (n-1)

is given to the pair of two elements with n elements in between if the two elements are arranged in the ascending order; otherwise, zero. Thus a sequence A-C-**E**-B-D (case 1) is given 20 points (Table 1) whereas a sequence A-C-B-D-**E** (case 2) is given 26 points (Table 2). We can decide which sequence is closer to the correct answer when we need to place E in a target sequence A-C-B-D.

[Table 1: Scoring process for 'Linearity' (case 1)]

ACEBD		*4*	*3*	*2*	*1*	Sum
	A	4	3	2	1	10
		C	4	0	2	6
			E	0	0	0
				B	4	4
					D	**20**

[Table 2: Scoring process for 'Linearity' (case 2)]

ACBDE		*4*	*3*	*2*	*1*	Sum
	A	4	3	2	1	10
		C	0	3	2	5
			B	4	3	7
				D	4	4
					E	**26**

'Crossing Constraint' (step 20) is a device to prevent unnecessary moves. Take an example of E-G-H-D-F-C-B-A, where the MRS is E-G-H and the floating elements are D, F, C, B, and A. Without this constraint element F, which has the largest Direction value (=6), would move left aiming at the position between E and G (= E-F-G-...) indicated by the 'Linearity' judgement. It crosses over D, a younger element to F. In the next cycle D has to cross over F, which makes the initial movement of F redundant. The total magnitude of displacement would be 25. But if we started with A it would be 23. In principle, this crossing over undue elements occurs when an element moves left crossing over younger element(s) or moves right crossing over older element(s). If a 'harmful' crossing is detected, the programme resets the Direction value of the tentative floating element to -9999, giving way to the next candidate element so it turns out to have the highest Direction value now. In our example above, the order of the floating candidates is initially F, A, B, C. As a result of this crossing check, the new order becomes A, B, C, F.

Having calculated the recovery distance for each answer sequence, one needs to incorporate it into the scoring by MRS. Our tentative adjustment equation is

Adjusted score = MRS x (1 - Penalty rate) ...(4)
where
Penalty rate = (Recovery distance) / (Maximum recovery distance)
where
Maximum recovery distance = $n \times (n - 1) / 2$
where
n is the number of elements in the sequence.

Table 3 shows some cases of sequences and corresponding MRS and adjusted MRS scores. The top three cases which have the same MRS score have a variety of adjusted scores.

[Table 3: Sample scorings by MRS and MRS+Distance]

Answer	MRS	Score	Distance	Adj score
BADCE	ACE	2	2	1.8
DBACE	ACE	2	4	1.2
BADEC	ADE	2	3	1.4
EDCAB	ED	1	9	0.1
BCDEA	BCDE	3	4	1.8
EDCBA		0	10	0

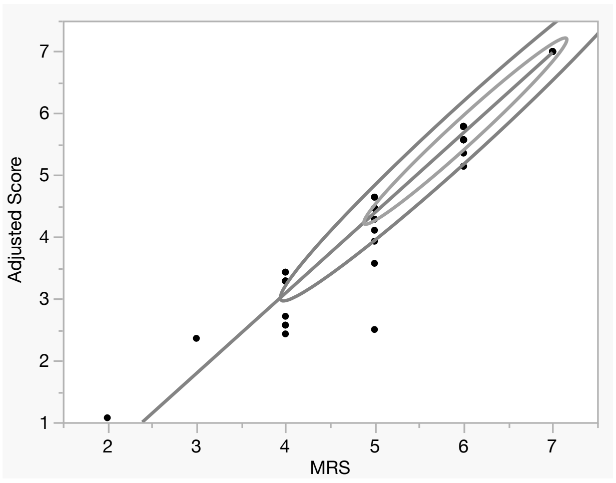
4. Application

The scoring by our programme was applied to an actual test data. The test, conducted against Japanese university students (n = 149) as part of the reading class, consisted of three comprehension questions. In Question 2 the subjects watched a video story and were asked to reorder eight statements to make them coherent. This task was scored by five methods: Binary, Exact, Adjacent, MRS, and MRS+Distance. The Alpha reliability coefficient was calculated for the entire test by each scoring method (Table 4). The result shows that MRS+Distance does not indicate the highest reliability score, although it exceeds Binary and Adjacent methods by far and Exact slightly.

[Table 4: Reliability of scoring by five methods]

Binary	0.273
Exact	0.534
Adjacent	0.479
MRS	0.624
MRS+Distance	0.560

Using the same data, scores by MRS and scores by MRS+Distance was compared by JMP[12] (Fig. 6). The Pearson's correlation coefficient was as high as 0.982. However, a close look reveals that the scores are scattered in the middle range. It would mean improved discrimination among these test-takers.



[Fig. 6: Bivariate distribution of MRS and MRS+Distance scores. Ovals represent density eclipses at p = 0.50 (inner oval) and p = 0.90 (outer oval).]

In the next stage of trial the programme was challenged by a large size of data. Considering that the test data described above is a sample of a larger population, the data was first duplicated ten times to generate a middle-size set of 1490 answers, then the original set was duplicated a hundred times to generate a large-size set of 14900 answers. With the original small-size data (n = 149) the average processing time after five times' trial was 9.0 seconds. The middle-size data took 100.4 seconds, and the large-size data 1602.3 seconds. The average processing speed tends to slow down slightly, but this test has supported a positive view that our programme stands against the data size in practical situations.

5. Validation

In order to secure the correct scripting of the programme a test procedure was conducted using artificial sequences of all possible permutations of elements. For example, when the sequence size is three, all possible permutations are A-B-C, A-C-B, B-C-A, B-A-C, C-A-B, and C-B-A. To conduct this enumeration a new Xojo programme was written (Appendix 3)[5].

Here, in brief, a sequence of mutually exclusive elements is generated by converting a serial index into a combination of elements. For example, index 1 is converted to A-A-A, 2 to A-A-B, 3 to A-A-C, 4 to A-B-A, ... 6 to C-B-A, ... 27 to C-C-C. Given an index *i*, the sequence size *n*, and column of element (or location in the sequence) *c*, the number of the character *P* (A = 1, B = 2, C = 3, etc.) is calculated as[6]

$$P = ((i - 1) \setminus n^\wedge(c - 1)) \bmod n) + 1 \qquad ...(5)$$

Having obtained a sequence, the mutual exclusiveness is checked by the product of the primes assigned uniquely to each of the characters (eg., 2 to A, 3 to B, 5 to C, etc.). When the sequence size is three, all the combinations of mutually exclusive elements have the product value of 30 (=

[5] Although my algorithm is simple enough, there may be less resource-consuming alternatives (eg., Semba, 1989[13], p.67).

[6] The notation 'x \ y' means integer quotient of x over y. The formula (5) can be expressed by Excel as
=MOD(QUOTIENT([i]-1,[n]^([c]-1)),[n])+1
where bracketed items should be replaced by locations of corresponding cells.

2 x 3 x 5) in common. If a tentative sequence has this value it is the target sequence with mutually exclusive elements. The index is given from 1 to the maximum size (= n^n), and the number of the target sequences is n!. Since the index may become a very large number (eg., a sequence of eight elements would need 16,777,216 cases to be tested), a normal spreadsheet would soon use up the computer memory — in fact, the maximum number of lines Excel can handle is only 1.05 million. I used JMP[12] instead, which can deal in theory with up to 2,147,483,647 cases — yet not enough for a 10-character sequence, which would require 10 billion lines. Mutually exclusive sequences were extracted and put into the MRS+Distance programme. Still under the memory limitation, I managed to generate lists of permutation sequences for up to eight elements.

The permutation sequences were the test input for our MRS+Distance programme. Where the sequence size was between three and seven, the programme completed calculation. When the sequence size was eight, a memory error occurred probably due to the size of the output record that was stored in the internal memory. So I had to cut the list into small sections of 2500 sequences, and repeated the production to complete the exhaustive confirmation.

6. Conclusion and future prospect

The present study is a proposal for a rational measurement of partially completed answers to reordering tasks. It reflects the principle that the more remote an answer is to the correct answer the less evaluation value the test-taker should be given. The procedure involves a dynamic algorithm, and some techniques, notably ‘Linearity’ score which helps specify the optimal sequence formation and ‘Crossing constraint’ which prevents redundant moves of elements, were incorporated to properly implement the procedure.

The validation of this programme has been incomplete due to the limitation of memory resource. Nevertheless, considering the fact that, in language testing, for example, the sequence size can be up to eight to ten, further development in memory management is expected.

The present measurement procedure is a general solution to partial scoring. In its practical application task-specific restrictions may occur. Alderson, Percsich, and Szabo (2000)[5] quote a reordering task for reading comprehension in English. They claim that the text in Appendix 1b is to be partially credited in contrast with that in Appendix 1a, which is arranged in the correct order. As far as the author investigated, about half of the native speaker language teachers judged 1b as *perfectly* acceptable rather than *partially* acceptable; furthermore, they judged 1c as perfectly unacceptable. The one irrelevant sentence (a) is two elements away from the correct position in B whereas it is one element away in 1c. 1c turned out to be unacceptable because sentence (a) interrupted the close connection between sentences (e) and (c). So a simple calculation of recovery distance does not necessarily reflect the acceptability judgement for the natural language discourse. In cases where such specific restrictions are evident, a simple MRS might work better. Alternatively, we would have to devise a more complex penalty system.

References

[1] Amma K., “整序問題の部分採点法とそのプログラミング” [Partial scoring of sequencing problems and its programming]. 『獨協大学外国語教育研究』（獨協大学外国語教育研究所）, No.28, pp.1-29 (2010a)

[2] Amma K., “Comparison of partial scoring methods in sequencing tasks with reference to internal reliability”, Paper presented at the 49th National Conference of the Japan Association of College English Teachers (Miyagi, Japan) (2010b)

[3] Xojo (version 2017 release 2). URL: https://www.xojo.com/ (2017)

[4] 独立行政法人大学入試センター [National Centre for University Entrance Examinations],「平成25年度本試験の問題」[Examination questions for academic year 2013]. URL: http://www.dnc.ac.jp/sp/data/shiken_jouhou/h25/jisshikekka/ (Retrieved November 2017)

[5] Alderson, J. C., Percsich, R., Szabo, G., “Sequencing as an item type”, *Language Testing*, Vol.17, No.4, pp. 423-447

[6] Bollegala, D, Okazaki, N., Ishizuka, M., “A bottom-up approach to sentence ordering for multi-document summarization”, *Information Processing and Management*, No.46, pp.89–109 (2010)

[7] Amma K., “Partial scoring of sequencing tasks”, Paper presented at the International Meeting of the Psychometrics Society, 2007 (Tokyo) (2007a)

[8] Amma K., “Appraisal of partial scoring in sequencing tasks”, Paper presented at the 46th National

Conference of the Japan Association of College English Teachers (Hiroshima, Japan) (2007b)

[9] Amma K., “Partial scoring of sequencing tasks with distance penalty”, Paper presented at The Association for Language Testing and Assessment of Australia and New Zealand (ALTAANZ) 2016 (Auckland, New Zealand) (2016)

[10] Levenshtein, V. I., “Binary codes capable of correcting deletions, insertions and reversals”, *Soviet Physics-Doklady*, 10(8), pp.707-710 (1966)

[11] Haga T., Personal communication at Multivariate Study Group Meeting on 15 December 2006 (Tokyo) (2006)

[12] JMP (version 13.0). Cary, NC: SAS Institute, Inc. URL: http://www.jmp.com/ (2016)

[13] Semba I. [仙波一郎], 『組合せアルゴリズム』 [*Combinational Algorithm*], サイエンス社 (1989)

Appendix 1a: Reordering task in Alderson, et al (2000)[5]: correct sequence

(b) A technician at Compaq Computers told of a frantic call he received on the help line.

(d) It was from a woman whose new computer simply wouldn’t work.

(a) She said she’d taken the computer out of the box, plugged it in, and sat there for 20 minutes waiting for something to happen.

(e) The tech guy asked her what happened when she pressed the power switch.

(c) The woman replied, ‘What power switch?’

Appendix 1b: Partially credited sequence arranged from Appendix 1a

(b) A technician at Compaq Computers told of a frantic call he received on the help line.

(d) It was from a woman whose new computer simply wouldn’t work.

(e) The tech guy asked her what happened when she pressed the power switch.

(c) The woman replied, ‘What power switch?’

(a) She said she’d taken the computer out of the box, plugged it in, and sat there for 20 minutes waiting for something to happen.

Appendix 1c: Incorrect sequence arranged from Appendix 1a

(b) A technician at Compaq Computers told of a frantic call he received on the help line.

(d) It was from a woman whose new computer simply wouldn’t work.

(e) The tech guy asked her what happened when she pressed the power switch.

(a) She said she’d taken the computer out of the box, plugged it in, and sat there for 20 minutes waiting for something to happen.

(c) The woman replied, ‘What power switch?’

Appendix 2: Xojo source code of MRS+Distance

MRS+Distance coding script
©AMMA Kazuo, 2016

```
Dim CR, Ref, AnsRaw, AnsH, Ans, AnsList, AnsListSegment, Sprout(), Elem, SeedSeq, Seed, Connect, MRSe(), MRSseq,
HeadDelimiter, Float, DistanceMaxE, Stable, NewStable, H, Target, BestOption as String
Dim n, i, j, k, NR, NL_Ans, NC, Mark, LocAns, Cycle, NI_Sprout, NI_SproutOrig, NI_Float, Sign, DirectionMax, Linearity,
LinearityMax, CumDistance, DisplacementDistance, ReturnPosition, CumReturnDist, EMovingDistance, CrossingFlag as Integer
Dim E(26), R(26), Option(26), TestStable(26), LinearityMaxOption(26) as String          //Set tentative array size.
Dim LinearityScore(26), FRankRel(26), FPos0Rel(26), FPos1Rel(26), FPosProd(26) as Integer      //Set tentative array size.
Dim Init, Fin, Lapse as Double
Dim Now as Date

    CR = EndOfLine.Macintosh                        //For Windows, change to "EndOfLine.Windows".
    NL_Ans = CountFields(Input.Text, CR)
    Ref = Reference.Text
    NR = Len(Ref)

    ProgressBar1.Value = 0                          //Reset progress bar.
    Score.Text = ""
    MRS.Text = ""
    Distance.Text = ""
    SearchResult.Text = ""
    AnsList = ""

    Now = New Date
    Init = Now.TotalSeconds                         //Record initial time.

    For i = 1 to NR
        R(i) = Mid(Ref, i, 1)                       //Read elements of correct answer.
    Next

    For n=1 to NL_Ans
        LogBox.Text = ""                            //Reset monitor for each answer.
        Ans = ""
        SeedSeq = ""
        Redim Sprout(0)
        Sprout(0) = ""
        Redim MRSe(0)
        MRSe(0) = ""
        AnsRaw = NthField(Input.Text, CR, n)        //Pick up each line.
        NC = Len(AnsRaw)

        Mark = 0
        AnsH = ""                                   //Reset temporary answer
        For i = 1 to NC
            Elem = Mid(AnsRaw, i, 1)                //Pick up single character.
            If InStr(Ref, Elem) <> 0 then           //Confirm: Elem is contained in Ref; if non-member, then exclude
                                                    from Ans.
                AnsH = AnsH + Elem                  //Recreate purified answer sequence.
            Else
                Mark = 1
            End
        Next

        If Mark = 1 then
            LogBox.Text = LogBox.Text + “@ Line " + Str(n) + " contained illegal character(s), purified as " + AnsH + CR + CR
        Else
            AnsH = AnsRaw
        End

        Mark = 0
        For i = 1 to NC
    E(i) = Mid(AnsH, i , 1)                     //Pick up single character.
Next
For i = 1 to NC
    For j = 2 to NC
        If i <> j And E(i) = E(j) then          //Check if there is any overlap.
            E(i) = ""                           //All overlapping characters are deleted.
            E(j) = ""                           //All overlapping characters are deleted.
            Mark = 1
        End
    Next
Next
For i = 1 to NC
    Ans = Ans + E(i)                            //Recreate purified answer sequence.
Next
If Mark = 1 then
    LogBox.Text = LogBox.Text + "@ Line " + Str(n) + " contained overlapping characters, replaced as " + Ans + CR + CR
Else
    Ans = AnsH
End

LocAns = InStr(AnsList, Ans + "/")              //Search past record.
If LocAns <> 0 then //If answer exists in past record, then refer to the relevant data.
    AnsListSegment = Mid(AnsList, LocAns, Len(AnsList))
    Cycle = Val(Mid(AnsListSegment, InStr(AnsListSegment, "/") + 1, (InStr(AnsListSegment, "_") - InStr(AnsListSegment,
            "/")) - 1))
    CumDistance = Val(Mid(AnsListSegment, InStr(AnsListSegment, "_") + 1, (InStr(AnsListSegment, "|") -
            InStr(AnsListSegment, "_")) - 1))
    MRSseq = Mid(AnsListSegment, InStr(AnsListSegment, "|") + 1, (InStr(AnsListSegment, ",") - InStr(AnsListSegment,
            "|")) - 1)
    SearchResult.Text = "Found past record: " + Str(Round(LocAns / Len(AnsList) * 100)) + "% position."
                                                //Display result of search record.  If overlap, then skip calculation.
Else
    NC = Len(Ans)
    For i = 1 to NC
        E(i) = Mid(Ans, i, 1) //Pick up single character.
    Next

    For i = 1 to NC-1
        Seed = E(i)
        For j = i+1 to NC
            If InStr(Ref, E(i)) < InStr(Ref, E(j)) then  //2 characters to be concatenated must be in ascending order.
                Seed = E(i) + E(j)              //Concatenate adjacent characters.
                If InStr(SeedSeq, Seed) = 0 then
                    Sprout.Append Seed          //Add concatenated character sequence Seed to array Sprout.
                    SeedSeq = SeedSeq + "," + Seed
                                                //Store concatenated character sequence Seed in text string SeedSeq.
                End
            End
        Next
    Next

    Cycle = 1
    NI_SproutOrig = UBound(Sprout())            //Count items in array Sprout.

    Do
        NI_Sprout = UBound(Sprout())
        For i = 1 to NI_Sprout
            For j = 2 to NI_SproutOrig
                If Right(Sprout(i), 1) = Left(Sprout(j), 1) then
                                                //Find a sprout whose head meets the end of present sprout.
                    Connect = Sprout(i) + Mid(Sprout(j), 2, Len(Sprout(j)))       //Concatenate adjacent items.
                    If InStr(SeedSeq, Connect) = 0 then     //If newly concatenated sequence does not exist so far,...
                        Sprout.Append Connect   //Add concatenated sequence Connect to array Sprout.
                        SeedSeq = SeedSeq + "," + Connect
                                                //Store concatenated sequence in text string SeedSeq.
                End
            End
        Next
    Next

    If UBound(Sprout()) = NI_Sprout then        //If no more increment of Sprout, then exit.
        If SeedSeq = "" then                    //If Ans is perfect reverse, then reset score.
            Cycle = 0
        End
        Exit
    Else
        Cycle = Cycle + 1
    End
Loop

For i = NI_Sprout DownTo 1
    If Len(Sprout(i)) = Len(Sprout(NI_Sprout)) then     //Choose longest sprouts.
        MRSe.Append Sprout(i)                   //Add concatenated sequence Sprout to goal array MRSe.
    Else
        Exit
    End
Next

//Calculate distance of recovery.
If Ans = Ref then                               //If Ans = Ref, then circumvent loop.
    CumDistance = 0
Else
    CumReturnDist = 0
    AnsH = Ans      //Reset AnsH by initial value.
    If SeedSeq = "" then                        //If Ans is complete reverse, then circumvent loop.
        Stable = ""
    Else
        Stable = MRSe(1)                        //Pick up 1st MRS as initial Stable.
    End
    LogBox.Text = LogBox.Text + “@ Line=" + Str(n) + ", Ans=" + AnsH + ", MRS=" + Stable + CR

    Do      //Within a single instance of MRS
        Dim F(26) as String
        Dim Direction(26) as Integer
        LogBox.Text = LogBox.Text + "Floats: "
        j = 0
        Float = ""
        For i = 1 to NR
            If InStr(Stable, R(i)) = 0 then
                j = j + 1
                F(j) = R(i)                     //Floats
                Float = Float + F(j)
                LogBox.Text = LogBox.Text + F(j) + "(" + Str(InStr(Ref, F(j))) + "), "      //Display Floats.
            End
        Next
        NI_Float = j//Number of Float elements
        LogBox.Text = LogBox.Text + CR + "Initial direction value of Float(s) = "

        For j = 1 to NI_Float
            If InStr(AnsH, F(j)) < InStr(Ref, F(j)) then    //Judge direction from current to correct position.
                Sign = 1//Float should move right
            Else
                Sign = -1                       //Float should move left
            End
            Direction(j) = Sign * InStr(Ref, F(j))  //Direction value = magnitude with sign
            LogBox.Text = LogBox.Text + Str(Direction(j)) + ", "    //Display direction and magnitude.
        Next

        Do                                      //Within a stage of securing Option
            DirectionMax = -9999
```

```
DistanceMaxE = ""

For j = 1 to NI_Float
    If Direction(j) > DirectionMax then
        DirectionMax = Direction(j)
        DistanceMaxE = F(j)          //Find Float with maximal direction value.
    End
Next
LogBox.Text = LogBox.Text + CR + "Float with maximal direction value: " + DistanceMaxE + CR

If InStr(Ref, DistanceMaxE) < InStr(Ref, Left(Stable, 1)) Or Stable = "" then
                                     //In case Float should come to the front or Stable is empty,
    NewStable = DistanceMaxE + Stable              //Attach E to the front of Stable.
Else
    For j = 1 to Len(Stable)
        If InStr(Ref, DistanceMaxE) > InStr(Ref, Mid(Stable, j, 1)) then                  //Otherwise
            NewStable = Mid(Stable, 1, j) + DistanceMaxE + Mid(Stable, j+1, Len(Stable))
                                     //Insert Float in Stable in correct order.
        End
    Next
End
LogBox.Text = LogBox.Text + “% Temporary new Stable = " + NewStable + CR

Target = ""                          //Is a sequence of Ans excluding DistanceMaxE.
For i = 1 to NR
    TestStable(i) = ""
    If Mid(AnsH, i, 1) <> DistanceMaxE then        //If element is not DistanceMaxE,
        Target = Target + Mid(AnsH, i, 1)   //Combine Answer elements except DistanceMaxE.
    End
Next

LogBox.Text = LogBox.Text + "Target options:" + CR
For i = 1 to NR
    Option(i) = Mid(Target, 1, i-1) + DistanceMaxE + Mid(Target, i, NR-1)
                                     //Varieties of combinations of Target and E
    LogBox.Text = LogBox.Text + Option(i) + ", "   //Display target options.
Next i

LogBox.Text = LogBox.Text + CR + "Linearity test segments (=value):" + CR
For i = 1 to NR
    For j = 1 to Len(Option(i))
        H = Mid(Option(i), j, 1)
        If InStr(NewStable, H) <> 0 then
            TestStable(i) = TestStable(i) + H      //Correspond TestStable to Option.
        End
    Next
Next

//Choose Option with max linearity in TestStable (primary step).
For i = 1 to NR
    Linearity = 0
    For j = 1 to NR
        For k = 1 to NR-j
            If InStr(Ref, Mid(TestStable(i), j, 1)) < InStr(Ref, Mid(TestStable(i), j+k, 1)) then
                                     //Compare linearity within TestStable as to the correct order when
                                     TestStable has 2+ elements.
                Linearity = Linearity + NR - k
                                     //Place differentially weighted score for correct linearity; eg. NR-1 to
                                     adjacent pair, NR-2 to pair with 1 leap.
            End
        Next
    Next
    LogBox.Text = LogBox.Text + TestStable(i) + "=" + Str(Linearity) + ", "  //Display confirmed TestStable &
                                     Linearity value.
    LinearityScore(i) = Linearity
```

```
Next

LinearityMax = 0
For i = 1 to NR
    If LinearityScore(i) >= LinearityMax then
                                    //LineariltyMax will be the last option, including the case where the
                                    number of elements in TestStable =1.
        LinearityMax = LinearityScore(i)    //Maximal Linearity score
        ReturnPosition = i                  //Position of displacement in Target
    End
Next

j = 0
For i = 1 to NR
    If LinearityScore(i) = LinearityMax then
        j = j + 1                           //Count number of max Linearity score
        LinearityMaxOption(i) = Option(i)
                                    //Find the Option with maximal Linearity score for TestSegment.
    End
Next

//Choose Option with max linearity if there are multiple max values of linerarity in TestStable (secondary
                                    step).
If j > 1 then
    LogBox.Text = LogBox.Text + CR + "Linearity test segments (2nd step) (=value):" + CR

    For i = 1 to NR
        Linearity = 0
        If LinearityScore(i) = LinearityMax then
            For j = 1 to NR
                For k = 1 to NR-j
                    If InStr(Ref, Mid(LinearityMaxOption(i), j, 1)) < InStr(Ref, Mid(LinearityMaxOption(i),
                        j+k, 1)) then
                                    //Compare linearity within LinearityMaxOption as to the correct
                                    order when TestStable has 2+ elements.
                        Linearity = Linearity + NR - k
                                    //Place differentially weighted score for correct linearity; eg. NR-1 to
                                    adjacent pair, NR-2 to pair with 1 leap.
                    End
                Next
            Next
            LogBox.Text = LogBox.Text + LinearityMaxOption(i) + "=" + Str(Linearity) + ", "
                                    //Display confirmed TestStable & Linearity value.
            LinearityScore(i) = Linearity
        End
    Next

    LinearityMax = 0
    For i = 1 to NR
        If LinearityScore(i) >= LinearityMax then
                                    //LineariltyMax will be the last option, including the case where the
                                    number of elements in TestStable =1.
            LinearityMax = LinearityScore(i)        //Maximal Linearity score
            ReturnPosition = i      //Position of displacement in Target
        End
    Next
End

BestOption = LinearityMaxOption(ReturnPosition)
LogBox.Text = LogBox.Text + CR + "Best Stable = " + TestStable(ReturnPosition) + " (" + Str(ReturnPosition)
    + ")"
LogBox.Text = LogBox.Text + CR + "Best Option = " + BestOption + " (" + Str(ReturnPosition) + ")"

//Crossing constraint: Moving left does not allow crossing over younger elements; Moving right does not
                                    allow crossing over older elements.
```

```
388:    LogBox.Text = LogBox.Text + CR + "DistanceMaxE = " + Str(DistanceMaxE) + "(" + Str(DirectionMax) + ")"
390:    EMovingDistance = InStr(BestOption, DistanceMaxE) - InStr(Ans, DistanceMaxE)   //Moving distance of
                DistanceMaxE with sign ...(d) (ie. +: right, -: left)
393:    LogBox.Text = LogBox.Text + CR + "Moving direction & distance = "
395:    LogBox.Text = LogBox.Text + Str(EMovingDistance)
396:
397:    For j = 1 to NI_Float
398:        FRankRel(j) = InStr(Ref, F(j)) - InStr(Ref, DistanceMaxE)
                                //Relative position of Floats in Ref ...(r) (ie. +: older, -: younger)
401:        FPos0Rel(j) = InStr(AnsH, F(j)) - InStr(AnsH, DistanceMaxE)
                                //Relative position of Floats in AnsH ...(0)
403:        FPos1Rel(j) = InStr(BestOption, F(j)) - InStr(BestOption, DistanceMaxE)
                                //Relative position of Floats in BestOption ...(1)
406:        FPosProd(j) = FPos0Rel(j) * FPos1Rel(j)         //Product of (0) and (1) ...(p)
408:    Next
409:
410:    LogBox.Text = LogBox.Text + CR + "FRankRel = "
411:    For j = 1 to NI_Float
412:        LogBox.Text = LogBox.Text + F(j) + "(" + Str(FRankRel(j)) + "), "
414:    Next
415:
416:    LogBox.Text = LogBox.Text + CR + "FPos0Rel = "
417:    For j = 1 to NI_Float
418:        LogBox.Text = LogBox.Text + Str(FPos0Rel(j)) + ", "
419:    Next
420:
421:    LogBox.Text = LogBox.Text + CR + "FPos1Rel = "
422:    For j = 1 to NI_Float
423:        LogBox.Text = LogBox.Text + Str(FPos1Rel(j)) + ", "
424:    Next
425:
426:    LogBox.Text = LogBox.Text + CR + "FPosProd = "
427:    For j = 1 to NI_Float
428:        LogBox.Text = LogBox.Text + Str(FPosProd(j)) + ", "
429:    Next
430:
431:    LogBox.Text = LogBox.Text + CR + "Crossing constraint for "
433:    CrossingFlag = 0
434:    For j = 1 to NI_Float
435:        LogBox.Text = LogBox.Text + CR + Str(F(j)) + ": "
436:        If FPosProd(j) < 0 then         //If Product (p) is negative (ie. Crossing has occurred),
438:            If FRankRel(j) * EMovingDistance > 0 then   //and if either both (r) & (d) are negative (ie. moving
                                left & being crossed over as younger) or both (r) and (d) are positive
                                (ie. moving right & being crossed over as older),
442:                CrossingFlag = 1        //declare constraint is positive and
444:                Direction(InStr(Float, DistanceMaxE)) = -9999
                                //reset Direction value of DistanceMaxE to smallest possible.
447:                LogBox.Text = LogBox.Text + "applied (positive value) "
449:                LogBox.Text = LogBox.Text + Str(FRankRel(j) * EMovingDistance)
451:            Else
452:                LogBox.Text = LogBox.Text + "void (negative value) "
453:                LogBox.Text = LogBox.Text + Str(FRankRel(j) * EMovingDistance)
455:            End
456:        Else
457:            LogBox.Text = LogBox.Text + "[not applicable]"
                                //If Product (p) is 0 or positive, crossing did not occur.
459:        End
460:    Next
461:
462:    If CrossingFlag = 1 then            //If illegal crossing has occurred, go back loop and choose a new
                                DistanceMaxE.
464:        LogBox.Text = LogBox.Text + CR + "DistanceMaxE (" + DistanceMaxE + ") has been rejected." + CR
466:    Else
467:        Exit                            //Otherwise exit loop and proceed to summary.
468:    End
```

```
                    Loop                                      //Within a stage of securing Option

                    DisplacementDistance = ReturnPosition - InStr(AnsH, DistanceMaxE)
                                                              //Magnitude of displacement with direction (sign)
                    CumReturnDist = CumReturnDist + Abs(DisplacementDistance)
                                                              //Cumulative magnitude of displacement as Distance.
                    LogBox.Text = LogBox.Text + CR + "Optimal sequence = " + BestOption + CR + "Displacement distance with
                          direction = " + Str(DisplacementDistance) + CR + "¶ Cum Distance=" + Str(CumReturnDist) + CR + CR

                    Stable = NewStable
                    AnsH = BestOption                         //Reset AnsH by replacing it with Option (after 1 displacement).

                    If AnsH = Ref then
                        CumDistance = CumReturnDist           //Record cumulative magnitude of displacement as Distance.
                        Exit
                    End
                    ReDim F(-1)
                    Redim Direction(-1)
                Loop                                          //End of Count distance for recovery within a single instance of MRS
            End

            MRSseq = Join(MRSe, "/")                          //Concatenate elements of array MRSe with delimiter "/".
            MRSseq = Mid(MRSseq, 2, Len(MRSseq))              //MRSseq is delineated by "/", eg. /AB/AC/BC.
            AnsList = AnsList + Ans + "/" + Str(Cycle) + "_" + Str(CumDistance) + "|" + MRSseq + “,"
                                                              //Create record.
        End

        Score.Text = Score.Text + Str(Cycle) + CR             //Output Score.
        If MRSseq = "" then
            HeadDelimiter = ""
        Else
            HeadDelimiter = "/"
        End
        MRS.Text = MRS.Text + HeadDelimiter + MRSseq + CR      //Output Alternative sequences starting with "/".
        Distance.Text = Distance.Text + Str(CumDistance) + CR  //Output Distance
        Refresh
        ProgressBar1.Value = n/NL_Ans * 100                   //Show progress.
    Next

    Now = New Date
    Fin = Now.TotalSeconds
    Lapse = Fin - Init
    SearchResult.Text = "Completed.  Time = " + Str(Lapse) + “ sec."          //Display lapse time.
```

Appendix 3: Xojo source code of Permutation

Permutation coding script
©AMMA Kazuo, 2016

```
Dim CR, Seed, E(10), H, Seq as String
Dim Prime(), NC, NCmax, i, j, CodeMin, TCodeProd,
HCodeProd, Mark, Count as Integer
Dim Init, Fin, Lapse As Double
Dim Now as Date

Index.Text = ""
CodeProduct.Text = ""
Permutation.Text = ""
SearchResult.Text = ""
Count = 0
Refresh
ProgressBar1.Value = 0

CR = EndOfLine.Macintosh                          //For Windows, change to "EndOfLine.Windows".
Seed = Reference.Text
NC = Len(Seed)
Prime = Array(1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29)    //Prepare primes (for up to 10 elements).  Add primes where
                                                  necessary.

Now = New Date
Init = Now.TotalSeconds                           //Record initial time.

CodeMin = 9999
For i = 1 to NC
    E(i) = Mid(Seed, i, 1)                        //Get individual element from Seed.
    If CodeMin > Asc(E(i)) then
        CodeMin = Asc(E(i))                       //Get the code of youngest element.
    End
Next

TCodeProd = 1
For i = 1 to NC
    TCodeProd = TCodeProd * Prime(Asc(E(i)) - CodeMin + 1)
                                                  //Product of primes as code in target sequence.
Next

NCmax = NC ^ NC
i = 1
Do                                                //Repeat all possible repeated permutations (= exponent of NC).
    Seq = ""
    HCodeProd = 1
    Mark = 0
    For j = 1 to NC
        H = E(Floor((i - 1)/NC^(j - 1) Mod NC) + 1)     //Get element for position j.
        If InStr(Seq, H) <> 0 then                //Check overlap.
            Mark = 1
            Exit For
        Else
            Seq = Seq + H                         //Combine test elements
            HCodeProd = HCodeProd * Prime(Asc(H) - CodeMin + 1)     //Product of primes as code in test sequence.
        End
    Next

    If Mark = 0 then
        If HCodeProd = TCodeProd then             //If cubic sum of test sequence matches cubic sum of target,
            Count = Count + 1
            If Count = 1 then
                CodeProduct.Text = CodeProduct.Text + Str(HCodeProd) + CR
            Else
                If NC < 6 then                            //Displays index only when less than 6.
                    Index.Text = Index.Text + Str(i) + CR
                End
            End
            Permutation.Text = Permutation.Text + Seq + ","    //Record sequence as permutation.
        End
    End

    ProgressBar1.Value = i/NCmax * 100
    If i = NCmax then
        CodeProduct.Text = CodeProduct.Text + "Count = " + Str(Count)
        Exit Do
    Else
        i = i + 1
    End
Loop

Now = New Date
Fin = Now.TotalSeconds
Lapse = Fin - Init
SearchResult.Text = "Completed.  Time =" + Str(Lapse) + " sec."            //Display lapse time.
```