Investigation of some probabilistic properties of the problem

Yurii Lahodiuk
yura.lagodiuk@gmail.com

May 2017

Contents

1 Summary

The goal of the problem is to generate a "properly-placed" sequence of n characters (there are two instances of each character). A properly-placed sequence of characters is a such sequence, where amounts of characters between pairs of the same characters are equal to the indices of characters in the alphabetic order (e.g.: distance between characters "A" is 1, distance between characters "B" is 2, and so on).

In scope of this problem, I have revealed the following facts:

- The sequence of n pairs of characters doesn't require extra space-character in case if:
  - either: $4|n$
  - or: $4|(n-3)$

  Otherwise an extra space-character (e.g. hyphen) is needed. For more details (and proof) check sections 2.1 and 2.2.

- I have analysed the possibility to obtain a properly-placed sequence as a result of a random permutation of a sequence of n pairs of characters. I have considered a random variable ξ, which denotes an amount of correctly placed pairs of characters (the distances between the characters of these pairs are equal to the indices of these characters). In scope of this exercise I have derived an expected amount and a variance of ξ:
  - $\mathbb{E}[\xi] = \frac{3n-3}{4n-2}$
  - $\mathrm{Var}(\xi) = \frac{3}{4} + \frac{4}{3n} + \frac{4}{n-1} - \frac{127}{96\cdot(2n-3)} - \frac{363}{32\cdot(2n-1)} - \frac{9}{16\cdot(2n-1)^2}$

  For more details (and derivation) check sections 3.1 and 3.2. The values obtained via derived formulas of expectation and variance are consistent to the values obtained via computer simulation (for more details check a section 3.3).

- Using Markov's and Chebyshev's inequalities I have derived weak bounds for the probability to obtain a properly-placed sequence as a result of a random permutation:
  - $P(\xi \geqslant n) \leqslant \frac{3n-3}{4n^2-2n}$
  - $P(\xi \geqslant n) \leqslant \frac{16+5n-160n^2+215n^3-116n^4+24n^5}{(n-1)\cdot n\cdot(2n-3)\cdot(4n^2-5n+3)^2}$

  For more details check a section 3.4.

- It turns out, that to some extent, it is viable to use a Poisson distribution as a rough approximation of the distribution of values of the random variable ξ. The event rate of the given Poisson distribution is $\lambda = \frac{3}{4}$. The probability of occurrence of k correctly placed pairs is:
  - $P(\xi = k) = e^{-\frac{3}{4}} \cdot \left(\frac{3}{4}\right)^k \cdot \frac{1}{k!}$

  For more details and results of experimental evaluation (using computer simulation) check a section 3.5.

- I have implemented a Simulated Annealing algorithm for generation of the proper placement. The code can be found inside appendix A.4.

- One of the possible proper placements for $n = 26$ is (obtained using Simulated Annealing algorithm): Q-NDSTGYDZROFXGPENQFJVEUSWTOLRCJPYCKZMXBILBHVUAKAWIMH

Below is presented a table of the correct placements from $n = 1$ up to $n = 26$, obtained via Simulated Annealing algorithm:

n	Placement
1	A-A
2	ABA-B
3	CABACB
4	DACABDCB
5	ECADACEBD-B
6	DBF-BDECAFACE
7	AEAFCGDECBFDBG
8	BFGBDHECFDGCEAHA
9	B-HBEIFAGAEHDFCIGDC
10	CDEJCID-EHFAGAJIBFHBG
11	AEAFHIJEKBFGBHCIDJCGKD
12	GBHJBFILGEKHFCJEICDALAKD
13	JACAEICLMDEJKGDIBHFBLGM-KFH
14	KNHIJ-LBEMBHKIEJNCFLGCDMAFADG
15	ADAOHEDNFJGELHMFIKGOJBNCBLICMK
16	PGAJAIENLGOFEKJIMPFDHLNCDKOCBHMB
17	FDO-JKDFPCQBNCBJLKOMAIAHEPGNQLEIHMG
18	GELMAQAEGCPRKCJLIMN-OFHQKJIPFDRHBNDBO
19	JNKAHAMOLPFJRHKSNFQBMLBOGDPIECDRGCESQI
20	HKEMRFDNEHSDFKTOQMILCPNRCGJAIASOLGQTBJPB
21	HEISFNREQHBFIBMTP-KUNGLSORQDMGKJDPCLTACAOUJ
22	AIADLGCNDMCIRGVQULPJTENMSHOEKBJRBQHPFVU-KTOFS
23	CUQICVNJGPRWLIOFGTJMQNFUSLPKVROHAMAWDETKHDBESB
24	FNCJGDCFQVDWGMJTNSAXARPUEOQMBLEBVIKWTSHPROLIXUKH
25	UHCKGVCLEOHXGSEKM-YWLTUNJODRVQMDPSFJXANAIFTWYBRQBPI
26	Q-NDSTGYDZROFXGPENQFJVEUSWTOLRCJPYCKZMXBILBHVUAKAWIMH

2 With or without extra space-character?

2.1 Exploration of small instances of the problem using a brute-force approach

In order to figure out, in which cases an extra space-character is required to solve the problem, I have checked the small instances of the problem using a brute-force approach (based on the backtracking technique), which can be summarized in a following recurrence relation:

$$F(\mathbb{X}, d) = \begin{cases} True, & if\ \mathbb{X} = \emptyset \\ False, & if\ \forall x, y \in \mathbb{X}, (y - x) \neq d + 1 \\ \displaystyle\bigvee_{(x,y) \in \{(x,y) | x,y \in \mathbb{X}, (y-x) = d+1\}} F\left(\mathbb{X} \setminus \{x, y\}, d + 1\right), & otherwise \end{cases} \tag{1}$$

where: $\mathbb{X}$ – a set of non-occupied positions
d – a distance (gap) between the current pair of characters
$\bigvee$ – denotes a logical disjunction between multiple expressions (similar to the capital-sigma notation)

For different amounts of pairs of unique characters n - either $F(\{1, 2, ..., 2n\}, 1)$ or $F(\{1, 2, ..., (2n+1)\}, 1)$ is $True$. The Java implementation of the brute-force code can be found inside appendix A.1.

The analysis of small instances of the problem leads to the following observation:

n (pairs of unique characters)	1	2	3	4	5	6	7	8	9	10
no extra space-character needed			✓	✓			✓	✓		
requires extra space-character	✓	✓			✓	✓			✓	✓

Based on this observation, I have conjectured, that in case if $4|n$ or $4|(n-3)$ the n pairs of unique characters can be arranged into the sequence of length $2n$, otherwise an extra space-character is needed.

2.2 In which cases the problem can be solved without extra space-character?

Let's consider an arrangement of n pairs of unique characters into a sequence without an extra space-character (length is $2n$), which satisfies conditions of the problem.

For the sake of convenience, let's establish a mapping from characters to natural numbers:

A	1
B	2
C	3
...	
Z	26

Every character can be associated with two numbers: x_i and y_i, which represent the indices of the first and the second occurrence of the character in the sequence (i - is a number, which corresponds to the character). According to the requirements of the problem, we can define the system of following constraints:

$$\begin{cases} y_i = x_i + i + 1, \forall i \in \{1, ..., n\} \\ x_i, y_i \in \{1, ...2n\}, \forall i \in \{1, ..., n\} \\ \{x_1, x_2, ..., x_n\} \cup \{y_1, y_2, ..., y_n\} = \{1, 2, 3, ..., 2n\} \end{cases} \tag{2}$$

The third and the second constraints mean, that all values of x_i and y_i must be unique.

Having the third constraint in mind, let's calculate the sum of all x_i and y_i:

$$\sum_{i=1}^{n} x_i + \sum_{i=1}^{n} y_i = \sum_{i=1}^{2n} i = n \cdot (2n+1) \tag{3}$$

On the other hand, having the first constraint in mind, the sum of all x_i and y_i can be represented in a following way:

$$\sum_{i=1}^{n} x_i + \sum_{i=1}^{n} y_i = \sum_{i=1}^{n} x_i + \sum_{i=1}^{n} (x_i + i + 1) = \sum_{i=1}^{n} (2 \cdot x_i + i + 1) = \left(2 \cdot \sum_{i=1}^{n} x_i\right) + \frac{n^2 + 3n}{2} \tag{4}$$

Hence, from equations (3) and (4) we have:

$$\left(2 \cdot \sum_{i=1}^{n} x_i\right) + \frac{n^2 + 3n}{2} = n \cdot (2n+1) \iff \sum_{i=1}^{n} x_i = \frac{3n^2 - n}{4} \tag{5}$$

As far as the sum of all x_i is integer, then $\frac{3n^2-n}{4}$ must be integer as well. It is possible iff $4|n$ or $4|(n-3)$, which is consistent with observations based on the small instances of the problem.

Hence, in case if $4|n$ or $4|(n-3)$ the n pairs of unique characters can be arranged into the sequence of length $2n$ (with respect to the constraints of the problem), otherwise an extra space-character is needed.

3 Random permutations of the sequence of n pairs of characters

Let's explore the possibility to obtain a correct placement (which will satisfy all constraints) as a result of the random permutation of the sequence of n pairs of characters (let's consider the sequence without an extra space-character - so, the total length is $2n$).

3.1 Expectation of the amount of correct placements

Let's consider a discrete valued random variable ξ, which represents an amount of correct placements of pairs of characters (which satisfy the first constraint from the system of constraints (2)). In order to proceed, let's represent ξ as a sum of indicator random variables:

$$\xi = \xi_1 + \xi_2 + ... + \xi_n \tag{6}$$

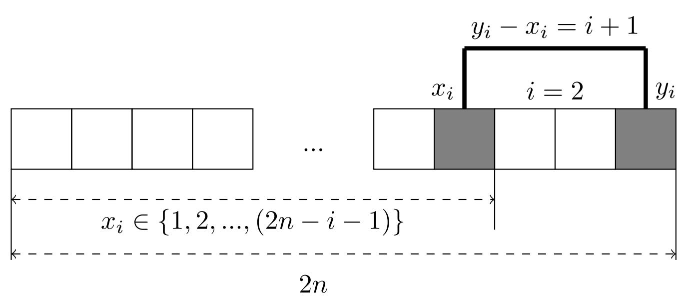
where ξ_i is a random variable, which indicates - whether the pair of i-th characters placed correctly (based on notation of x_i and y_i from the section (2.2)):

$$\xi_i = \begin{cases} 1, \ if \ y_i - x_i = i + 1 \\ 0, \ otherwise \end{cases} \quad , \forall i \in \{1, 2, ..., n\} \tag{7}$$

Due to the linearity of the expectation:

$$\mathbb{E}[\xi] = \mathbb{E}[\xi_1 + \xi_2 + ... + \xi_n] = \sum_{i=1}^{n} \mathbb{E}[\xi_i] = \sum_{i=1}^{n} P\left(y_i - x_i = i + 1\right) \tag{8}$$

For every pair of characters with index $i \in \{1, 2, ..., n\}$ out of $\binom{2n}{2}$ possible positions for x_i and y_i - there are only $(2n - i - 1)$ positions, which satisfy constraint $y_i = x_i + i + 1$. For example:



Hence:

$$P(y_i - x_i = i + 1) = \frac{2n - i - 1}{\binom{2n}{2}}, \ \forall i \in \{1, 2, ..., n\} \tag{9}$$

So, expectation of the amount of correct placements can be represented as:

$$\mathbb{E}[\xi] = \sum_{i=1}^{n} \frac{2n - i - 1}{\binom{2n}{2}} = \frac{3n^2 - 3n}{2 \cdot \binom{2n}{2}} = \frac{3n - 3}{4n - 2} \tag{10}$$

3.2 Variance of the amount of correct placements

Let's calculate the variance of ξ:

$$\mathrm{Var}(\xi) = \mathbb{E}[\xi^2] - (\mathbb{E}[\xi])^2 = \mathbb{E}\left[\xi_1^2 + ... + \xi_n^2 + \sum_{i \neq j} \xi_i \cdot \xi_j\right] - (\mathbb{E}[\xi])^2 \tag{11}$$

As far, as ξ_i is an indicator random variable, then:

$$\xi_i^2 = \xi_i = \begin{cases} 1, \ if \ y_i - x_i = i + 1 \\ 0, \ otherwise \end{cases} \quad , \forall i \in \{1, 2, ..., n\} \tag{12}$$

We can make use of the linearity property of expectation again:

$$\mathrm{Var}(\xi) = \mathbb{E}\left[\xi_1^2 + ... + \xi_n^2\right] + \mathbb{E}\left[\sum_{i \neq j} \xi_i \cdot \xi_j\right] - (\mathbb{E}[\xi])^2 = \mathbb{E}[\xi] + \mathbb{E}\left[\sum_{i \neq j} \xi_i \cdot \xi_j\right] - (\mathbb{E}[\xi])^2 \tag{13}$$

However, we need to calculate somehow the value of: $\mathbb{E}\left[\sum_{i \neq j} \xi_i \cdot \xi_j\right]$. Due to the symmetry, we can consider only the cases, when $i < j$:

$$\mathbb{E}\left[\sum_{i \neq j} \xi_i \cdot \xi_j\right] = 2 \cdot \mathbb{E}\left[\sum_{i < j} \xi_i \cdot \xi_j\right] \tag{14}$$

It turns out, that $\xi_i \cdot \xi_j$ is also an indicator random variable, which is defined in a following way:

$$\xi_i \cdot \xi_j = \begin{cases} 1, \ if \ y_i - x_i = i + 1 \ \wedge \ y_j - x_j = j + 1 \\ 0, \ otherwise \end{cases} \quad , \forall i, j \in \{1, 2, ..., n\}, \ i \neq j \tag{15}$$

Hence, we have:

$$\mathbb{E}\left[\sum_{i < j} \xi_i \cdot \xi_j\right] = \sum_{i < j} P\left(\xi_i \cdot \xi_j = 1\right) \tag{16}$$

$P\left(\xi_i \cdot \xi_j = 1\right)$ is equal to a joint probability $P\left(y_i - x_i = i + 1, \ y_j - x_j = j + 1\right)$:

$$P\left(\xi_i \cdot \xi_j = 1\right) = P\left(y_i - x_i = i + 1, \ y_j - x_j = j + 1\right) = \frac{M_{i,j}}{\binom{2n}{2} \cdot \binom{2n-2}{2}} \tag{17}$$

where: $M_{i,j}$ – number of ways to place correctly the i-th and j-th pair of characters
$\binom{2n}{2} \cdot \binom{2n-2}{2}$ – number of ways to choose values for: x_i, y_i, x_j and y_j

Let's calculate $M_{i,j}$. The set of all correct relative arrangements of x_i, y_i, x_j and y_j can be divided into three disjoint sets with *interleaved* ($\mathbb{I}_{ij}$), *nested* ($\mathbb{N}_{ij}$) and *separate* ($\mathbb{S}_{ij}$) relative placements. Hence:

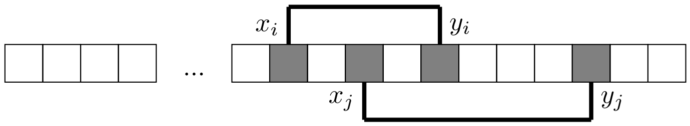
$$M_{i,j} = |\mathbb{I}_{ij} \sqcup \mathbb{N}_{ij} \sqcup \mathbb{S}_{ij}| = |\mathbb{I}_{ij}| + |\mathbb{N}_{ij}| + |\mathbb{S}_{ij}| \tag{18}$$

1. Interleaved arrangements:

- $x_i < x_j < y_i < y_j$
- $x_j < x_i < y_j < y_i$

Without loss of generality let's consider only a first variant: $x_i < x_j < y_i < y_j$ (the counting for the second variant will be exactly the same - so, we will just multiple an obtained result by 2).

In order to produce the correct interleaved placement, we need to choose the proper positions of the leftmost and rightmost occupied cells (x_i and y_j), and afterwards we can solely define the positions of y_i and x_j inside:



An amount of cells between the leftmost and rightmost occupied cells is:

$$d_{ij} = y_j - x_i - 1 = (x_j + j + 1) - (y_i - i - 1) - 1 = i + j + 2 - (y_i - x_j + 1) = i + j + 2 - k \quad (19)$$

where: $\quad k = y_i - x_j + 1$

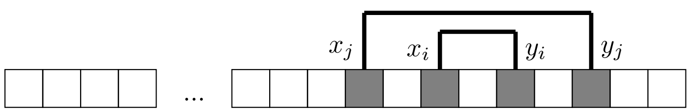
As far as $y_i \in \{x_j + 1, x_j + 2, ..., x_j + i\}$, then $k \in \{2, 3, ..., i + 1\}$. Hence the total amount of all possible interleaved placements is:

$$|\mathbb{I}_{ij}| = 2 \cdot \sum_{k=2}^{i+1} (2n - d_{ij} - 1) = 2 \cdot \sum_{k=2}^{i+1} (2n - 3 - i - j + k) \quad (20)$$

2. Nested arrangement:

   - $x_j < x_i < y_i < y_j$

In order to produce the correct nested placement, we need to choose the proper positions of the leftmost and rightmost occupied cells (x_j and y_j), and afterwards we should choose the proper positions for x_i and y_i inside:



An amount of ways to choose the proper positions for x_j and y_j is: $(2n - j - 1)$. Now, we need to choose the proper positions for x_i and y_i inside, and an amount of ways to do so is: $(j - i - 1)$. Hence:

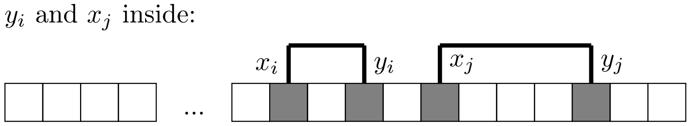
$$|\mathbb{N}_{ij}| = (2n - j - 1) \cdot (j - i - 1) \quad (21)$$

3. Separate arrangements:

   - $x_i < y_i < x_j < y_j$
   - $x_j < y_j < x_i < y_i$

Without loss of generality let's consider only a first variant: $x_i < y_i < x_j < y_j$ (the counting for the second variant will be exactly the same - so, we will just multiple an obtained result by 2).

In order to produce the correct interleaved placement, we need to choose the proper positions of the leftmost and rightmost occupied cells (x_i and y_j), and afterwards we can solely define the positions of y_i and x_j inside:



An amount of cells between the leftmost and rightmost occupied cells is:

$$d_{ij} = y_j - x_i - 1 = (x_j + j + 1) - (y_i - i - 1) - 1 = i + j + 2 + (x_j - y_i - 1) = i + j + 2 + k \quad (22)$$

where: $k = x_j - y_i - 1$

As far as $x_j \in \{y_i + 1, y_i + 2, ..., y_i + (2n - i - j - 2)\}$, then $k \in \{0, 1, ..., 2n - 3 - i - j\}$. Hence the total amount of all possible interleaved placements is:

$$|\mathbb{S}_{ij}| = 2 \cdot \sum_{k=0}^{2n-i-j-3} (2n - d_{ij} - 1) = 2 \cdot \sum_{k=0}^{2n-i-j-3} (2n - 3 - i - j - k) \quad (23)$$

The tidy analysis shows, that in case if $i + j > 2n - 3$ any separate arrangement of i-th and j-th pairs of characters is impossible within $2n$ slots. Hence, the complete formula for $|\mathbb{S}_{ij}|$ is:

$$|\mathbb{S}_{ij}| = \begin{cases} 2 \cdot \sum_{k=0}^{2n-i-j-3} (2n - 3 - i - j - k), & if\ i + j \leqslant 2n - 3 \\ \\ 0, & if\ i + j > 2n - 3 \end{cases} \quad (24)$$

So, finally, using equations (20), (21) and (24) - we can express $M_{i,j}$ in terms of i, j and n:

$$M_{i,j} = |\mathbb{I}_{ij}| + |\mathbb{N}_{ij}| + |\mathbb{S}_{ij}| = \\ = \begin{cases} \left(2 \cdot \sum_{k=2}^{i+1} (2n - 3 - i - j + k)\right) + (2n - j - 1) \cdot (j - i - 1) + \left(2 \cdot \sum_{k=0}^{2n-i-j-3} (2n - 3 - i - j - k)\right), & if\ i + j \leqslant 2n - 3 \\ \\ \left(2 \cdot \sum_{k=2}^{i+1} (2n - 3 - i - j + k)\right) + (2n - j - 1) \cdot (j - i - 1), & if\ i + j > 2n - 3 \end{cases} \quad (25)$$

After simplification, we can obtain the closed form for $M_{i,j}$:

$$M_{i,j} = \begin{cases} -2n \cdot (i + j + 6) + ij + 3i + 5j + 4n^2 + 7, & if\ i + j \leqslant 2n - 3 \\ \\ -i^2 + 2n \cdot (i + j - 1) - ij - 2i - j^2 + 1, & if\ i + j > 2n - 3 \end{cases} \quad (26)$$

Now, we are almost ready to calculate $\mathbb{E}\left[\sum_{i<j} \xi_i \cdot \xi_j\right]$, using equations (16) and (17):

$$\mathbb{E}\left[\sum_{i<j} \xi_i \cdot \xi_j\right] = \sum_{i<j} P\left(y_i - x_i = i + 1,\ y_j - x_j = j + 1\right) = \frac{\sum_{i<j} M_{i,j}}{\binom{2n}{2} \cdot \binom{2n-2}{2}} \quad (27)$$

In order to calculate the sum $\sum_{i<j} M_{i,j}$ let's consider separately the cases where $i + j \leqslant 2n - 3$ and $i + j > 2n - 3$:

$$\sum_{i<j} M_{i,j} = \left(\sum_{\substack{i<j,\\ i+j\leqslant 2n-3}} M_{i,j} \right) + \left(\sum_{\substack{i<j,\\ i+j>2n-3}} M_{i,j} \right) = \\ = \left(\sum_{j=2}^{n-1} \sum_{i=1}^{j-1} M_{i,j} + \sum_{i=1}^{n-3} M_{i,n} \right) + (M_{n-1,n} + M_{n-2,n}) \tag{28}$$

After simplification, we can obtain the closed form:

$$\sum_{i<j} M_{i,j} = \frac{1}{8} \cdot (n-2) \cdot (9n^3 - 28n^2 + 27n + 8) \tag{29}$$

According to equations (14), (27) and (28) we can obtain the value of $\mathbb{E}\left[\sum_{i\neq j} \xi_i \cdot \xi_j\right]$:

$$\mathbb{E}\left[\sum_{i\neq j} \xi_i \cdot \xi_j\right] = 2 \cdot \frac{\frac{1}{8} \cdot (n-2) \cdot (9n^3 - 28n^2 + 27n + 8)}{\binom{2n}{2} \cdot \binom{2n-2}{2}} \tag{30}$$

So, using equations (10), (13) and (14) we can calculate the variance:

$$\mathrm{Var}(\xi) = \frac{3n-3}{4n-2} + \frac{(n-2) \cdot (9n^3 - 28n^2 + 27n + 8)}{4 \cdot \binom{2n}{2} \cdot \binom{2n-2}{2}} - \left(\frac{3n-3}{4n-2}\right)^2 \\ = \frac{3}{4} + \frac{4}{3n} + \frac{4}{n-1} - \frac{127}{96 \cdot (2n-3)} - \frac{363}{32 \cdot (2n-1)} - \frac{9}{16 \cdot (2n-1)^2} \tag{31}$$

3.3 Experimental evaluation

In order to verify derived formulas for expectation and variance, I have developed an application for simulation of random permutations on the sequences with different amounts of pairs of unique characters n (for simplicity in every case the length of a sequence was $2n$). For every n - application simulates $5 \cdot 10^5$ random permutations, and calculates the average amount (and variance) of correctly placed pairs of characters.

The code of a Java application for simulation is inside appendix A.2.

Below is presented a table with comparison of the expectation and variance obtained experimentally and calculated, based on the derived formulas. As you can see, the values obtained via derived formulas of expectation and variance are consistent to the values obtained via computer simulation.

n (amount of pairs of characters)	average amount of correct placements	theoretical expectation	experimental variance	theoretical variance
2	0.502	0.500	0.250	0.250
3	0.601	0.600	0.462	0.462
4	0.643	0.643	0.520	0.520
5	0.669	0.667	0.562	0.560
6	0.681	0.682	0.590	0.589
7	0.691	0.692	0.611	0.611
8	0.699	0.700	0.630	0.628
9	0.708	0.706	0.644	0.641
10	0.713	0.711	0.653	0.651
11	0.716	0.714	0.660	0.660
12	0.720	0.717	0.670	0.667
13	0.719	0.720	0.674	0.674
14	0.723	0.722	0.679	0.679
15	0.724	0.724	0.684	0.684
16	0.725	0.726	0.688	0.688
17	0.728	0.727	0.695	0.691
18	0.727	0.729	0.692	0.695
19	0.730	0.730	0.699	0.698
20	0.730	0.731	0.700	0.700
21	0.733	0.732	0.703	0.703
22	0.734	0.733	0.705	0.705
23	0.732	0.733	0.705	0.707
24	0.735	0.734	0.711	0.708
25	0.733	0.735	0.710	0.710
26	0.736	0.735	0.711	0.712
27	0.732	0.736	0.710	0.713
28	0.737	0.736	0.714	0.714
29	0.737	0.737	0.715	0.716
30	0.737	0.737	0.717	0.717
31	0.739	0.738	0.719	0.718
32	0.739	0.738	0.718	0.719
33	0.738	0.738	0.718	0.720
34	0.738	0.739	0.720	0.721
35	0.738	0.739	0.719	0.721
36	0.739	0.739	0.722	0.722
37	0.741	0.740	0.725	0.723
38	0.739	0.740	0.721	0.724
39	0.742	0.740	0.727	0.724
40	0.739	0.741	0.727	0.725
41	0.741	0.741	0.725	0.726
42	0.741	0.741	0.727	0.726
43	0.745	0.741	0.729	0.727
44	0.740	0.741	0.728	0.727
45	0.743	0.742	0.730	0.728

3.4 Probabilistic bounds

Having the explicit expressions for expectation and variance we can make use of a couple of probabilistic inequalities, in order to estimate the probability of obtaining the correct placement of all n pairs of characters:

- Markov's inequality:

$$P(\xi \geqslant n) \leqslant \frac{\mathbb{E}[\xi]}{n} = \frac{3n-3}{4n^2-2n} \tag{32}$$

- Chebyshev's inequality:

$$\begin{aligned} P(\xi \geqslant n) &= P\left(\xi - \mathbb{E}[\xi] \geqslant n - \mathbb{E}[\xi]\right) \leqslant \\ &\leqslant P\left(|\xi - \mathbb{E}[\xi]| \geqslant n - \mathbb{E}[\xi]\right) \leqslant \\ &\leqslant \frac{\operatorname{Var}(\xi)}{(n-\mathbb{E}[\xi])^2} = \frac{16+5n-160n^2+215n^3-116n^4+24n^5}{(n-1)\cdot n\cdot(2n-3)\cdot(4n^2-5n+3)^2} \end{aligned} \tag{33}$$

Unfortunately, these probabilistic bounds are still very weak.

3.5 Speculation around the Poisson distribution

The indicator random values of ξ_i are not independent, hence generally speaking, we can't model the distribution of values of ξ using the Poisson distribution.

Nevertheless, let's consider the values of $\mathbb{E}[\xi]$ and $\operatorname{Var}(\xi)$ for the large values of pairs n:

$$\lim_{n\to\infty} \mathbb{E}[\xi] = \lim_{n\to\infty} \frac{3n-3}{4n-2} = \frac{3}{4} \tag{34}$$

$$\lim_{n\to\infty} \operatorname{Var}(\xi) = \lim_{n\to\infty} \left[\frac{3}{4} + \frac{4}{3n} + \frac{4}{n-1} - \frac{127}{96\cdot(2n-3)} - \frac{363}{32\cdot(2n-1)} - \frac{9}{16\cdot(2n-1)^2}\right] = \frac{3}{4} \tag{35}$$

We can see, that $\lim_{n\to\infty} \mathbb{E}[\xi] = \lim_{n\to\infty} \operatorname{Var}(\xi)$. Hence, to some extent, I assume that it is still viable to use a Poisson distribution as a rough approximation of the distribution of values of the random variable ξ. This assumption is also based on the observation, that occurrence of the correctly placed pairs of characters is a rare event.

The event rate of the given Poisson distribution is $\lambda = \frac{3}{4}$. Hence, the probability of occurrence of k correctly placed pairs is:

$$P(\xi = k) = e^{-\lambda} \cdot \frac{\lambda^k}{k!} = e^{-\frac{3}{4}} \cdot \left(\frac{3}{4}\right)^k \cdot \frac{1}{k!} \tag{36}$$

Let's evaluate this approximation experimentally:

n	k	simulated $P(\xi = k)$	Poisson $P(\xi = k)$
20	0	0.47307	0.47237
20	1	0.36148	0.35427
20	2	0.13068	0.13285
20	3	0.02946	0.03321
20	4	0.00471	0.00623
20	5	0.00056	0.00093
20	6	0.00005	0.00012
20	7	0.00000	0.00001

n	k	simulated $P(\xi = k)$	Poisson $P(\xi = k)$
40	0	0.47407	0.47237
40	1	0.35732	0.35427
40	2	0.13153	0.13285
40	3	0.03100	0.03321
40	4	0.00533	0.00623
40	5	0.00068	0.00093
40	6	0.00008	0.00012
40	7	0.00000	0.00001

The code of a Java application for simulation is inside appendix A.3.

A Appendices

A.1 The code for Brute-force exploration of the problem

Below is presented a snippet of Java code, which implements the brute-force recurrence (1):

```java
// BruteForce.java
public class BruteForce {

   public static void main(String... args) {
      for (int alphabet_size = 1; alphabet_size < 11; alphabet_size++) {
         check(alphabet_size);
      }
   }
   static void check(int alphabet_size) {
      if (can_solve(new boolean[2 * alphabet_size], 1, alphabet_size)) {
         System.out.printf("%3d - no extra space needed %n", alphabet_size);

      } else if (can_solve(new boolean[2 * alphabet_size + 1], 1, alphabet_size)) {
         System.out.printf("%3d + requires extra space %n", alphabet_size);
      }
   }
   static boolean can_solve(boolean[] occupied, int distance, int alphabet_size) {
      if (distance == alphabet_size + 1) {
         return true; // All positions are occupied, hence the problem can be solved
      }
      for (int pos = 0; pos < occupied.length - distance - 1; pos++) {
         // Iterate over all available positions
         if (!(occupied[pos] || occupied[pos + distance + 1])) {
            occupied[pos] = true;
            occupied[pos + distance + 1] = true;
            if (can_solve(occupied, distance + 1, alphabet_size)) {
               return true; // Problem can be solved
            }
            // Backtracking
            occupied[pos] = false;
            occupied[pos + distance + 1] = false;
         }
      }
      return false; // No solutions found
   }
}
```

A.2 The code for experimental evaluation of the formulas for expectation and variance

```
// ExpectationVarianceCheck.java
import java.util.*;

public class ExpectationVarianceCheck {

   public static void main(String[] args) {
      Random rnd = new Random(1);
      for (int size = 2; size < 46; size++) {
         evaluate(size, rnd);
      }
   }

   private static void evaluate(int size, Random rnd) {

      int trials_count = 500000;

      int correct_placements_cnt = 0;
      int correct_placements_cnt_sqr = 0;
      for (int i = 0; i < trials_count; i++) {
         List<Character> characters = random_placement(size, rnd);
         int correct = calc_correct_placements(characters);
         correct_placements_cnt += correct;
         correct_placements_cnt_sqr += correct * correct;
      }

      double avg_correct_placements =
            (double) correct_placements_cnt / trials_count;
      double experimental_variance =
            (double) correct_placements_cnt_sqr / trials_count - Math.pow(avg_correct_placements,
                2);

      System.out.printf("%d %3.3f %3.3f %3.3f %3.3f %n",
            size, avg_correct_placements, expectation(size),
            experimental_variance, variance(size));
   }

   private static double expectation(int n) {
      return (3.0 * n - 3) / (4 * n - 2);
   }

   private static double variance(int n) {
      return 3.0 / 4 + 4.0 / (n - 1) + 4.0 / (3 * n)
            - 127.0 / (96 * (2 * n - 3)) - 363.0 / (32 * (2 * n - 1))
            - 9.0 / (16 * Math.pow(2 * n - 1, 2));
   }

   private static int calc_correct_placements(List<Character> characters) {
      Map<Character, Integer> first_occurrence = new HashMap<>();
      int correct = 0;
      for (int i = 0; i < characters.size(); i++) {
         char c = characters.get(i);
         if (first_occurrence.get(c) == null) {
```

```
                first_occurrence.put(c, i);
            } else {
                int firstOccurrencePos = first_occurrence.get(c);
                if (i - firstOccurrencePos - 1 == c - 'A' + 1) {
                    correct++;
                }
            }
        }
        return correct;
    }

    private static List<Character> random_placement(int size, Random rnd) {
        List<Character> characters = new ArrayList<>();
        for (int i = 0; i < size; i++) {
            char curr_char = (char) ('A' + i);
            characters.add(curr_char);
            characters.add(curr_char);
        }
        Collections.shuffle(characters, rnd);
        return characters;
    }
}
```

A.3 The code for experimental evaluation of the Poisson approximation

```java
// PoissonDistributionCheck.java
import java.util.*;

public class PoissonDistributionCheck {

   public static void main(String[] args) {
      Random rnd = new Random(1);
      for (int size = 20; size < 41; size += 5) {
         evaluate(size, rnd);
         System.out.println();
      }
   }

   private static void evaluate(int size, Random rnd) {
      int trialsCount = 500000;
      Map<Integer, Integer> correct_placements_count = new TreeMap<>();
      for (int i = 0; i < trialsCount; i++) {
         List<Character> characters = random_placement(size, rnd);
         int correct = calc_correct_placements(characters);
         int count = correct_placements_count.getOrDefault(correct, 0);
         correct_placements_count.put(correct, count + 1);
      }
      for (int correct : correct_placements_count.keySet()) {
         int simulated_count = correct_placements_count.get(correct);
         double simulated_prob = (double) simulated_count / trialsCount;
         System.out.printf("%d %d %1.5f %1.5f %n", size, correct, simulated_prob,
              poisson_prob(correct));
      }
   }

   private static double poisson_prob(int correct) {
      long fact = factorial(correct);
      return Math.exp(-0.75) * Math.pow(0.75, correct) / fact;
   }

   private static long factorial(int n) {
      long fact = 1;
      for (int i = 1; i <= n; i++) {
         fact *= i;
      }
      return fact;
   }

   private static int calc_correct_placements(List<Character> characters) {
      Map<Character, Integer> first_occurrence = new HashMap<>();
      int correct = 0;
      for (int i = 0; i < characters.size(); i++) {
         char c = characters.get(i);
         if (first_occurrence.get(c) == null) {
            first_occurrence.put(c, i);
         } else {
            int firstOccurrencePos = first_occurrence.get(c);
            if (i - firstOccurrencePos - 1 == c - 'A' + 1) {
```

```java
                correct++;
            }
        }
    }
    return correct;
}

private static List<Character> random_placement(int size, Random rnd) {
    List<Character> characters = new ArrayList<>();
    for (int i = 0; i < size; i++) {
        char curr_char = (char) ('A' + i);
        characters.add(curr_char);
        characters.add(curr_char);
    }
    Collections.shuffle(characters, rnd);
    return characters;
}
}
```

A.4 Simulated Annealing solution

```java
// SimulatedAnnealingSolution.java
import java.util.*;

public class SimulatedAnnealingSolution {

   private static final boolean DEBUG_OUTPUT = false;

   public static void main(String[] args) {
      for (int size = 1; size <= 100; size++) {
         int[] solution = findOptimalSolution(size);
         int error = calcError(solution);
         if (error > 0.1) {
            break;
         }
         System.out.println(size + "\t" + error + "\t" + solutionToString(solution));
      }
   }

   private static int[] findOptimalSolution(int size) {
      return findOptimalSolution(size, 0.1, 1000, 0.01, 0.999, 100, new Random(1));
   }

   private static int[] findOptimalSolution(
         int size,
         double minEnergy, // minimal value of energy (termination criteria)
         double initialTemperature, // initial temperature
         double minTemperature, // minimal value of temperature (termination criteria)
         double temperatureDecreaseRatio, // (decreasing geometric progression)
         int numberOfTrials, // per iteration
         Random random) {

      // Initialize current solution
      int[] currentSolution = generateInitialSolution(size);
      int sequenceLength = currentSolution.length;

      // Initialize energy of a current solution
      int[] counter = new int[size + 1];
      int currentEnergy = calcError(currentSolution, counter);

      if (DEBUG_OUTPUT) {
         System.out.println("Current energy is: " + currentEnergy);
      }

      // Memorize the solution with smallest value of energy
      int[] bestSolution = currentSolution.clone();
      int bestEnergy = currentEnergy;

      double temperature = initialTemperature;

      while (temperature > minTemperature
            && currentEnergy > minEnergy) {

         for (int i = 0; i < numberOfTrials; i++) {
```

```java
            // Generate new solution:
            // swap two nearby items
            int pos1 = random.nextInt(sequenceLength);
            int pos2 = (pos1 + 1) % sequenceLength;
            swap(currentSolution, pos1, pos2);

            int newEnergy = calcError(currentSolution, counter);

            // According to the Boltzmann distribution
            double acceptanceProbability =
                  Math.exp(-(newEnergy - currentEnergy) / temperature);

            // Solutions with smaller energy - will be accepted always
            if (newEnergy < currentEnergy
                  || random.nextDouble() < acceptanceProbability) {

               currentEnergy = newEnergy;
               if (DEBUG_OUTPUT) {
                  System.out.println("Current energy is: " + currentEnergy);
               }

               if (newEnergy < bestEnergy) {
                  // Current solution is better than the best solution found so far
                  System.arraycopy(currentSolution, 0, bestSolution, 0, currentSolution.length);
                  bestEnergy = newEnergy;
               }
            } else {
               // If solution can't be accepted - rollback:
               // un-swap the items, which were swapped
               swap(currentSolution, pos1, pos2);
            }
         }
         // Decreasing temperature
         temperature *= temperatureDecreaseRatio;
      }
      // Return the best solution
      return bestSolution;
   }

   private static final int GAP_ITEM = -1;

   // Initial solution is: "1 1 2 2 3 3...."
   private static int[] generateInitialSolution(int size) {
      boolean withGap = (size - 3) % 4 != 0 && size % 4 != 0;
      int sequenceLength = 2 * size;
      if (withGap) {
         sequenceLength += 1;
      }
      int[] sequence = new int[sequenceLength];
      int pos = 0;
      for (int i = 0; i < size; i++) {
         int currChar = i + 1;
         sequence[pos] = currChar;
         sequence[pos + 1] = currChar;
```

```java
            pos += 2;
        }
        if (withGap) {
            sequence[sequence.length - 1] = GAP_ITEM;
        }
        return sequence;
    }

    private static final int NOT_INITIALIZED = -1;

    private static int calcError(int[] sequence, int[] counter) {
        Arrays.fill(counter, NOT_INITIALIZED);
        int error = 0;
        for (int i = 0; i < sequence.length; i++) {
            int item = sequence[i];
            if (item == GAP_ITEM) {
                continue;
            }
            if (counter[item] == NOT_INITIALIZED) {
                int expectedPosition = i + item + 1;
                counter[item] = expectedPosition;
            } else {
                int expectedPosition = counter[item];
                error += Math.abs(expectedPosition - i);
            }
        }
        return error;
    }

    private static int calcError(int[] sequence) {
        return calcError(sequence, new int[sequence.length / 2 + 1]);
    }

    private static void swap(int[] sequence, int i, int j) {
        int tmp = sequence[i];
        sequence[i] = sequence[j];
        sequence[j] = tmp;
    }

    private static String solutionToString(int[] solution) {
        char[] chars = new char[solution.length];
        for (int i = 0; i < solution.length; i++) {
            if (solution[i] == GAP_ITEM) {
                chars[i] = '-';
            } else {
                chars[i] = (char) ('A' + solution[i] - 1);
            }
        }
        return new String(chars);
    }
}
```