Evaluation of a Permutation-Based Evolutionary Framework for Lyndon Factorizations

🖂 Lily Major [1] https://orcid.org/0000-0002-5783-8432
Amanda Clare [1] https://orcid.org/0000-0001-8315-3659
Jacqueline W. Daykin [1,2] https://orcid.org/0000-0003-1123-8703
Benjamin Mora [3] https://orcid.org/0000-0002-2945-3519
Leonel Jose Peña Gamboa [1] https://orcid.org/0000-0002-7034-8438
Christine Zarges [1] https://orcid.org/0000-0002-2829-4296 *

[1]Department of Computer Science, Aberystwyth University, UK
{jam86,afc,jwd6,lep31,chz8}@aber.ac.uk
[2]Department of Information Science, Stellenbosch University, South Africa
[3]Computer Science Department, Swansea University, UK
b.mora@swansea.ac.uk

Abstract

String factorization is an important tool for partitioning data for parallel processing and other algorithmic techniques often found in the context of big data applications such as bioinformatics or compression. Duval's well-known algorithm uniquely factors a string over an ordered alphabet into Lyndon words, i.e., patterned strings which are strictly smaller than all of their cyclic rotations. While Duval's algorithm produces a pre-determined factorization, modern applications motivate the demand for factorizations with specific properties, e.g., those that minimize the number of factors or consist of factors with similar lengths. In this paper, we consider the problem of finding an alphabet ordering that yields a Lyndon factorization with such properties. We introduce a flexible evolutionary framework and evaluate it on biological sequence data. For the minimization case, we also propose a new problem-specific heuristic, Flexi-Duval, and a problem-specific mutation operator for Lyndon factorization. Our results show that our framework is competitive with Flexi-Duval for minimization and yields high quality and robust solutions for balancing where no problem-specific algorithm is available.

Keywords — Alphabet Ordering · Biosequences · Duval's Algorithm · Lyndon words · Permutations · String Factorization

1 Introduction

Many application areas, including text compression and bioinformatics, benefit from first breaking a string into factors with mathematically interesting properties. Such a factorization can suggest a mechanism to skip over irrelevant regions of a string, to partition computation or to construct local indices

[*] The first author is the main contributor. The ordering of the remaining authors is alphabetical with respect to the last name and does not imply any kind of weighting.

into the string. For example, Mantaci et al. [17] propose that a factorization into Lyndon words will permit the separate processing of local suffixes when building a suffix array, so that internal or external memory can be used, or to allow online processing. The Burrows-Wheeler Transform (BWT) [4] is at the heart of compression algorithms such as bzip2 [20], and bioinformatics algorithms such as Bowtie2 and BWA [15, 16]. The bijective BWT is based on Lyndon factorization [14] and recent work has shown [2] that when using this transformation, the number of search steps for a pattern is based on the number of distinct Lyndon factors in a particular suffix of the pattern. Lyndon words have also been used to discover tandem approximate repeats in biosequences [10] and in musicology [6]. A Lyndon word is a string which is minimal in the lexicographic order of its cyclic rotations. Duval's Algorithm (DA) [11] takes a string over an ordered alphabet, such as the Roman alphabet, and factors it into Lyndon factors in linear time and constant space. While the Lyndon factorization of a string for a given alphabet is unique [7], it may not yield the most suitable factors for a given application. Hence interest arises in seeking different factorizations via alphabet ordering techniques. For example, the string *parallelproblem* factors into the three Lyndon words $(p)(ar)(allelproblem)$ when using the Roman alphabet ordering. Using the alphabet ordering $m < b < o < e < l < r < a < p$ increases the number of Lyndon factors to $(p)(a)(ra)(l)(l)(elpr)(o)(ble)(m)$, and similarly, the ordering $p < r < o < l < a < b < e < m$ approximately balances the lengths as $(parallel)(problem)$. However, choosing an alphabet ordering can be computationally non-trivial. Regarding the BWT, the problem of deciding whether there exists an alphabet ordering satisfying a parameterized number of runs (maximal unary substrings) is NP-complete, while the corresponding minimization problem is APX-hard [3].

We wish to attempt two tasks: the production of the minimal number of Lyndon factors achievable over all possible alphabet orderings, and alternatively, the production of factors that are evenly balanced in size. We have developed an Evolutionary Algorithm (EA) to search for alphabet orderings for both tasks. EAs are *anytime* algorithms, i. e., they can be terminated at any time and will produce a solution, though the solution might be improved later if time permits. Finding improved permutations for a variety of tasks is an ideal use case for EAs [22].

To the best of our knowledge, alphabet ordering has not been considered other than for natural language text [5]. Furthermore, no consideration has been made in the domain for a heuristic setting, or in the context of Evolutionary Computation other than preliminary work in [8], where the impact of alphabet permutations produced significant variations in the numbers of factors obtained. Four fitness functions (maximize/minimize/parameterize/balance) for the number of factors demonstrated different factorizations suitable for various downstream tasks. This previous work mostly concentrated on random sequences; now we consider protein sequences and look at factorizations of proteins across many genomes. Here, we concentrate on comparing different mutation operators to develop the algorithm in a principled way. We also present a new problem-specific algorithm for the task of minimizing the number of factors.

2 Preliminaries

Let an alphabet Σ be a non-empty set of unique symbols also known as letters or characters. We denote Σ^+ as the set of all non-empty finite strings over the alphabet Σ and $\Sigma^* = \Sigma^+ \cup \{\boldsymbol{\varepsilon}\}$ where $\boldsymbol{\varepsilon}$ is the empty string with length of zero.[1]

For a string $\boldsymbol{w} = w_0 w_1 \dots w_{n-1}$ with each $w_i \in \Sigma$ and length n (or $|\boldsymbol{w}|$), we will represent $\boldsymbol{w}$ as an array $\boldsymbol{w}[0..n-1]$ with entries $\boldsymbol{w}[0]$, $\boldsymbol{w}[1]$, ..., $\boldsymbol{w}[n-1]$. A string is also known as a word.

Repetitions in strings will be denoted with a superscript as follows: for $\Sigma = \{a\}$ and $\boldsymbol{w} = aaa$ then $\boldsymbol{w} = a^3$. Lyndon words involve cyclic rotations (cyclic shifts or conjugates) of a string. For example, the cyclic rotations of the string *abc* are: *abc*, *bca*, *cab*. For a string $\boldsymbol{w} = \boldsymbol{w_p w_f w_s}$ of non-zero length, and strings $\boldsymbol{w_p}, \boldsymbol{w_f}, \boldsymbol{w_s} \in \Sigma^*$, then $\boldsymbol{w_p}$ is said to be a prefix of $\boldsymbol{w}$ while $\boldsymbol{w_s}$ is said to be a suffix of $\boldsymbol{w}$ and $\boldsymbol{w_f}$ is said to be a substring or factor.

For any non-empty string $\boldsymbol{w}$, $\boldsymbol{w} \in \Sigma^+$, there exists a unique factorization of $\boldsymbol{w}$ into Lyndon words, such that $\boldsymbol{w} = (\boldsymbol{u_1}) \geq (\boldsymbol{u_2}) \geq \cdots \geq (\boldsymbol{u_k})$ [7]. We denote these Lyndon factors as strings between

[1] We denote strings using boldface and characters in plainface.

brackets. For $\Sigma = \{a, b, c, d\}$, and a string $\boldsymbol{w} = dcba$, then the Lyndon factorization is written as $(d)(c)(b)(a)$.

We refer to the number of Lyndon factors produced from Duval's Algorithm for a given string $\boldsymbol{w} \in \Sigma^*$ and alphabet ordering π as $L(\Sigma, \pi, \boldsymbol{w})$.

3 Algorithms

In this section, we introduce our mutation-based evolutionary framework including the problem representation and different fitness functions. Afterwards, we derive a problem-specific mutation operator based on a modified version of Duval's Algorithm [11].

3.1 Representation and Fitness Function

We consider the problem of finding an alphabet ordering that induces a Lyndon factorization with a given property when used as input for Duval's Algorithm. [11]. Since an ordering of an alphabet Σ is equivalent to a permutation π of the letters in Σ, a permutation-based representation is the most natural choice for the problem at hand. Formally, given an alphabet Σ with size σ, the objective is to find a permutation π^* of Σ that optimizes a fitness function $f\colon S_\sigma \to \mathbb{R}^+$, where S_σ denotes the permutation space of the alphabet Σ.

As before, let $L(\Sigma, \pi, \boldsymbol{w})$ denote the number of Lyndon factors for a word $\boldsymbol{w} \in \Sigma^*$ and alphabet ordering π for Σ. Furthermore, let ℓ_j be the length of the j-th factor in a given factorization. In this paper, we consider three of the fitness functions introduced in [8], which deal with two aims for the Lyndon factorizations:

- Minimization of the number of factors, i. e., we minimize $f(\pi) = L(\Sigma, \pi, \boldsymbol{w})$.
- Balancing the length of the factors, i. e., all factors should be similar in length:
  1. difference between the maximum and the minimum length:
  $$f(\pi) = \max_{1 \leq j \leq L(\Sigma,\pi,\boldsymbol{w})} \ell_j - \min_{1 \leq j \leq L(\Sigma,\pi,\boldsymbol{w})} \ell_j \tag{1}$$
  2. the standard deviation of the factor length:
  $$f(\pi) = \sqrt{\frac{\sum_{j=1}^{L(\Sigma,\pi,\boldsymbol{w})} (\mu - \ell_j)^2}{L(\Sigma, \pi, \boldsymbol{w})}}, \tag{2}$$
  where μ is the mean length of the factors.

Note that for the balancing version we additionally require at least two factors. We do this by penalizing individuals with only one factor so that they are never accepted. All three fitness functions use Duval's [11] linear time and constant space algorithm to compute a unique Lyndon factorisation for a given string and alphabet ordering.

3.2 Mutation-based Evolutionary Framework

We consider a simple mutation-based evolutionary framework in the spirit of a (1+1) EA (see Alg. 1). The algorithm starts with a random permutation. It uses mutation to create an offspring and keeps the offspring if it is at least as good as its parent.

We compare three common mutation operators for permutation problems [12]:

- *Swap*: Selects two random letters and swaps them.
- *Insert*: Selects a random letter and moves it to a new random position.
- *Scramble*: Selects a random interval in the permutation and randomizes it.

Algorithm 1: (1+1) EA

```
1     Input: Word $\boldsymbol{w} \in \Sigma^*$; Fitness function $f$
2     Output: Best ordering of $\Sigma$ found for $f$
3 select $x \in S_\sigma$ uniformly at random;
4 while termination criterion not met do
5     $y \leftarrow$ mutate($x$);
6     if $f(y) \geq f(x)$ then
7         $x \leftarrow y$;
```

For minimization, we additionally use a fourth, problem-specific mutation operator, the *LF-inspired* operator, that is proposed in Sect. 3.4. For each operator, we consider three variants: one application of the operator, three applications in succession and a random number of applications in succession where the random number is determined by a Poisson distribution with parameter $\lambda = 1$. This gives a total of twelve mutation operators for minimization and nine for balancing. Note that the EA (Alg. 1) is formulated in a very general way and without defining specific stopping criteria. We provide details for the different cases we consider in Sect. 4.

We remark that we conducted preliminary experiments with Boltzmann selection (as used in Simulated Annealing [12, 13]) instead of elitist selection in lines 4–5 of Alg. 1. However, the results are not competitive and are therefore not included in this paper.

3.3 Flexi-Duval Algorithm

Here, we present a modification to Duval's Algorithm (DA) [11] for computing the Lyndon factorization (LF) of a string. For DA, the input string is over a totally ordered alphabet, such as the Roman or integer alphabet. When comparing a pair of letters during its linear scan, a non-fortuitous order such as $b > a$, or a repetition, will cause a factor(s) to be created. The new heuristic algorithm Flexi-Duval (FDA, Alg. 2) is formulated from DA, with the goal of minimizing the number of factors. In contrast to DA, FDA does not assume that the underlying alphabet is ordered but rather induces an ordering while scanning the string. In the above case of comparing b and a, if currently unassigned, the reverse order $b < a$ would be given thus causing the current factor to be extended. The result is a Lyndon factorization over a partially ordered alphabet. This ordering may produce fewer Lyndon factors (and therefore an increase in length for some factors) than the original DA. Note however, for a repetition such as a^k, $a \in \Sigma$, both DA and FDA will produce k factors.

Tab. 1 shows examples of factoring strings using both DA and FDA. We see that input strings exist for both algorithms such that neither is necessarily optimal for minimizing the number of Lyndon factors. This depends of course on the alphabet ordering, however, for some applications there is no inherent ordering of the letters, such as nucleotide or amino acids letters in molecular biology. This allows flexibility in specifying suitable factors for a task.

3.4 Problem-specific Mutation Operator

We introduce a problem-specific mutation operator (Alg. 3) which is based on Duval's Algorithm and the concept of inducing the alphabet ordering in Flexi-Duval (Alg. 2). By partially applying Duval's Algorithm to the input string using our alphabet ordering, the values of i and j can be found when a new Lyndon factor would be created. From this, we can find the characters in the alphabet ordering which cause the Lyndon factor to be created and then move the character at position j in the string in the alphabet ordering so that $\boldsymbol{w}[j] > \boldsymbol{w}[i]$. The effect of this is that, typically, a Lyndon factor is lengthened.

Algorithm 2: Algorithm FDA

```
    Input: A string $w$ of length $n$ on an unordered $\Sigma$
    Output: End positions of the Lyndon words
$h \leftarrow 0$;                              // start of the current Lyndon factor
while $h < n - 1$ do
    $i \leftarrow h$;                    // search for the start of the next Lyndon factor
    $j \leftarrow h + 1$;                  // search for the end of the next Lyndon factor
    while $j < n$ and ($isNotAssigned(w[j], w[i])$ or $w[j] \geq w[i]$) do
        if $w[i] \neq w[j]$ and $isNotAssigned(w[j], w[i])$ then
            assign($w[j] > w[i]$);
            $i \leftarrow h$;
        else
            if $w[j] > w[i]$ then
                $i \leftarrow h$;
            else
                $i \leftarrow i + 1$;
        $j \leftarrow j + 1$;
    repeat
        $h \leftarrow h + (j - i)$;
        output($h - 1$);
    until $h \geq i$;
```

Table 1: Example factorizations for DA and FDA.

String	Algorithm	Alphabet Ordering	Factorization
$bab^{O(n)}$	Duval's	$a < b$	$(b)(abb...b)$
$bab^{O(n)}$	Flexi-Duval	$b < a$	$(ba)(b)...(b)$
$dcba$	Duval's	$a < b < c < d$	$(d)(c)(b)(a)$
$dcba$	Flexi-Duval	$d > c > b > a$	$(dcba)$

4 Methodology

To provide a large dataset of strings, protein sequences from a collection of genomes of prokaryotic organisms were obtained from NCBI RefSeq [19]. We are interested in producing Lyndon factoring algorithms that either reduce the number of factors (EA & FDA - Sect. 5.2) or balance their length (EA only - Sect. 5.3). Our *LF-inspired* operator is specifically designed for minimization, and thus is only analyzed in this case. We select one genome (GCF_000064305.2_ASM6430v2) to find the most suitable mutation operator for both the minimization and balanced factor fitness functions (Sect. 5.1). Each fitness function (Sect. 3.1) is evaluated using each of the 2446 proteins in this genome for each of the mutation operators (Sects. 3.2 and 3.4). We use two methods of finding the best mutation operator for the EA. For minimization the best operator has a minimal average of the best fitness found over all iterations for all proteins. For balancing the best operators have minimal average standard deviation. We remark that the best operator does not change if the least difference between the average longest and average shortest Lyndon factor is used instead.

We select a set of 90 genomes to evaluate the performance of the selected mutation operator for the minimization fitness function with DA and FDA (Sect. 5.2) and balancing fitness functions (Sect. 5.3). As a baseline for minimization of the number of factors we consider two additional, very simple heuristics for each protein to produce an alphabet ordering: sorting unique characters by decreasing frequency for

Algorithm 3: LF-inspired Mutation Operator

```
    Input: A string $\boldsymbol{w}$ of length $n$, an alphabet $\Sigma$ and alphabet ordering $\pi$
    Output: A new alphabet ordering $\pi$
if $L(\Sigma, \pi, \boldsymbol{w}) > 1$ then
    i ← 0;
    j ← 1;
    while true do
        if $j = n$ or $\boldsymbol{w}[j] < \boldsymbol{w}[i]$ then
            break;
        else
            if $\boldsymbol{w}[j] > \boldsymbol{w}[i]$ then
                i ← 0;
            else
                i ← i + 1;
            j ← j + 1;
    $c = \boldsymbol{w}[j - i]$;
    remove $c$ from $\pi$;
    $p$ ← index of $\boldsymbol{w}[i]$ in $\pi$ or $|\pi|$ if missing;
    if $p = |\pi|$ then
        append $c$ to $\pi$;
    else
        $m \leftarrow |\pi| - 1 - p$;
        $r \leftarrow 0$;
        if $m > 0$ then
            $r \leftarrow$ uniformly at random from $\{0, 1, \ldots, m\}$;
        insert $c$ into $\pi$ at position $p + 1 + r$, shifting elements to the right;
output($\pi$);
```

(a) each single protein and (b) for entire genomes.

We stop at a predefined number of iterations or if the optimum is found (a value of 1 for the minimization of the number of Lyndon factors or 0 for the balancing factor length fitness functions). Our preliminary experiments[2] show the majority of the improvement in fitness for each fitness function to occur after a relatively low number of iterations. Improvement after this generally takes many more iterations for a slight improvement. We therefore used 3000 iterations for minimization of the number of Lyndon factors and 10000 iterations for balancing the length of factors (Fig. 1).

5 Results and Discussion

5.1 Single Genome

We present results for the EA (Alg. 1) using the proteins in GCF_000064305.2_ASM6430v2 as the input. For minimization of the number of Lyndon factors, we find that *Insertion* mutation applied three times in succession gives the lowest average of the best found number of Lyndon factors (Tab. 2). We observe that all settings involving standard mutation operators perform somewhat similarly with averages ranging from 1.709 (*Insertion (3)*) to 1.933 (*Scramble*). In general, *Insertion* performs slightly better than *Swap* which in turn performs slightly better than *Scramble*, no matter how often these

[2] Code is available at: https://github.com/jam86/Evaluation-of-a-Permutation-Based-Evolutionary-Framework-for-Lyndon-Factorizations

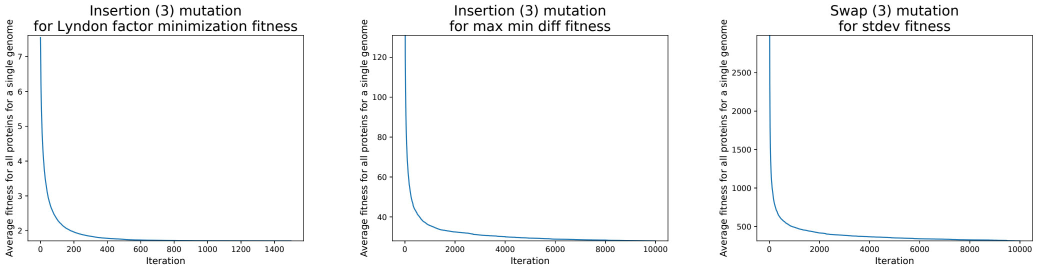

Figure 1: Average fitness over time for all proteins in GCF_000064305.2_ASM6430v2 for each fitness function of the EA for the best mutation operators for each fitness function.

Table 2: Distribution of the best number of factors per protein for a single genome for the minimization of the number of Lyndon factors. The selected operator is in bold. We show DA for comparison.

Mutation Operator	Max factors	Min factors	Mode factors	Mean ± stdev
Swap	8	1	1	1.877 ± 1.002
Swap (3)	8	1	1	1.727 ± 1.043
Scramble	8	1	1	1.933 ± 1.136
Scramble (3)	9	1	1	1.911 ± 1.155
Insertion	8	1	1	1.753 ± 1.042
Insertion (3)	**8**	**1**	**1**	$\mathbf{1.709 \pm 1.047}$
LF-inspired	14	1	5	5.601 ± 2.057
LF-inspired (3)	13	1	5	5.460 ± 2.020
Swap Poisson	8	1	1	1.796 ± 1.027
Scramble Poisson	8	1	1	1.913 ± 1.156
Insertion Poisson	8	1	1	1.739 ± 1.044
LF-inspired Poisson	13	1	5	5.596 ± 2.107
DA	16	2	7	7.284 ± 2.296

operators are applied. However, applying an operator three times performs slightly better than applying it only once or a random number of times. All standard operators achieve min and mode number of Lyndon factors of 1 and max number of Lyndon factors of 8 or 9.

All standard mutation operators yield averages smaller than 2. The best average achieved by *LF-inspired (3)* is 5.460, but this clearly improves over DA. Similar observations can be made for min, mode, and max number of Lyndon factors, no matter how often the operator is applied in succession. Due to the bias introduced by *LF-inspired* mutation, the EA quickly converges to a local optimum and is unable to escape. Combining the mutation with crossover as done previously for other (permutation-based) combinatorial optimization problems [21] is a promising direction for future research.

For the balance fitness functions we find again the mutation operators which give the lowest average standard deviation to be *Insertion* mutation applied three times in succession for Eq. 1 (Tab. 3). Further, we find the best mutation operator for Eq. 2 to be *Swap* mutation applied three times (Tab. 4). For both balancing fitness functions we make very similar observations in terms of the standard mutation operators and the problem-specific operator: All standard operators perform very similarly while our *LF-inspired* operator performs worse. However, we do not show *LF-inspired* for balancing as the operator is designed to minimize rather than balance. We remark that it performs poorly with both balancing fitness functions. Moreover, applying standard operators three times in succession outperforms the other considered variants. When looking into the different standard operators *Swap* and *Insertion* are slightly

Table 3: Balance factor length for Eq. 1 average maximum, minimum, and average standard deviation of Lyndon factor length per protein for a single genome. The selected operator is in bold.

Mutation Operator	Avg. Max	Avg. Min	Avg. Mean ± Avg. Stdev
Swap	112.956	79.516	96.033 ± 13.227
Swap (3)	123.076	95.123	109.110 ± 11.365
Scramble	119.588	85.857	102.228 ± 13.391
Scramble (3)	121.521	90.147	105.445 ± 12.592
Insertion	112.007	78.131	94.885 ± 13.271
Insertion (3)	**122.953**	**95.378**	**109.198 ± 11.182**
Swap Poisson	120.794	90.632	105.524 ± 12.199
Scramble Poisson	120.842	89.093	104.725 ± 12.702
Insertion Poisson	115.520	84.570	99.866 ± 12.229

Table 4: Balance factor length for Eq. 2 average maximum, minimum, and average standard deviation of Lyndon factor length per protein for a single genome. The selected operator is in bold.

Mutation Operator	Avg. Max	Avg. Min	Avg. Mean ± Avg. Stdev
Swap	96.988	47.251	63.456 ± 14.708
Swap (3)	**108.813**	**75.157**	**89.153 ± 11.112**
Scramble	106.498	63.335	79.080 ± 13.606
Scramble (3)	108.308	72.090	86.853 ± 11.889
Insertion	94.676	40.298	56.681 ± 15.567
Insertion (3)	107.425	71.074	84.978 ± 11.571
Swap Poisson	104.742	62.679	78.067 ± 13.160
Scramble Poisson	106.829	67.627	82.797 ± 12.611
Insertion Poisson	98.418	49.903	65.407 ± 14.254

better than *Scramble* for minimizing the difference while the picture is mixed when minimizing the standard deviation.

5.2 Minimizing the Number of Lyndon Factors

We find that FDA produces fewer Lyndon factors across our entire set of 90 test genomes than DA (Tab. 5). On average, the EA with the fitness function of the number of Lyndon factors almost performs as well as FDA. Given that FDA is a problem-specific heuristic developed for the purpose of minimizing the number of Lyndon factors, it is encouraging to see that our simple EA yields comparable performance in terms of solution quality. With respect to the runtime, FDA is far faster than the EA. For the same set of 90 genomes, FDA computed the factorization of all proteins in minutes compared to hours for the EA.

As the output of the FDA is a partial order, we convert each alphabet ordering result to a total order using depth first traversal, for inspection of the relevance of the order to biology. The positions of the characters within the orderings, averaged over each genome, are shown in Fig. 2. Proteins are not random, and usually begin with a methionine (M) [1]. This is reflected in its positions across the bottom of the graph. Some of the more common amino acids (L, K) are found next, and some of the less common amino acids (C, W) are towards the top of the graph, later in the ordering. We therefore compare the FDA results to the baseline of using a frequency-based ordering (Tab. 5).

FDA is significantly better than using an ordering based on the frequency of amino acids in each

Table 5: Number of Lyndon factors for methods that minimize the number of factors on the test set of 90 genomes.

Alphabet Ordering	Max Factors	Min Factors	Mode Factors	Mean ± Stdev
Duval's Algorithm	41	2	6	6.802 ± 2.169
Protein based frequency	31	1	7	7.227 ± 2.307
Genome based frequency	25	2	7	7.164 ± 2.238
EA with Insertion (3)	12	1	1	1.725 ± 1.102
Flexi-Duval Algorithm	6	1	1	1.197 ± 0.443

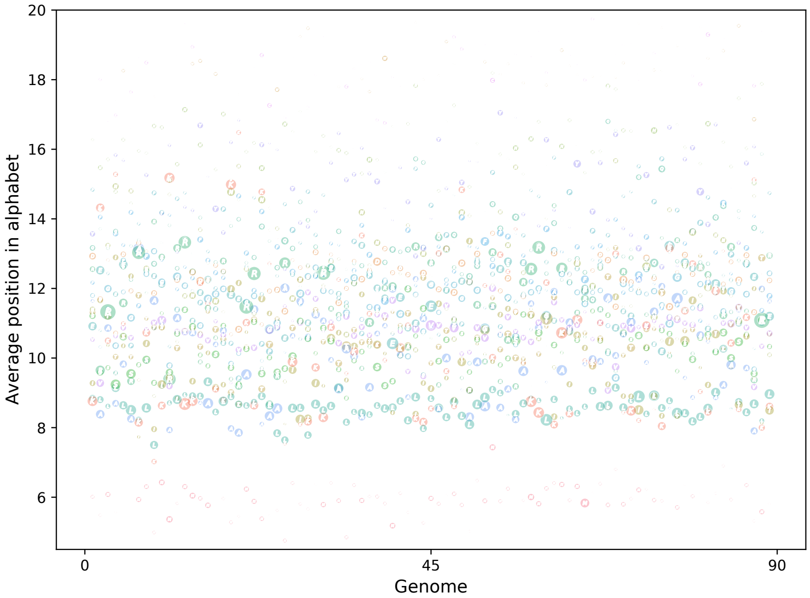

Figure 2: Average position of each character in the FDA alphabet ordering across all genomes in the test set of 90 genomes. The size of each point is proportional to the relative frequency of each amino acid in the entire genome. The y axis is truncated to begin at 4.5.

protein or genome.[3] Surprisingly, some proteins are factored into fewer Lyndon factors when using the whole genome frequency for alphabet ordering than when using the protein-specific frequency. Of the total 303,007 proteins in the test set of 90 genomes, 114,975 (37.94%) are factored into more Lyndon factors when using the protein frequency-based ordering than when using the genome frequency-based ordering.

5.3 Balancing the Length of Lyndon Factors

While there exists a problem-specific heuristic for the minimization problem, to the best of our knowledge no such method exists for balancing the lengths of the factors. Applying our EA for this use case is therefore an important step to demonstrate its usefulness. Based on our results in Sect. 5.1, we consider *Insertion (3)* for minimizing the difference of the factor lengths and *Swap (3)* for minimizing the standard deviation. We find that the fitness functions for balancing the length of Lyndon factors

[3] Using a Kolmogorov-Smirnov [18] (KS) two-sample test we obtained extremely low p-values calculated as zero.

Table 6: Average maximum, minimum, mean, and standard deviation of Lyndon factor lengths for the EA and DA per protein in 90 test genomes.

Alphabet Ordering	Avg. Max	Avg. Min	Avg. Mean ± Avg. Stdev
Duval's Algorithm	197.195	1.384	48.197 ± 70.204
Max Min Diff – Insertion (3)	116.816	91.474	104.110 ± 10.309
Stdev – Swap (3)	104.220	72.955	86.114 ± 10.486

are a great improvement over the regular output of DA (Tab. 6). The fitness function that uses the standard deviation of factor length produces smaller factors than the fitness function that minimizes the difference in factor length. However both produce equivalently stable variation in balancing the factor lengths (as seen by the average standard deviations).

Comparing results in Tab. 6 for the 90-genome set and Tabs. 3 and 4 for a single genome, we observe that the average standard deviation values on the 90-genome test set are very similar to the single genome (e.g., 10.309 (Tab. 6) vs 11.182 (Tab. 3)). This indicates that the genome we used to determine the best mutation operator is representative for the whole dataset. It is also important to remark, that the optimal solutions for all genomes are currently unknown, i. e., we do not know what the smallest possible difference/standard deviation is. However, achieving similar results across different sets of genomes indicates that our proposed method is successful in finding good solutions in a robust way.

6 Conclusion and Future Work

We have proposed a flexible mutation-based evolutionary framework for the problem of finding string factorizations with specific properties. We have considered two optimization goals (minimizing the number of factors and balancing the length of factors) and evaluated our framework on biological sequence data. We find that for the minimization problem our framework is competitive with a simple problem-specific heuristic in terms of solution quality while it yields high quality and robust solutions for the balancing variant where no such problem-specific algorithm is available. Moreover, we observe that a novel problem-specific mutation operator for minimization is not yet competitive with standard permutation-based operators from the literature.

Finding alphabet orderings with desired effects has been considered previously in [8] for random sequences as well as proteins from a single genome. Biological sequences have structure [9, 23] that is not necessarily present in totally random, fixed length sequences. This work more fully explores the factoring of proteins from a large number of genomes.

Future work will deal with the improvement and further evaluation of our framework, particularly the analysis of other fitness functions, problem-specific mutation operators and a comparison with population-based algorithms. Moreover, we plan to apply our framework to other types of strings such as a natural language corpus. While the EA produces a total alphabet ordering, FDA only determines a partial ordering, which is more flexible and may prove useful for finding consensus orderings in the context of sets of genomes. Future work will also investigate how such consensus orderings can be determined and how these could inform us about the biological structures of the collection of proteins.

Acknowledgments

This work is supported by the UKRI AIMLAC CDT, http://cdt-aimlac.org, grant no. EP/S023992/1, and was part-funded by the European Regional Development Fund through the Welsh Government, grant 80761-AU-137 (West).

UNDEB EWROPEAIDD
EUROPEAN UNION

Llywodraeth Cymru
Welsh Government

Cronfa Datblygu
Rhanbarthol Ewrop
European Regional
Development Fund

References

[1] Adams, J. M., and Capecchi, M. R. N-formylmethionyl-sRNA as the initiator of protein synthesis. *Proc. Natl. Acad. Sci. USA 55*, 1 (1966), 147–155. doi:10.1073/pnas.55.1.147.

[2] Bannai, H., Kärkkäinen, J., Köppl, D., and Piatkowski, M. Indexing the Bijective BWT. In *30th Annual Symposium on Combinatorial Pattern Matching (CPM 2019)* (2019), Leibniz International Proceedings in Informatics (LIPIcs), pp. 17:1–17:14.

[3] Bentley, J., Gibney, D., and Thankachan, S. V. On the complexity of BWT-runs minimization via alphabet reordering. arXiv:1911.03035v2, 2019.

[4] Burrows, M., and Wheeler, D. J. A block-sorting lossless data compression algorithm. Tech. rep., Digital Systems Research Center, Palo Alto, 1994.

[5] Chapin, B., and Tate, S. R. Higher compression from the Burrows-Wheeler Transform by modified sorting. In *Data Compression Conference, DCC 1998, Snowbird, Utah, USA, March 30 - April 1, 1998* (1998), IEEE Computer Society, p. 532. doi:10.1109/DCC.1998.672253.

[6] Chemillier, M. Periodic musical sequences and Lyndon words. *Soft Comput. 8*, 9 (2004), 611–616. doi:10.1007/s00500-004-0387-2.

[7] Chen, K.T., Fox, R.H., and Lyndon, R.C. Free differential calculus, IV – the quotient groups of the lower central series. *Ann. Math. 68*, 1 (1958), 81–95. doi:10.2307/1970044.

[8] Clare, A., Daykin, J. W., Mills, T., and Zarges, C. Evolutionary search techniques for the Lyndon factorization of biosequences. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2019, Prague, Czech Republic, July 13-17, 2019* (2019), ACM, pp. 1543–1550. doi:10.1145/3319619.3326872.

[9] De Lucrezia, D., Slanzi, D., Poli, I., Polticelli, F., and Minervini, G. Do natural proteins differ from random sequences polypeptides? Natural vs. random proteins classification using an evolutionary neural network. *PloS One 7*, 5 (2012), e36634–e36634. doi:10.1371/journal.pone.0036634.

[10] Delgrange, O., and Rivals, E. STAR: an algorithm to search for tandem approximate repeats. *Bioinformatics (Oxford, England) 20*, 16 (2004), 2812–2820. doi:10.1093/bioinformatics/bth335.

[11] Duval, J. P. Factorizing words over an ordered alphabet. *J. Algorithms. 4*, 4 (1983), 363–381. doi:10.1016/0196-6774(83)90017-2.

[12] Eiben, A. E., and Smith, J. E. *Introduction to Evolutionary Computing, Second Edition.* Natural Computing Series. Springer, 2015.

[13] Kirkpatrick, S., Jr., D. G., and Vecchi, M. P. Optimization by simulated annealing. *Science 220*, 4598 (1983), 671–680.

[14] Kufleitner, M. On bijective variants of the Burrows-Wheeler Transform. In *Proceedings of the Prague Stringology Conference 2009, Prague, Czech Republic* (2009), J. Holub and J. Zdárek, Eds., pp. 65–79.

[15] Langmead, B., and Salzberg, S. L. Fast gapped-read alignment with Bowtie 2. *Nat Methods 9* (2012), 357–359. doi:10.1038/nmeth.1923.

[16] Li, H., and Durbin, R. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics 25*, 14 (2009), 1754–1760. doi:10.1093/bioinformatics/btp324.

[17] Mantaci, S., Restivo, A., Rosone, G., and Sciortino, M. Suffix array and Lyndon factorization of a text. *J. Discrete Algorithms. 28* (2014), 2 – 8.

[18] Massey, F. J. The Kolmogorov-Smirnov test for goodness of fit. *J. Amer. Statist. Assoc. 46*, 253 (1951), 68–78. doi:10.2307/2280095.

[19] O'Leary, N. A., Wright, M. W., Brister, J. R., Ciufo, S., Haddad, D., McVeigh, R., Rajput, B., Robbertse, B., Smith-White, B., Ako-Adjei, D., Astashyn, A., Badretdin, A., Bao, Y., Blinkova, O., Brover, V., Chetvernin, V., Choi, J., Cox, E., Ermolaeva, O., Farrell, C. M., Goldfarb, T., Gupta, T., Haft, D., Hatcher, E., Hlavina, W., Joardar,

V. S., Kodali, V. K., Li, W., Maglott, D., Masterson, P., McGarvey, K. M., Murphy, M. R., O'Neill, K., Pujar, S., Rangwala, S. H., Rausch, D., Riddick, L. D., Schoch, C., Shkeda, A., Storz, S. S., Sun, H., Thibaud-Nissen, F., Tolstoy, I., Tully, R. E., Vatsan, A. R., Wallin, C., Webb, D., Wu, W., Landrum, M. J., Kimchi, A., Tatusova, T., DiCuccio, M., Kitts, P., Murphy, T. D., and Pruitt, K. D. Reference sequence (RefSeq) database at NCBI: current status, taxonomic expansion, and functional annotation. *Nucleic Acids Research 44*, D1 (2016), D733–D745. doi:10.1093/nar/gkv1189.

[20] Seward, J. bzip2 and libbzip2, 1996. http://sourceware.org/bzip2/.

[21] Tinós, R., Helsgaun, K., and Whitley, L. D. Efficient recombination in the Lin-Kernighan-Helsgaun traveling salesman heuristic. In *Parallel Problem Solving from Nature - PPSN XV - 15th International Conference, Coimbra, Portugal, September 8-12, 2018, Proceedings, Part I* (2018), A. Auger, C. M. Fonseca, N. Lourenço, P. Machado, L. Paquete, and L. D. Whitley, Eds., vol. 11101 of *LNCS*, Springer, pp. 95–107. doi:10.1007/978-3-319-99253-2_8.

[22] Whitley, D. Permutations. In *Handbook of Evolutionary Computation*, T. Bäck, D. B. Fogel, and Z. Michalewicz, Eds. IOP Publishing and Oxford University Press, 1997, p. C1.4.

[23] Yu, J.-F., Cao, Z., Yang, Y., Wang, C.-L., Su, Z.-D., Zhao, Y.-W., Wang, J.-H., and Zhou, Y. Natural protein sequences are more intrinsically disordered than random sequences. *Cellular and Molecular Life Sciences 73*, 15 (Aug 2016), 2949–2957. doi:10.1007/s00018-016-2138-9.