Indian Journal of Science and Technology, Vol 8(23), DOI: 10.17485/ijst/2015/v8i23/70645, September 2015
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

# Searching Gapped Palindromes in DNA Sequences using Dynamic Suffix Array

**Shivika Gupta[2*], Rajesh Prasad[1] and Sunita Yadav[2]**

[1]Department of Computer Science, Yobe State University, Damaturu, Nigeria; rajesh_ucer@yahoo.com
[2]Department of Computer Science and Engineering, Ajay Kumar Garg Engineering College, Ghaziabad - 201009, Uttar Pradesh, India; guptashivika05@yahoo.com, yadav.sunita104@gmail.com

## Abstract

**Background/Objectives**: In the biological sequences, palindromes can create structures that differ from the common structure of non-palindromic sequences. Unfortunately, mutations introduced by evolutionary methods, make it difficult to search the palindromes in DNA sequences. **Methods/Statistical Analysis**: The mutations occur in DNA sequences with spacer (i.e. set of characters). One version of such algorithms has been intended to search for palindromes with gaps (i.e. spacer) - gapped palindromes. The concept of Dynamic Suffix Array (DSA) is used to propose algorithms to search two classes of gapped palindromes-length constrained and long armed. DSA modifies the previous built suffix arrays when there is insertion and deletion of a new character, due to which efficiency is improved. DNA datasets obtained from National Centre for Biotechnology Information (NCBI) is taken as input. The execution time, palindrome weight and length of palindrome arms and spacer are analysed. **Findings**: Our proposed algorithms search maximal length constrained and long armed gapped palindromes in DNA sequences efficiently. Time complexity of our proposed algorithms is $O(n)$, where n is input parameters. Also, we compute palindrome weights in the DNA sequences. For length constrained gapped palindromes, our proposed algorithm is compared with existing one. The existing algorithm uses only suffix array. Experimental Observations reveal that by the use of DSA, execution time of our algorithm on different DNA sequences has been improved by maximum 57.89%. It also shows a decrease in the execution time over existing approach, proving our designed algorithm is space efficient, faster and easy to implement. **Applications/Improvement**: Our algorithms analyze short DNA sequences easily. These algorithms can be executed and tested on standard DNA and datasets with large number of base pairs.

**Keywords:** Dynamic Suffix Array, Gapped Palindromes, Length Constrained, Long Armed, Palindromes

## 1. Introduction

A string w is said to be palindrome, if it is equal to its reverse[1,3,4,6,7,11,13,16]. Recently, several researchers in the field of computer science have been interested towards palindromes[4,8,13]. Identifying palindromic structures is an important test case from an algorithmic viewpoint[2].

Computation related to palindrome pattern matching was considered in[16]. Palindrome pattern matching problem is to find all positions 'i', $1 \le i \le n$, in a reference sequence r[1…n] such that r[i … i + m – 1] and given pattern p[1 … m] are palindromic equivalent.

In molecular biology, analysis of DNA is one of the thrust areas of research. It is used to recognize the major functioning of living beings at the biological level. Any DNA sequence is string over four different base pairs: Adenine (A), Cytosine (C), Guanine (G) and Thymine (T)[7,8,10,17]. Palindromes in the DNA sequence can create structures that differ from the common structure of non-palindromic sequences as the nucleotides are uniformly spaced and lead to associate into complementary DNA strands. Palindrome structures in sequence show a major role in determining parameters of gene activities or further developments in cells[8,15].

Evolutionary processes introduce transformations that obscure the searching for palindrome structures in genome datasets. These transformations occur in structures with spacer (set of characters). Apart for searching

*Author for correspondence

exact palindromes in genome sequences, algorithms searches for gapped palindromes[12]. Hence, execution time of these types of algorithms is more, complicates their practical utility.

Two linear-time algorithms for computing different classes of gapped palindromes were proposed by Kolpakov and Kucherov[13]. The first algorithm is able to compute long-armed gapped palindromes using Lempel-Ziv factorization method. The other algorithm computes length-constrained gapped palindromes using suffix array. Those two algorithms gave extensive exemplification of all gapped palindromes existing in the word. Computation of palindrome weight (number and size of gapped palindromes) has also been a significant problem. Henstra[9] efficiently solved the problem of palindrome weight by using suffix array and longest common prefix array. Also, an algorithm for searching approximate palindromes in biological sequences has been designed in[10].

This paper presents algorithms for gapped palindromes in DNA sequences. Gapped palindrome is a string of the form $vuv^T$ for some v, u, and $v^T$ is reverse complement ('reversal' + 'complement of base pairs') of $v$[7,8,13]. In the string $vuv^T$: $v^T$ is called right arm, v is called left arm and u is spacer (gap). The first version is called *length-constrained gapped palindromes* in which the length of spacer is bounded by lower and upper length values, i.e. $MinGap \leq |u| \leq MaxGap$, and a length of palindrome arm by the lower bound, i.e. $MinArm \leq |v|$; where *MinGap, MaxGap, MinArm* are pre-defined constants. The second class is *long-armed gapped palindromes* that verify the condition $|u| \leq |v|$. In both the classes, palindromes discovered are required to be maximal.

Our main objective is to design efficient methods that can search maximal length constrained and long armed gapped palindromes in DNA sequences by using the concept of Dynamic Suffix Array (DSA)[14]. DSA is based on the dynamic construction of new suffix array[5] from the previously built suffix arrays. In this, the suffixes of the DNA sequence is stored by their index number not by long string of characters, therefore the constructed suffix array consumes less space. DSA modifies the previous built suffix arrays when there is insertion and deletion of a new character, due to which efficiency is improved.

Also, we compute palindrome weights in the DNA sequences. Both of our designed algorithms are implemented in asymptotic time $O(n)$, n denotes the length of an input sequence. For length constrained gapped palindromes, we compared the results of our proposed algorithm with KK algorithm explained in[13] as it uses only suffix array. Experimental results show a decrease in the execution time over existing approach, proving our designed algorithm is space efficient, faster and easy to implement. It shows that by the use of DSA, execution time of our algorithm on different DNA sequences has been improved by maximum 57.89%.

## 2. Palindromes

### 2.1 Length Constrained Gapped Palindromes by Kolpakov and Kucherov

Kolpakov and Kucherov[13] proposed an algorithm (we are saying KK Algorithm) for computing length-constrained palindromes in a word. This algorithm uses suffix array and longest common sub-word methods. It consists of two main steps that perform respectively a preparatory pre-processing and the main computation.

#### *2.1.1 Pre-Processing*

Considered an input v = v [1 … n]. For a position 'a', they considered palindromic arms in input word $a^+$ and $a^-$; where $a^-$, $a^+$ are denoted as starting positions in backward and forward direction respectively. For each position, define equivalence relation such that $l_1$, $l_2 \in$ equivalence relation, i.e. $l_1 \equiv l_2$ if and only if $V(l_1) = V(l_2)$. In this step, authors assigned the number to each position $a^-$, $a^+$ of its equivalence class under the above equivalence relation. This was done by using the suffix array for the input word $v\#v^T\$$ which takes time $O(n)$.

#### *2.1.2 Computation*

The goal of this step was to find all pair of positions satisfying $a < b$ such that

- $V(a^-) = V(b^+)$ (length constraint of palindrome arm)
- $MinGap \leq b - a \leq MaxGap$ (gap length constraint)
- $v[a] \neq v[b - 1]$ (maximal condition).

For such pair of positions, palindrome arm length has been computed by using the longest common sub-word. The gapped palindromes that are maximal and verifies length constraints can be found in time $O(n + S)$

retaining a set of equivalence relation, where n refers to the length of input word and S is referred as number of output length constrained palindromes. Kolpakov and Kucherov analyzed the algorithm theoretically; we have evaluated the algorithm using three standard DNA datasets (obtained from NCBI). The algorithm computes all maximal and length constrained palindromes in a given sequence.

### 2.2 Liner Time Palindrome Pattern Matching Algorithm

Tomohiro I et al.[16], proposed a linear time algorithm (KMP: Knuth Morris Pratt Type Algorithm) to solve the palindrome pattern matching problem. They found the maximal palindromic structure in the DNA of characters that matches structure of pattern. Given DNA sequence *S* of length 'c' and pattern *P* of length 'd', their objective was to find structure of pattern *P* in *S* by using KMP Type Algorithm. The KMP type algorithm is implemented by using Lpal array and failure function.

The Lpal array for the pattern and sequence is computed. The palindrome pattern matching is success if the Lpal array of pattern matches with Lpal in sequence. The Lpal array means the length of the lengthiest suffix palindrome for each position 'i'. For example, the Lpal array of the sequence A C A A G C G C is 1132113 for the above sequence.

The failure function is constructed for the Lpal array of pattern, so that it is considered when there is mismatch between the values of Lpal arrays of pattern and sequence. The failure function from state i to state j means that j is the maximum length of prefix and suffix of P [1…i] that palindrome matches.

Let the sequence T = A C A A G T G A G G C T C T. The Lpal array in the sequence is computed so that it matches with the Lpal array of pattern: At position '0', the longest suffix palindrome is 'A' itself, therefore the first element of Lpal array is '1' and then move right to the next state; in a similar manner compute the longest suffix palindrome at each position so that Lpal array is constructed. But at position '7', the element of Lpal array is 5, which cannot be transmitted, as it does not match with Lpal array of pattern P. Then, the failure function was considered according to which back to state '3'. Then, move three positions back and ignore the first four characters and then again recomputed Lpal array. The algorithm followed the same procedure until the Lpal array in sequence matches with the Lpal array of pattern at some position.

The palindrome structure of pattern is found in the sequence. This efficient approach takes linear time i.e. O(c+d) to perform the computation on arbitrary number of characters in a sequence.

### 2.3 Gapped Palindromic Weight in RNA

Henstra[9] proposed an algorithm to determine palindrome weight in RNA. It represents structures in a large RNA sequence which is to be identified and is defined by the number and size of gapped palindromes of the input sequence. Suffix arrays and longest common prefix arrays are both calculated for the input which enables quick retrieval of maximal gapped palindromes.

## 3. Proposed Algorithms

In this section, we describe two proposed algorithms (LCGP and LAGP) for computing all maximal gapped palindromes occurring in DNA sequence using dynamic suffix array. The first algorithm LCGP computes all gapped palindromes which are both maximal and length constrained. The other algorithm LAGP computes maximal long armed gapped palindromes.

### 3.1 Length Constrained Gapped Palindromes (LCGP)

This section presents algorithm for verifying length constrained maximal gapped palindrome. The outward/inward stretching of a palindrome can lead to a palindrome that may not be able verify the length constraints. For instance, if MinArm = 3, MinGap = 3 and MaxGap = 5, the palindrome CAA TTGA TTG is not maximal but verifies length constraints. On the other hand, its extension CAAT TG ATTG is maximal but is unable to verify length constraints.

For MinArm = 3, MinGap = 3 and MaxGap = 5, we now describe algorithm that computes all maximal length constrained gapped palindromes. Our proposed algorithm begins with the sliding window, which slides in forward direction across the input DNA string of length 'n' for the sequence of length 'k' and then, slides one position forward for new sequence, until the entire input is processed. For sequence of length 'k', our algorithm computes if it is length-constrained or not.

### 3.1.1 Computing Length-Constrained Gapped Palindromes

The steps for computing length-constrained are as follows:

*Step 1:* ***Reverse Complement***- First, we compute reverse complement of sequence. The reverse complement is defined as the complementary of base pairs and then its reversal.

*Step 2:* ***Concatenation with $***- In this step, the sequence and its reverse complement is concatenated with special symbol '$' (Here, $ indicates end of the string).

*Step 3:* ***Using Generalized Suffix Array***- This step involves the paradigm of generalized suffix array. We firstly build suffix arrays (permutation of index numbers giving the starting positions of suffixes of a string in alphabetical order) of sequence and its reverse complement. Then we compare each suffix of sequence with each and every suffix of the reverse complement for finding all maximal and non-maximal gapped palindromes.

*Step 4*: ***Compute Length Constraints***- Consider the gapped palindrome, in which number of matched characters (palindromic arms) should be at least to the length of *MinArm* and length of spacer should satisfy *MinGap* ≤ *spacer length* ≤ *MaxGap*. The palindrome arms should be complementarily equal on the both sides of the spacer.

*Step 5*: ***Use of Dynamic Suffix Array***- The suffix array is constructed for the sequence, but any edit operation in the sequence would lead to the construction of a brand new suffix array. Therefore, it is possible to modify previous suffix array when there is insertion and deletion of a character in the sequence.

The major steps of the algorithm are illustrated in Example 1.

**Example 1:** Consider the sequence S = 'GTTAACAAAC' of length '10'. To compute if sequence S is length-constrained, the following steps are:

- The reverse complement of sequence S is GTTTGTTAAC.
- The sequence S and its reverse complement is concatenated with $ to get GTTAACAAAC$ and GTTTGTTAAC$ respectively.
- The suffix arrays are constructed for sequence S and its reverse complement. Each suffix of sequence S is compared with each and every suffix of the reverse complement for finding maximal and non-maximal gapped palindromes. For example, the suffix AAAC$ is compared to the each and every suffix of reverse complement. First the suffix AAAC$ is compared to AAC$, the match found is AA*. Then the suffix AAAC$ is compared to AC$. The match found is A*; where * is unmatched character. This method will work until the suffix AAAC$ is compared to all the suffixes of reverse complement. Only those subsequence is considered as gapped palindromes when it contains matched and unmatched characters, for example, the match found like '***' is not gapped palindrome as it does not contain any matched character and the match found like 'AAC' is also not gapped palindrome as it does not contain any unmatched character. Here, for suffix AAAC$, the possible gapped palindromes found are AA*, A*, *AA* and **A*.

Similarly, the suffix AAC$ is compared to the each and every suffix of reverse complement. Here, for suffix AAC$, possible gapped palindromes found are A*, *A*. This procedure will follow until we find all maximal and non-maximal gapped palindromes. Table 1 shows all maximal and non-maximal gapped palindromes found in sequence S.

- Our goal of this step is to determine palindromic arms and spacer. Consider the gapped palindrome GTT****AAC (Table 1.) with palindromic arm (matched characters) of length 3 which is equal to value of MinArm. The length of spacer (unmatched characters) is 4 satisfying *MinGap (=3)* ≤ *spacer length (=4)* ≤ *MaxGap (=5)*. The palindrome arms are complementarily equal on the both sides of spacer (i.e. left arm and reverse of right arm are complementarily equal). Hence, the sequence S = 'GTTAACAAAC' is length constrained gapped palindrome.
- Continuing with sequence S = GTTAACAAAC$, suppose there is insertion of the character A at the 11$^{th}$ position and deletion of G at the 1$^{st}$ position so that the new sequence is TTAACAAACA$. The new suffix array of new sequence is constructed from the previous suffix array of S. As the character A is newly inserted character in S, therefore find out the position of A$ in previous suffix array of S and then insert in it. The one of the suffixes in S i.e. GTTAACAAAC$ is deleted from previous suffix array. The character A is appended at the end of remaining suffixes other than $.

**Table 1.** Maximal and Non-maximal gapped palindromes found in GTTAACAAAC$

| Suffixes of GTTAACAAAC$ | Gapped Palindromes |
|---|---|
| $ | – |
| AAAC$ | AA* |
| | A* |
| | *AA* |
| | **A* |
| AAC$ | A* |
| | *A* |
| AACAAAC$ | A* |
| | ***AA* |
| | *A** |
| | ****AAC |
| | ***A* |
| | *****A* |
| AC$ | A* |
| ACAAAC$ | A** |
| | ***AAC |
| | **A* |
| | ****A* |
| | **AA* |
| C$ | - |
| CAAAC$ | *A* |
| | ***A* |
| | *AA* |
| | **AAC |
| GTTAACAAAC$ | GTT****AAC |
| | **T*A* |
| | *T*A* |
| | *T****A* |
| | **TT***AA* |
| TAACAAAC$ | *A* |
| | ****A* |
| | T***AA* |
| | T*A** |
| | T****AAC |
| | T*****A* |
| TTAACAAAC$ | *T*A** |
| | *T*****A* |
| | T*A* |
| | T****A* |
| | TT***AA* |
| | TT****AAC |

### *3.1.2 Asymptotic Analysis*

The algorithm for finding length constrained gapped palindromes can be implemented in time $O(n)$. Step 1 is takes time $O(n)$. The step 2 is done in constant time. The step 3 requires $O(n)$ time by generalized suffix array. At step 4, finding palindromic arm's length and computing spacer length requires $O(n)$ time. At step 5, building of dynamic suffix array takes time $O(n)$. Hence, total execution time of LCGP is $O(n)$.

## 3.2 Long-Armed Gapped Palindromes (LAGP)

LAGP algorithm computes long armed maximal gapped palindromes occurring in input DNA sequences. LAGP begins with the sliding window, which slides in forward direction across the input DNA string of length 'n' for the sequence of length 'k' and then, slides one position forward for new sequence, until the entire input is processed. For sequence of length 'k', our algorithm computes whether it is long-armed or not.

### *3.2.1 Computing Long-Armed Palindromes*

The steps for computing long-armed are as follows:

*Step 1:* Repeat steps 1–3 that are also used for searching length constrained gapped palindromes.

*Step 2:* ***Compute Length Constraints***- Our goal of this step is to determine length constraints of palindromic arms and spacer. Consider maximal gapped palindrome, in which there are maximum number of matched characters and minimum number of unmatched characters. The palindromic arms (matched characters) should be complementarily equal on both sides. The length of palindromic arms should be greater than or equal to the length of spacer (unmatched characters).

*Step 3:* ***Use of Dynamic Suffix Array***- The suffix array is constructed for the sequence as shown in step 3, but any edit operation in the sequence would lead to the construction of a brand new suffix array. Therefore, it is possible to modify previous suffix array when there is insertion and deletion of a character in the sequence.

The major steps of the algorithm are illustrated in Example 2.

**Example 2:** Consider the sequence S = 'GTTAACTAAC' of length '10'. To compute if sequence S is long-armed, the following steps are:

- The reverse complement of sequence S is GTTAGTTAAC.
- The sequence S and its reverse complement is concatenated with $ to get GTTAACTAAC$ and GTTAGTTAAC$ respectively.
- The suffix arrays of GTTAACTAAC$ and GTTAGTTAAC$ are shown in Table 2. Each suffix of sequence S is compared with each and every suffix of the reverse complement for finding maximal and non-maximal gapped palindromes as shown in Table 3.
- Consider the maximal gapped palindrome GTTA**TAAC found in Table 3, in which there are maximum number of matched characters and minimum number of unmatched characters. Here, the palindromic arm (matched characters) is of length 4 and spacer (unmatched characters) is of length 2. The palindromic arms are complementarily equal on the both sides (i.e. left arm and reverse of right arm are complementarily equal). Also, the length of spacer is less than the length of palindromic arms.
- Hence, we say that the S = 'GTTAACTAAC' is long armed gapped palindrome.
- Continuing with sequence S = GTTAACTAAC$, suppose there is insertion of the character A at the 11$^{th}$ position and deletion of G at the 1$^{st}$ position so that the new subsequence is TTAACTAACA$. The new suffix

**Table 2.** Suffix Arrays of GTTAACTAAC$ and GTTAGTTAAC$

| Index for Subsequence | Suffix of Subsequence | Index for Reverse Complement | Suffix of Reverse Complement |
|---|---|---|---|
| 11 | $ | 11 | $ |
| 8 | AAC$ | 8 | AAC$ |
| 4 | AACTAAC$ | 9 | AC$ |
| 9 | AC$ | 4 | AGTTAAC$ |
| 5 | ACTAAC$ | 10 | C$ |
| 10 | C$ | 5 | GTTAAC$ |
| 6 | CTAAC$ | 1 | GTTAGTTAAC$ |
| 1 | GTTAACTAAC$ | 7 | TAAC$ |
| 7 | TAAC$ | 3 | TAGTTAAC$ |
| 3 | TAACTAAC$ | 6 | TTAAC$ |
| 2 | TTAACTAAC$ | 2 | TTAGTTAAC$ |

**Table 3.** Maximal and Non-Maximal Gapped palindromes found in GTTAACTAAC$

| Suffixes of GTTAACAAAC$ | Gapped Palindromes |
|---|---|
| $ | – |
| AAC$ | A* |
| | A** |
| | *AA |
| | *A* |
| ACTAAC$ | A* |
| | A**TAAC |
| | ****A* |
| | *A** |
| AC$ | A* |
| | A* |
| ACTAAC$ | A** |
| | A* |
| | A*T*A |
| | **TAAC |
| | **TA** |
| | ***A* |
| C$ | - |
| CTAAC$ | *T*A* |
| | *T*A* |
| | **A* |
| | *TAAC |
| | *TA** |
| GTTAACTAAC$ | **T*A** |
| | GTTA**TAAC |
| | *T*A* |
| | *T*A***A* |
| TAAC$ | *A* |
| | TA** |
| | T*A* |
| | T*A* |
| TAACTAAC$ | *A* |
| | ****A* |
| | TA**TAAC |
| | T*A** |
| | T*A*T*A* |
| TTAACTAAC$ | *T*A** |
| | *T*A*T*A* |
| | T*A* |
| | T****A* |
| | TTA**TAAC |

array of new sequence is constructed from the previous suffix array of S. As the character A is newly inserted character in S, therefore find out the position of A$ in previous suffix array of S and then insert in it. The one of the suffixes in S i.e. GTTAACTAAC$ is deleted from previous suffix array. The character A is appended at the end of remaining suffixes other than $.

### 3.2.2 *Asymptotic Analysis*

All long armed gapped palindromes is to be found in $O(n)$ time. Step 1 is implemented in time $O(n)$. At step 2, finding palindromic arm's length and computing spacer length requires $O(n)$ time. At step 3, dynamic suffix array construction requires $O(n)$ time. Hence, total execution time of LAGP is $O(n)$.

## 4. Main Results

For experiments, we use three DNA datasets obtained from National Centre for Biotechnology Information (NCBI):

- **Dataset 1:** Chain B, Structure of the Rep Associates Tyrosine Transposase Bound to A Rep Hairpin.
- **Dataset 2:** Bovine testis satellite I (1.715 g/ml) DNA, segment 3.
- **Dataset 3:** Human lymphocyte antigen (HLA-DN2) mRNA, 3' flank.

In dataset 1, Rep associates represents class of extra genic sequences that form nucleotide stem-loop structures. In dataset 2, Cytosine methylation (mC) is tissue specific. mC location is non-random and clustered. All mCs are located in 'CG' doublets, better than half of which are in short palindromes. The palindrome 'CCGG' is almost completely methylated in various tissues, 'TGCA' differently in the various tissues.

We implemented the algorithms (LCGP, LAGP and KK) in C, compiled with Turbo C++ compiler, with Windows 8, 2.13 GHZ processor with 4 GB RAM.

### 4.1 Length Constrained Gapped Palindromes (LCGP)

Execution time of the algorithm is measured in milliseconds (ms). For length constrained palindromes, the execution time varies with the base pairs in DNA datasets. As the base pairs increases, the execution time of the algorithm increases. Results are shown in Table 4. For analyzing *dataset 1* of length *32* base pairs (bp), the algorithm takes 0.439560 ms. for *dataset 2* of *131* base pairs (bp), the execution time increases to 0.318681 ms. similarly, for *dataset 3* of *230* base pairs (bp), the execution time increases to 1.520579 ms.

The Table 4 also gives a ***palindrome weight*** computed by LCGP algorithm. Palindrome weight is represented by number and size of gapped palindromes. Dataset 1 does not contain any length constrained gapped palindrome whereas dataset 2 and dataset 3 contain four and three length constrained palindromes of length '10' respectively. The Table 4 also shows actual length constrained gapped palindromes found by LCGP in DNA datasets.

**Table 4.** Comparison of existing Algorithm with proposed algorithm

| Dataset | Total base pairs in dataset | Palindrome Weight | | Palindrome Arm Length | Gap Length | Length Constrained Gapped Palindrome | Time (in ms) by KK algorithm[13] | Time (in ms) by LCGP algorithm |
|---|---|---|---|---|---|---|---|---|
| | | Number of palindrome | Size of palindrome | | | | | |
| Dataset 1 | 32 bp | 0 | – | – | – | – | 1.043956 | 0.439560 |
| Dataset 2 | 131 bp | 4 | 10<br>10<br>10<br>10 | 3<br>3<br>3<br>3 | 4<br>4<br>4<br>4 | GGACTTCTCC<br>ATCTTGCGAT<br>GGGACTCCCC<br>ACTCCCCAGT | 1.373626 | 1.318681 |
| Dataset 3 | 230 bp | 3 | 10<br>10<br>10 | 3<br>3<br>3 | 4<br>4<br>4 | TCTACAAAGA<br>CCAGGCATGG<br>GGCTGGGGCC | 1.538462 | 1.520579 |

## 4.2 Comparison with other Algorithm

We compared the execution time of our proposed LCGP algorithm with KK algorithm[13]. Results are shown in Table 4. It has been observed that, the execution time of LCGP is less than the execution time of KK algorithm. For DNA dataset 1, the execution time of our LCGP algorithm improves by 57.89%. For another dataset 2, execution time improves by 3.99%. In another dataset 3, the execution time is improved by 1.16%. Figure 1 shows fair comparison between the existing (KK Algorithm) and proposed algorithm. It provides comparison of execution time against number of base pairs. The results show that LCGP algorithm is efficient as compared to KK algorithm.

## 4.3 Long Armed Gapped Palindromes (LAGP)

Execution time of the algorithm is measured in milliseconds (ms). For long armed palindromes, the execution time varies with the base pairs number in DNA datasets. As the base pairs number increases, the execution time of the algorithm increases. Results are shown in Table 5. For analyzing *dataset 1* of length *32* base pairs (bp), the algorithm takes 0.824176 ms. For *dataset 2* of *131* base pairs (bp), the execution time increases to 0.989011 ms. similarly, for *dataset 3* of *230* base pairs (bp), the execution time increases to 1.483516 ms.

The Table 5 also gives a ***palindrome weight*** computed by LAGP algorithm. Each dataset contain one long armed gapped palindrome of length '10'. The Table 5 also shows actual long armed gapped palindromes found by LAGP in DNA datasets.

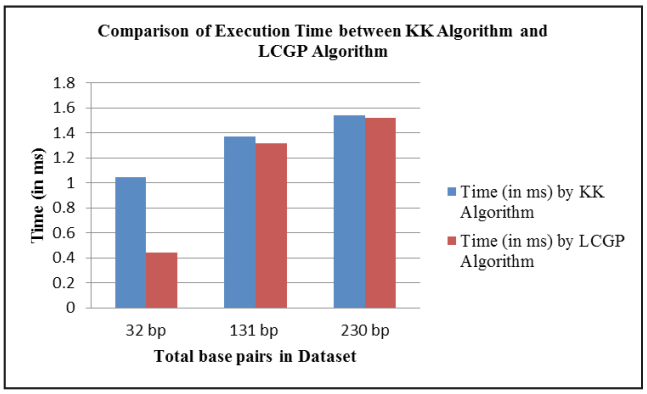
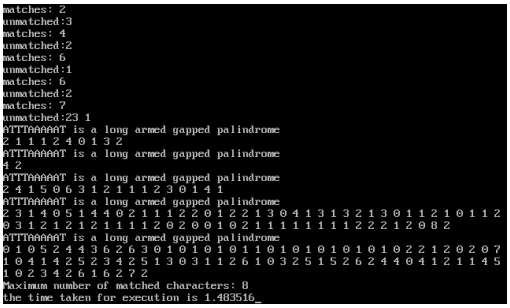



**Figure 1.** Comparison of execution time between existing (KK Algorithm) and proposed algorithm.


**(a)**




**(b)**

**Figure 2.** Snapshots of the output. **(a)** Searching length constrained Gapped Palindromes in Dataset 2 and **(b)** Searching long armed Gapped Palindromes in Dataset 3.

**Table 5.** Long Armed Gapped Palindromes detected in standard Datasets

| Dataset | Total base pairs in dataset | Palindrome Weight | | Length of Palindrome arm | Spacer Length | Long Armed Gapped Palindrome | Time (ms) by LAGP algorithm |
|---|---|---|---|---|---|---|---|
| | | Number of Palindrome | Size of Palindrome | | | | |
| Dataset 1 | 32 bp | 1 | 10 | 4 | 2 | GCGTTTACGC | 0.824176 |
| Dataset 2 | 131 bp | 1 | 10 | 4 | 2 | CGCTGTAGCG | 0.989011 |
| Dataset 3 | 230 bp | 1 | 10 | 4 | 2 | ATTTAAAAAT | 1.483516 |

## 5. Conclusions

This paper presents novel algorithms for searching the gapped palindrome. The proposed algorithms are based on dynamic suffix array and allow constructing the new suffix array from the already built suffix array. The presented algorithms ensure finding long armed and length constrained versions of gapped palindromes in the biological DNA sequence, verifying all the conditions. Our algorithms analyzed s*hort* DNA sequences easily.

The proposed algorithms utilize the advantages of suffix array and dynamic suffix array. They achieve time complexity *O*(n), consume less space and are easy to implement.

Comparing our algorithm (for length constrained) to other algorithm using suffix array, the algorithm shows improvement in execution time. Experimental results showed that execution time of our algorithm improves by 57.89%.

These algorithms can be executed and tested on standard DNA and datasets with large number of base pairs. The same can be extended for searching different classes of gapped palindromes in protein sequences. In addition, the work can be extended for biological sequences having palindromes with insertions, substitutions and deletions.

## 6. References


1. Bannai H, Masayuki T. Computing palindromic factorizations and palindromic covers on-line. Proceedings of 25th Annual Symposium on Combinatorial Pattern Matching (CPM); Moscow, Russia: Springer; 2014. p. 150–61.
2. Bille P, Gortz IL, Sach B, Vildhoj HW. Time-space tradeoffs for longest common extensions. Journal of Discrete Algorithms. 2014; (25):42–50.
3. Chowdhury SR, Hasan MM, Iqbal S, Rahman MS. Computing a longest common palindromic subsequence. Fundamental Informaticae. 2014; 129(4):329–40.
4. Glen A, Justin J, Widmer S, Zamboni LQ. Palindromic richness. European Journal of Combinatorics. 2009; 30(2):510–31.
5. Goel A, Prasad R. Efficient Indexing Techniques for Record Matching and Deduplication. International Journal of Computer Vision and Robotics (Inderscience). 2014; 4(1/2):75–85.
6. Groult R, Prieur E, Richomme G. Counting distinct palindromes in a word in linear time. Information Processing Letters. 2010; 110(20):908–12.
7. Gupta R, Mittal A, Gupta S. An efficient algorithm to detect palindromes in DNA sequences using periodicity transform. Signal Processing. 2006; 86:2067–73.
8. Gusfield D. Algorithms on Strings, Trees, and Sequences. Cambridge University Press, New York; 1997.
9. Henstra SJ. Determining gapped palindrome density in RNA using suffix arrays. Leiden Institute of Advanced Computer Science. Leiden University; 2010. p. 1–15.
10. Hsu PH, Chen KY, Chao KM. Finding all approximate gapped palindromes. Proceedings of ISAAC, LNCS. 2009; 5878:1084–93.
11. Jeuring J. Finding Palindromes: Variants and Algorithms, the Beauty of Functional Code, Lecture Notes in Computer Science 8106; 2013. p. 258–72.
12. Kolpakov R, Kucherov G. Finding repeats with fixed gap. Proceedings of the 7th International Symposium on String Processing and Information Retrieval (SPIRE), A Coruña, IEEE; 2000. p. 162–68.
13. Kolpakov R, Kucherov G. Searching for gapped palindromes. Theoretical Computer Science. 2009; 410(51):5365–73.
14. Salson M, Lecroq T, Leonard M, Mouchard L. Dynamic extended suffix arrays. Journal of Discrete Algorithms. 2010; 8:241–57.
15. Smith GR. Meeting Palindromes head- to- head. Genes and Development. 2008; 22:612–20.
16. Tomohiro I, Inenaga S, Takeda M. Palindrome Pattern Matching. Theoretical Computer Science; 2012. p. 1–9.
17. Sreejith K, Sebastian CD. Molecular Phylogeny and Genetic Analysis of Green Leafhopper-Nephotettix Virescens (Distant) Using Mitochondrial COI Gene. Indian Journal of Science and Technology. 2015; 8.1:61–4.