Primate Speciation Links to Massive and Directional Shrinkage of the Dinucleotide Short Tandem Repeat Compartment.

Arabfard M[1], Salesi M[1], Arabipour I[2], Khorram Khorshid H.R.[3], Najafipour R[4], Delbari A[5], Ohadi M[5*]

1- Chemical Injuries Research Center, Systems Biology and Poisonings Institute, Baqiyatallah University of Medical Sciences, Tehran, Iran.
2- Department of Biotechnology, Science and Research Branch, Islamic Azad University, Tehran, Iran.
3- Personalized Medicine and Genometabolomics Research Center, Hope Generation Foundation, Tehran, Iran.
4- Cellular and Molecular Research Center, Research Institute for Prevention of Non-Communicable Disease, Qazhin University of Medical Sciences Qazvin. Iran.
5- Iranian research center on aging, University of social welfare and rehabilitation sciences, Tehran, Iran.

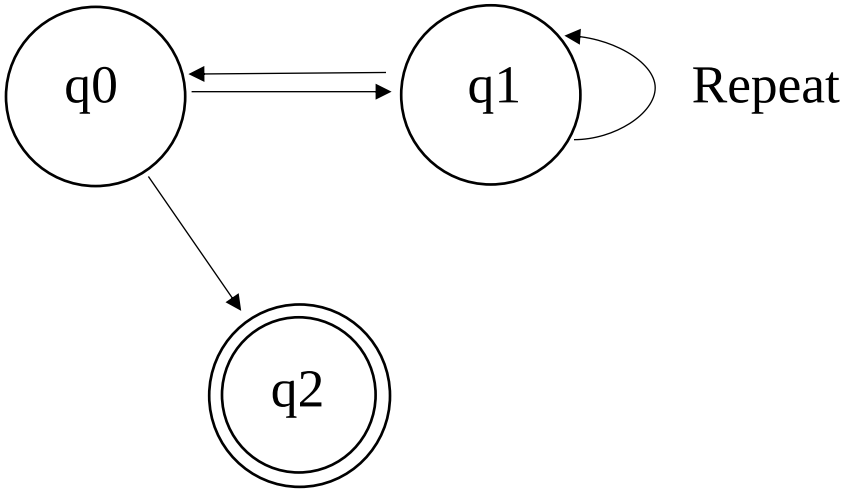
*__Correspondence to__: ohadi.mina@yahoo.com
mi.ohadi@uswr.ac.ir

Keywords: dinucleotide; short tandem repeat; directional; primate; mouse; speciation

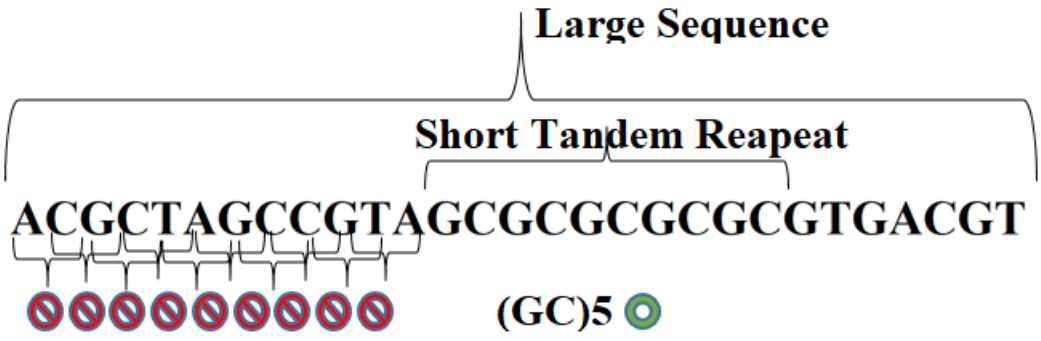
Dinucleotide Short Tandem Repeat Finder (Di-Finder).

In order to find short tandem repeat sequences in a large sequence, a repeating loop is designed equal to the length of the large sequence Inside this loop. All consecutive di-nucleotides are examined and their list is extracted. This process is checked by a regular expression algorithm.

A regular expression is a sequence of characters that defines a search pattern. These patterns are usually used by sequence search algorithms to "find" operations on sequences. This is a technique used in computer science, especially in formal language theory. The following state machine automata represent all algorithm states.



This process is repeated for the length of the sequence for each nucleotide pair. The implemented algorithm does not support mismatches (only perfect/pure STRs are analyzed). Following is an overview of the algorithm and Pseudo-code used:



Pseudo Code for Di-Finder:

```
int len_seq = sequence.Length;
for (int i = 0; i < len_seq ; i++)
{
    for (int j = min_length; j < max_length + 1; j++)
    {
        if ((i + j) > len_seq) { break; }
        string sub_seq = sequence.Substring(i, j);
        double len_sub_seq = sub_seq.Length;
        string sub_seq_pattern = sub_seq;

        int matches = 1;
        while (Regex.IsMatch(sub_seq_pattern, sequence.Substring(i + j *matches, j)))
        {
            matches++;
        }

        if (matches >= min_repeats && (j * matches) >= min_length_of_MR)
        {
            microsatellite.Add(sequence.Substring(i, j * matches).Substring(0,j));
            i += j * matches;
        }
    }
}
```