

The Low-Autocorrelation Problem of Binary Strings in MATLAB

Ofer M. Shir

Introduction

Given a binary string of dimension n , $\vec{y} \in \{-1, +1\}^n$, the merit factor is defined as:

$$f(\vec{y}) \equiv \frac{n^2}{2 \cdot E(\vec{y})} \quad (1)$$

where

$$E(\vec{y}) \equiv \sum_{k=1}^{n-1} \left(\sum_{i=1}^{n-k} y_i \cdot y_{i+k} \right)^2 \quad (2)$$

The low autocorrelated binary string problem is defined as finding the string with the maximal merit factor.

Implementation in MATLAB

General

We propose an implementation of the GA basic features using Matlab 7.0 operations. The provided *pseudocode* is written in a compact matrix notation, aiming to exploit MATLAB's well-known optimizations for matrices operations.

Special Features

The fitness calculation is extracted in a pure matrix notation.

- *Calculating the objective function.* Given a population of strings, $\{\vec{y}_i\}_{i=1}^{pop}$, represented as column vectors, let us construct the population matrix of dimension $(n \times pop)$, denoted as:

$$\overleftrightarrow{\Omega} \equiv [\vec{y}_1, \vec{y}_2, \dots, \vec{y}_{pop}] \quad (3)$$

For computing the inner sum of the merit factor **for the entire population**, as a function of the running index k , denoted as $\vec{\sigma}_k$ - let us construct 2 auxiliary matrices taken from the original population matrix as follows:

$$\overleftarrow{\psi} \equiv \overleftrightarrow{\Omega}(1..n-k, :) \quad \overleftarrow{\xi} \equiv \overleftrightarrow{\Omega}((k+1)..n, :) \quad (4)$$

Defining the DOT-SQUARE MATRIX-OPERATION as taking the square of each matrix element, we can write as a result:

$$\vec{\sigma}_k = [\text{diagonal}(\overleftarrow{\xi} \cdot \overleftarrow{\psi})].^2 \quad (5)$$

In this manner we have managed to get the inner sum of the entire population in few basic matrix operations.

The final merit factor could be calculated now easily with a simple summation over k .

- *Applying the proportional Selection Operator.* Given the objective function values of the current generation of the population, represented in a vector $\vec{\mathcal{F}}$ (this vector could have been scaled earlier by a constant c), apply the cumulative summation operation:

$$\vec{\chi} = \text{cumsum}(\vec{\mathcal{F}}) \equiv \left(\mathcal{F}_1, \mathcal{F}_1 + \mathcal{F}_2, \dots, \sum_{i=1}^n \mathcal{F}_i \right) \quad (6)$$

Given \vec{R} , a randomly initialized vector of length n with elements uniformly distributed on $[0, 1]$, define the next generation probability vector as follows:

$$\vec{\mathcal{P}} \equiv \chi_n \cdot \vec{R} \quad (7)$$

Now, iterating through this vector, the next generation will be selected easily:

$$\forall i \quad y_i^{\text{new}} = y_j^{\text{old}} \quad \text{s.t.} \quad j \equiv \min_{k=1}^n \{k \mid \mathcal{P}_k < \chi_k\} \quad (8)$$

The task of choosing the n strings of the next generation has been reduced to a short sequence of small and basic matrix operations.

- *Applying the Mutation Operator.* Given the old population matrix $\overleftrightarrow{\Omega}_{old}$, and the current mutation probability p , generate a mutation matrix $\overleftrightarrow{\mathcal{M}}$ of dimension $(n \times pop)$ with the following elements (single Matlab command):

$$\mathcal{M}_{ij} = \begin{cases} -1 & \text{with probability } p \\ 1 & \text{with probability } 1 - p \end{cases} \quad (9)$$

Now, using the DOT-MULTIPLICATION operation will yield the desired result:

$$\overleftrightarrow{\Omega}_{new} = \overleftrightarrow{\mathcal{M}} \cdot \overleftrightarrow{\Omega}_{old} \quad (10)$$