Evolutionary Algorithms for the Detection of Structural Breaks in Time Series

[Extended Abstract]

Benjamin Doerr
Max-Planck-Institut für Informatik
66123 Saarbrücken, Germany
doerr@mpi-inf.mpg.de

Paul Fischer
DTU Compute
2800 Lyngby, Denmark
pafi@dtu.dk

Astrid Hilbert
Linnaeus University
351 95 Växjö, Sweden
astrid.hilbert@lnu.se

Carsten Witt
DTU Compute
2800 Lyngby, Denmark
cawi@imm.dtu.dk

ABSTRACT

Detecting structural breaks is an essential task for the statistical analysis of time series, for example, for fitting parametric models to it. In short, structural breaks are points in time at which the behavior of the time series changes. Typically, no solid background knowledge of the time series under consideration is available. Therefore, a black-box optimization approach is our method of choice for detecting structural breaks. We describe a evolutionary algorithm framework which easily adapts to a large number of statistical settings. The experiments on artificial and real-world time series show that the algorithm detects break points with high precision and is computationally very efficient.

A reference implementation is availble at the following address:
`http://www2.imm.dtu.dk/~pafi/SBX/launch.html`

1. INTRODUCTION

We describe an evolutionary algorithm-approach for the problem of detecting structural breaks in time series. A structural break is a point in time where the behavior of the time series changes. What precisely a "change of behavior" is depends on the application. It might be a jump in the numerical values of the observed data or a change of the magnitude in the local variance (called *volatility* in financial mathematics). Often a statistical model is assumed, that is, the time series is assumed to be generated by a particular stochastic process. In this case, a structural break is defined as a "substantial change" of the type of the underlying model or of its parameters, see e.g., Davis et al. [1].

It should be clear from the above discussion that there cannot be a single algorithm for the detection of structural breaks. We would like to provide a generic framework for the design of such algorithms, where the user has to supply the application-specific knowledge, basically a procedure to evaluate how good a set of points is as structural breaks. In our approach some candidate break points are selected. Then the user-supplied fitness function is used to evaluate, how good they correspond to structural breaks. Afterwards the points are moved, new points are added, or existing ones are removed. This process is iterated until a stop criterion is met.

Copyright is held by the author/owner(s).
GECCO'13 Companion, July 6–10, 2013, Amsterdam, The Netherlands.
ACM 978-1-4503-1964-5/13/07.

2. AN EA SOLUTION

The statistical problem is stated as follows: A univariate time series is a sequence $Y = y_0, \ldots, y_{T-1}$ of real numbers, where y_t is the observation at time t. For notational convenience assume that the indexing is by time, i.e., that the series is equidistant, however, this not essential. The *break points* are an integer sequence $b_0, \ldots, b_{k-1}$, where $b_i \in \{0, \ldots, T-1\}$, and $b_i < b_{i+1}$, for $i = 0, \ldots, k-2$. We assume that $b_0 = 0$ as it is the starting point of the first interval. With each break point b_i additional *model data* B_i might be associated, e.g., parameters for the model which is valid in the interval $[b_i, b_{i+1} - 1]$.

A candidate *solution string* $X = x_0, \ldots, x_{T-1}$ is defined as a sequence of length T. Each x_i either is the symbol $*$, meaning "no break point at i" or B_i for "break point at i", where the possible parameters of B_i are those of the model valid for the interval $[b_i, b_{i+1} - 1]$, e.g.

$$\begin{array}{lccccccc} \text{index:} & 0 & 1 & \cdots & b_1 & \cdots & b_2 & \cdots & T-1 \\ \text{string:} & B_0 & * & \cdots & B_1 & \cdots & B_2 & \cdots & * \end{array} \quad (1)$$

The fitness $f(X)$ of X is a non-negative real number. In the application to structural breaks, the fitness function is often a sum of a number of terms in the following way: Every interval $[b_i, b_{i+1} - 1]$ between two successive break points contributes a positive term $f(b_i, b_{i+1})$, e.g., the goodness of fit of the statistical model described by B_i on that interval. When using goodness of fit, usually a perfect fit can be achieved, by making every point t in time a breakpoint and fitting a model to every single observation y_t. In order to control the number of break points one introduces a penalty term $p(k)$ which is a positive, increasing function of the number k of breakpoints and which is subtracted from the fitness. Examples of such fitness functions can be found in the full paper.

We use the following operations. The *uniform crossover* operation on two parent strings $X = x_0, \ldots, x_{T-1}$ and $X' = x'_0, \ldots, x'_{T-1}$ is defined in the usual way: A child string $C = c_0, \ldots, c_{T-1}$ is constructed by choosing the values from the parents with equal probabilities, i.e.,

$$c_i = \begin{cases} x_i, & \text{with probability } 1/2; \\ x'_i, & \text{with probability } 1/2. \end{cases}$$

The *one-point crossover* operation on two parent strings $X = x_0, \ldots, x_{T-1}$ and $X' = x'_0, \ldots, x'_{T-1}$ is defined in the usual way: An index k is chosen uniformly at random from $0, 1, \ldots, T-1$. The child C consist of the first k values from X and the last $T-k$ values from X', i.e.,

$$C = x_0, \ldots, x_{k-1}, x'_k, \ldots, x'_{T-1}.$$

Remark: The reason for using both types of crossover is based on empirical evidence. Uniform crossover quickly improves the fitness. One-point crossover supports the building block structure of the underlying problem. Both crossover operations maintain the expected number of break points. Based on empirical evaluations we have defined the crossover operations to produce only one child.

For the *mutation* operation on string $X = x_0, \ldots, x_{T-1}$. consider index i. Let p_m be the probability to perform any mutation at position i. If a mutation is performed at i, let p_b and $p_e = 1 - p_b$, denote, respectively, the probabilities to introduce a breakpoint at i and to (possibly) erase an existing break point at i. For the application considered, introducing a break point at position i means to generate a breakpoint b and a model description B. Details on how this is implemented for various statistical models can be found in the full paper.

2.1 Outline of the Basic Algorithm

The input to the procedure is a time series Y of length T. The algorithm starts by initializing a population of M strings of length T each having k_s break points, where k_s is an educated guess for the true number of break points.

The algorithm proceeds in rounds, in every round exactly one of the operations (one-point crossover, uniform crossover, mutation) is performed. The parent strings X and X' (only X in case of a mutation) are selected at random proportional to their fitness. Specifically, the fitnesses are scaled such that they form a probability distribution according to which the strings are drawn. Instead of using the fitness for the selection one can also use the squared or exponentiated fitnesses in order to increase the selection probability of the fittest strings.

The resulting child string C will replace the string $X_{\min} \in \mathcal{X}$ with minimum fitness with probability $f(C)/(f(C) + f(X_{\min}))$.

The algorithms is terminated after R rounds or when the maximum fitness of the best string in $\mathcal{X}$ has not been improved within the last R' rounds. The string with maximum fitness in the current population is returned.

Remark The decision for not performing a combination of operations in every step but only a single one is based on empirical evidence. It seems that mutations "spoil" the result of crossovers. This happens especially in the later phase, when a population already has a number of good strings. The improvement of the fitness is slower when allowing a mutation on top of crossover on the same string.

2.2 Modifications of the Algorithm

The following modifications of the algorithm have been implemented and evaluated.

Island model: A number of copies of the algorithm is run, each with its own population, and from time to time some individuals migrate between populations.

Multi-start: A number of copies of the algorithm, each with its own population, is run in parallel until termination. The overall best solution string is returned.

Champions league: This uses a greedy initialization of the evolutionary algorithm and applies to both of the just described modifications. The fittest strings from each population are collected into a set $\mathcal{C}$ of champions. The evolutionary algorithm is then run with $\mathcal{C}$ as initial population. If $\mathcal{C}$ is very large, one can use the M fittest strings from $\mathcal{C}$.

The experiments showed, that multi-start with champions-league gave the best results.

2.3 Implementation Issues

For our application the number of break points will be small as compared to the length of the time series, 1% or less. An explicit representation of a string as in (1) would then mainly consists of $*$-symbols, which is highly inefficient. Therefore, a string is implemented as a doubly linked list of the pair consisting of indices and model data sets B_i.

$$[b_0; B_0] \longleftrightarrow [b_1; B_1] \longleftrightarrow \cdots \longleftrightarrow [b_k, B_k] \longleftrightarrow [T; -]$$

The last entry is the pair $[T; -]$ which indicates the position one after the end of the time series is added for convenience of the implementation. We assume $b_0 = 0$. Then crossover and mutation can be performed in time proportional the number of break points. For the mutation we have to avoid to perform a coin toss at each index, see [2]. Let p_m denote the probability that an action (insertion or erasion of a break point) is done at a given position. The distances between these positions are distributed according to a geometric distribution with parameter p_m. We traverse the string starting at index $i = 0$. When arriving at index i we generate an integer d from the geometric distribution and perform a mutation at $i + d$. We continue from $i + d$ in the same way until we reach an index that is at least T.

Even though crossover and mutation can be performed in sub-linear time for the problem under consideration, the computation of the fitness will often still depend on all T observations of the time series and thus require at least linear time (in the length of the series). In the full paper we given an example, where also the fitness can be computed in time proportional to the number of break points multiplied by $\log(T)$.

3. REFERENCES

[1] R. Davis, T. Lee, and G. Rodriguez-Yam. Break detection for a class of nonlinear time series models. *J. of Time Series Analysis*, 29:834–867, 2008.

[2] T. Jansen and C. Zarges. Analysis of evolutionary algorithms: From computational complexity analysis to algorithm engineering. In *Proc. of the 11th ACM SIGEVO Workshop on Foundations of Genetic Algorithms (FOGA 2011)*, pages 1–14. ACM Press, 2011.