Memetic Genetic Algorithms for Time Series Compression by Piecewise Linear Approximation

Tobias Friedrich, Martin S. Krejca, J. A. Gregor Lagodzinski, Manuel Rizzo, and Arthur Zahn

Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
{tobias.friedrich, martin.krejca, gregor.lagodzinski}@hpi.de,
arthur.zahn@student.hpi.de

1 Appendix

We go into detail about the PLA-GA from Sec. 3. After acceptance, we will make our source code publicly available in an open repository, with all of the following details in its description.

1.1 The GA Operators

We describe Alg. 1 in detail, except for lines 2 and 9, which are operators specific for time series and are explained in Sec. 3.3. The PLA-GA mostly follows a standard $(\mu + \lambda)$ GA using one-point crossover. However, since each individual needs to be valid, we include specific repair and mutation operators, which satisfy this constraint. Note that, for all random choices, we break ties uniformly at random, and that all sets in the following are multisets. For an individual $x \in \{0,1\}^n$, we let $|x|_1$ denote the number of 1s.

Before crossover, given a parent population P of μ individuals, we determine a multiset L of individuals used for crossover by drawing λ individuals from P via ranking selection [14] (line 5). That is, the i-th best individual – the best individual has the lowest fitness, since we minimize the MSE – is picked with a probability of $2(\mu+1-i)/\big(\mu(\mu+1)\big)$, independently for each draw. We use ranking and not fitness-proportionate selection, as the fitness can differ drastically throughout the run, while the ranking provides a consistent distribution for selection.

Afterwards (line 6), we partition L into $\lfloor L/2 \rfloor$ pairs; if $|L|$ is odd, we are left with an unpaired individual x^*. Then, for each pair, we perform one-point crossover, yielding two offspring. These $2\lfloor L/2 \rfloor$ offspring together with x^*, if existent, comprise the offspring population O. Note that $|O| = \lambda$.

Since the crossover can lead to invalid individuals, we fix the number of 1s for each individual $x \in O$ as follows (line 7). If $|x_i|_1 = m + \ell$ for an $\ell \in \mathbb{N}^+$, we choose a subset $F \subseteq \{i \in [2..n-1] \mid x_i = 1\}$ with $|F| = \ell$ uniformly at random among all such subsets, and we flip the bits in F. Analogously, if $|x_i|_1 = m - \ell$, we choose a subset F with exactly ℓ 0s and flip these.

Next, we mutate each of the individuals $x \in O$ as follows (line 8). For each position $i \in [2..n-1]$ (in increasing order) with $x_i = 1$, with a probability of α, we choose an offset $d \in \mathbb{Z}$ and switch the bits at i and at $\max\{2, \min\{i+d, n-1\}\}$. To promote local changes, we roughly emulate the harmonic step operator [6], which weights distances with $\frac{1}{|d|}$, and cap it at $|d| < \frac{n-2}{m}$ since $\frac{n-2}{m}$ is roughly the average distance between 1s in x. In detail, we draw $r \in [0,1)$ uniformly and set $|d| = \left\lfloor \left(\frac{n-2}{m}\right)^r \right\rfloor$ while drawing the sign of d uniformly at random.

Last (line 10), we perform a plus-selection among the multiset $P \cup O$.

1.2 The Time Series Operators

In addition to the $(\mu + \lambda)$ GA outline presented in Sec. 3.2, the PLA-GA makes use of the following two times-series-specific operators: *seeding* (line 2) and *local search* (line 9), which improve the performance of PLA-GA in the domain of time series. We now explain both operators in detail.

Seeding. Seeding is the method of adding special individuals *(seeds)* to the initial population, which are not drawn uniformly at random. We generate seeds using established compression heuristics from the literature in order to accelerate the search. While any number $k \in \mathbb{N}$ of heuristics can be used for this step, we choose $k = 3$ in this paper, each heuristic generating exactly one seed, in order to not introduce a too strong bias into the initial population.

The compression heuristics that we use are Ramer–Douglas–Peucker [8,25], Visvalingam [30], and Swing Filter [15], all of which compress in different ways, thus adding diversity to the seeds. This makes the seeding more robust and counteracts the domination of a single solution from the start. Note that only the time points of the generated solutions are preserved during seeding, as their values get reassigned due to the representation. Recall that S denotes a time series of length n and that C denotes a compression of S of length m.

- **Ramer–Douglas–Peucker** [8,25] constructs C by greedily choosing m points from S. Initially, the algorithm adds S_1 and S_n to the compressed series and goes through $m-2$ iterations of adding a new time point from S. Let D denote the compressed series at the beginning of an iteration, then the point S_i with $i = \arg\max_{j \in [1..n]} |v_j^S - D^*(t_j^S)|$ has the biggest error to D and is added to the compressed series. Note that the original algorithm is based on recursion and a maximum error parameter as stopping criterion; we modified the algorithm to use a priority queue so it can take the desired length m as a parameter and then stops once C reaches this length.
- **Visvalingam** [30] starts with the whole time series and iteratively removes the point whose removal will introduce the smallest error according to the following heuristic. Let D denote the currently remaining time series at the beginning of an iteration, for $i \in [2..|\mathrm{dom}(D)| - 1]$, the error of removing D_i from D is estimated as follows: Let E be D without D_i, then we estimate the error as $\int_{t_{i-1}^D}^{t_{i+1}^D} \left(C^*(t) - E^*(t)\right)^2 \mathrm{d}t$ to roughly emulate the square error of the MSE.
- **Swing Filter** [15] receives a maximum error bound $\varepsilon \in \mathbb{R}_{>0}$ and creates a solution C fulfilling $\max_{i \in [1..n]} |v_i^S - C^*(t_i^S)| \leq \varepsilon$. After initially adding S_1 to C, it traverses S and adds new points when necessary to fulfill the constraint. Let C_i be the last point added to C and ℓ be such that $t_\ell^S = t_i^C$, then the algorithm determines the maximum $j \in [\ell + 1..n]$ such that a slope $a \in \mathbb{R}$ exists with $\max_{b \in [\ell..j]} |v_b^S - (v_i^C + (t_l^S - t_i^C)a)| \leq \varepsilon$. It adds a new point with $t_{i+1}^C = t_j^S$, where v_{i+1}^C is chosen such that the squared error of the new line is minimized while still respecting the ε-bound. The last point is added when $j = n$, after which the algorithm terminates. It is not possible to directly control the length of the generated C, though it tends to increase when ε decreases. To find a fitting ε, we wrap Swing Filter into a binary search with limited depth. The resulting C still might have an incorrect length but is added to the population nonetheless as its offspring will be repaired.

Local Search. We propose a novel local search algorithm, which improves a compressed time series C by greedily repositioning each point individually. It

iterates through the points of C and repositions each point locally optimally between both of its neighbors as explained below. For the resulting series L, we have $\mathrm{MSE}(L, S) \leq \mathrm{MSE}(C, S)$. We start by setting $L_1 = C_1$ and $L_m = C_m$. For $i \in [2..n-1]$ in increasing order, we determine L_i as follows. Let D denote the time series consisting of only L_{i-1}, L_i and C_{i+1}. We select $L_i \in (t^L_{i-1}, t^C_{i+1}) \cap \mathrm{dom}(S) \times \mathbb{R}$ such that $\mathrm{MSE}(D, S|[t^L_{i-1}, t^C_{i+1}])$ is minimized.

As the result L gets converted into an individual, the values of its points get reassigned. Applying the local search multiple times has the potential of enhancing a solution even further though it eventually converges.