BIO-INSPIRED OPTIMIZATION OF PARAMETRIC ONSET DETECTORS

Domenico Stefani and Luca Turchet

Department of Information Engineering and Computer Science
University of Trento
Trento, Italy
domenico.stefani@unitn.it | luca.turchet@unitn.it

ABSTRACT

Onset detectors are used to recognize the beginning of musical events in audio signals. Manual parameter tuning for onset detectors is a time consuming task, while existing automated approaches often maximize only a single performance metric. These automated approaches cannot be used to optimize detector algorithms for complex scenarios, such as real-time onset detection where an optimization process must consider both detection accuracy and latency. For this reason, a flexible optimization algorithm should account for more than one performance metric in a multi-objective manner. This paper presents a generalized procedure for automated optimization of parametric onset detectors. Our procedure employs a bio-inspired evolutionary computation algorithm to replace manual parameter tuning, followed by the computation of the Pareto frontier for multi-objective optimization. The proposed approach was evaluated on all the onset detection methods of the Aubio library, using a dataset of monophonic acoustic guitar recordings. Results show that the proposed solution is effective in reducing the human effort required in the optimization process: it replaced more than two days of manual parameter tuning with 13 hours and 34 minutes of automated computation. Moreover, the resulting performance was comparable to that obtained by manual optimization.

1. INTRODUCTION

Audio Onset Detection (OD) is the process of detecting the beginning of musical notes in audio signals and is typically used for music database analysis tasks, such as automatic music transcription or query by humming, and interactive music systems, such as novel smart musical instruments. In OD research, two main application scenarios can be distinguished: offline and real-time OD. Music database analysis can be performed with offline OD, which means that the recognition algorithm is applied to whole audio recordings, and detection time is not critical. On the other hand, real-time OD operates at a constant rate on an audio stream, and the detection must be performed before predefined deadlines. For this reason, real-time detection algorithms can only use the portion of the audio signal that is available at a given time along with past history (i.e., it cannot look into the future without increasing the latency).

To date, most research on OD methods has focused on offline applications [1, 2, 3] and real-time cases with rather generous time constraints [4]. For these purposes, a wide range of probabilistic methods have been developed, such as deep learning approaches, which typically run on PCs or powerful computers. Less attention has been devoted by researchers to the development and use of OD methods that are specifically suited for hard real-time scenarios and can run on embedded devices, such as the Raspberry Pi or BeagleBone Black single-board computers, which have limited computing power compared to a regular PC [5]. The use of such methods is particularly relevant to some applications in the emerging field of smart musical instruments [6], which may repurpose the information extracted from the signal with imperceptible latency for the player (see, e.g., [7]).

Deterministic (i.e., non-probabilistic) OD algorithms are suitable for such applications because they require less computational effort than existing methods based on neural networks [1, 4] and, therefore, comply with the limitations of embedded devices. However, these algorithms require precise tuning of their parameters to achieve high recognition accuracy low recognition latency. Such tuning can be performed manually or with grid search, by measuring the performance of the detector on a target data set of audio recordings for different parameter values. Both tuning processes can take a lot of time and can be impractical. Moreover, due to the prevalence of offline applications, automated solutions used in research often optimize a single objective function (i.e., detection accuracy), while real-time applications require the optimization of multiple metrics (i.e., detection accuracy and latency).

In this paper, we propose a general framework for parameter tuning and performance evaluation of time-constrained real-time onset detectors, which can be applied to any parametric OD tool with minor modifications. To optimize detector parameters, our method uses an Evolutionary Computation (EC) algorithm that models solutions as individuals of a population and parameters as genetic material. The EC population is updated iteratively using techniques inspired by natural evolution such as selection of the fittest, reproduction, mutation of genetic material, and generational replacement. The selection drives the evolution towards a goal by allowing the best solutions to take part in the reproduction and mutation processes in order to produce better new solutions. Moreover, the proposed method is suitable for multi-objective optimization.

The proposed procedure was applied to the OD algorithms of the free and open-source Aubio library in order to optimize the detector for a real-time timbre recognition method for acoustic guitars. The target system should manage to detect each onset in the audio signal and use a classifier to determine which playing technique the guitarist is using. The classification information is intended to be repurposed in real-time to play different sounds. Complex sequential sounds that are separated by less than 30 ms are generally perceived by the human hearing system as simultaneous [8], therefore this interval should be the maximum end-to-end

latency of the recognition and repurposing system. A maximum target latency of 20 ms was considered for the timbre recognition algorithm alone, which can be split between the OD, feature extraction, and classification tasks. It was found that the sum of the last two tasks in the pipeline consistently takes 6 milliseconds, giving a maximum latency of 14 ms for the OD task. Moreover, the variability of the detection latency across different sounds should be low, so that the actual onset time can be estimated by subtracting a fixed interval from the time of detection (see, e.g., [5]).

The following is the outline of our paper. In Section 2 we discuss various studies related to onset detection and evolutionary computation in music contexts. Section 3 describes the proposed method for multi-objective optimization of onset detectors. Section 4 reports the details of the application and evaluation of our method. Finally, we draw our conclusions in Section 5.

2. RELATED WORKS

2.1. The Aubio library

The Aubio library [9, 10] is a free and open-source library designed for audio feature extraction. At its current released version (0.4.9) the Aubio library is in active development, and since its inception, numerous improvements and algorithms were added to its functionalities [1]. One example of relevant improvement is the addition of Adaptive Whitening [2] [11], which is a method for preprocessing spectral frames that normalizes the magnitude of each frequency bin with respect to a recent maximum value for the bin. Such algorithm mitigates the effect of spectral roll-off and variations in the dynamic of the audio signal, thus improving the performance of various detection methods of the library. An exception to the improvement offered by Adaptive Whitening is the Modified Kullback–Leibler (MKL) distance method [9, p. 42, formula 2.9], which was proven to show detrimental effects, as also confirmed by the present study.

The onset methods currently implemented in the Aubio library are:

1. **energy:** Energy-based distance, which calculates the local energy of the input spectral frame.
2. **hfc:** High-Frequency content [12], which computes the high frequency content of signal. It has shown to be efficient at detecting percussive onsets.
3. **complex:** Complex domain OD function [13], which uses information both in frequency and in phase to determine changes in the spectral content of the signal that might correspond to musical onsets.
4. **phase:** Phase-based OD function [14], which uses information both in frequency and in phase to determine changes in the spectral content of the audio signal.
5. **specdiff:** Spectral difference OD function [15].
6. **kl:** Kullback–Leibler OD function [16].
7. **mkl:** Modified Kullback–Leibler OD function [10, Chapter 2].
8. **specflux:** Spectral flux [17].

While most of the research in the field has moved to probabilistic and data-driven approaches [1, 2, 4, 3] (applied either to offline contexts or real-time contexts with relaxed time constraints), deterministic OD methods still offer the guarantee to be able to run on tight time constraints for the real-time case. These methods are also less computationally expensive than most of the neural network-based counterparts (see e.g., [1, 4]), which is an essential requisite for embedded devices.

The Aubio library has been available for a long time, however, it can be verified that active development has been maintained over the years. Furthermore, most of the available audio libraries (e.g., Essentia[3] [18], Librosa [19], Madmom [20]) offer a very similar selection of deterministic OD methods, with the sole exception of some advanced methods (e.g., SuperFlux [21]), which tend to improve the performance with soft onsets and polyphonic recordings (not necessary for this study since we performed detection on monophonic recordings). On top of this, Aubio offers a very complete and easy-to-use library with the addition of command line executable programs that were of great aid when doing the performance evaluation reported below.

2.2. Evolutionary Computation

EC is a family of optimization algorithms inspired by natural evolution. These algorithms generate an initial population of random solutions of the optimization problem. Each solution is defined by a set of parameter values called "genotype". Individuals in the population are then selected for reproduction based on how "fit" they are. One or more crossover operators are used to combine the genetic material of the "good" solutions that were selected. Then, random mutation can be applied to the genes of the offspring to add randomness and help individuals to get out of local optima in the optimization landscape. Finally, the population is replaced by some or all of the offspring depending on a recombination operator, and the evolutionary process is repeated for the new generation. The degree of fitness of an individual is determined by an objective function that depends on the optimization problem. The evolutionary process explores the optimization landscape and searches for the global optimum.

The exploratory nature of EC algorithms lends itself well to the creative areas of generative audio synthesis and algorithmic composition. A typical use of evolutionary algorithms in sound synthesis is to create systems that can optimize sound parameters to achieve certain target sounds (see [22, 23]). The study reported in [24] employs two stages of evolutionary optimization to suggest topological arrangements for the functional elements of a sound synthesis algorithm, as well as to optimize the internal parameters of these elements. While the aforementioned approaches employ a mathematical formulation for the fitness function of EC algorithms, Johnson [25] uses an EC algorithm in an interactive manner by having a user subjectively evaluate the fitness of each individual in the population. Similar approaches are taken in the field of evolutionary music composition, where some authors choose a specific composition goal and define a fitness function that drives evolution towards it (see e.g., [26, 27]), while others use the interactive approach with a user-evaluated fitness function (see [28, 29]).

Vatolkin et al. [30] apply EC algorithms to the field of music information retrieval by optimizing feature selection for a musical instrument classifier: their approach is to use the performance of the machine learning classifier as a fitness function. This appli-

[1] https://github.com/aubio/aubio/releases

[2] Added in Aubio version 0.4.5 https://aubio.org/pub/aubio-0.4.5.changelog

[3] http://essentia.upf.edu

cation shows that EC algorithms perform well when there is no derivable objective function for the problem. The same advantage is shown by Faragó et al. [31] who successfully applied an EC algorithm to the design of the sound processor of a hearing aid. Similarly, Pepe et al. [32] employ two different EC algorithms for tuning filter parameters of a multichannel audio equalization system to match the desired frequency response.

Finally, while research on EC for musical onset optimization is scarce, more than one study applied EC algorithms to Electromyography onset detection for muscle activation analysis [33, 34]. Even though electromyographic OD is a different task than musical OD, the optimization strategy shown is similar. In particular, Rashid et al. [33] describe how an EC can be applied to any detection algorithm that is deemed appropriate for a given problem.

The evolutionary optimizer utilized in the present study was developed using the Inspyred Python library [35]. Inspyred provides a wide range of bio-inspired algorithms, including EC and swarm intelligence, which is well documented, easy to use, and customizable as needed.

3. PROPOSED METHOD

The procedure we devised for the optimization of an onset detector performance considers both detection accuracy and latency (i.e., the time interval between an onset and its reporting). It can be summarized with the following steps:

1. Create and annotate an input set of audio recordings for the onset detector. The dataset can then be divided into an optimization set and a test set;
2. Prepare an evaluation algorithm that, given a set of input parameter values, can execute the detector on the input data and compute the metrics of interest;
3. Identify the parameters that dictate the latency of the computation (A-parameters) and separate them from those that can be optimized without affecting it (B-parameters). Other parameters may need to be set to predefined values to meet problem constraints (fixed parameters);
4. Run the EC algorithm for each combination of the A parameters of interest, considering the list of B-parameters as the genotype of the EC algorithm. At this stage the performance on the optimization set is used to drive the evolution;
5. Once the best results for each combination of the A-parameters are obtained, find the Pareto front (or tradeoff curve) considering as objectives both the accuracy metric of choice and a metric that describes the latency distribution;
6. Select a solution that offers an appropriate compromise between the two competing metrics. All the solutions in the Pareto tradeoff curve are viable. The final choice of parameters can be evaluated on the test data to ensure that it can perform at best on new data.

The first step is to prepare a relevant recording sample and annotate every onset in it. Stratified random sampling can be used to maintain the characteristics of interest of the original dataset. The time resolution of the labels can vary depending on the application of interest and the labeling process can be carried out with the help of an annotation software. Furthermore, the same recordings can be labeled by more than one annotator to reduce potential errors [10, p. 52]. The dataset can then be divided into an optimization set and a smaller test set: the first will be used to drive the optimization process, while the second will be used to test the generalization performance of the final solution on new data. The dataset used for this study is described in Section 4.1.

Step 2 involves the development of an "evaluator" program that computes a measure of the success of the onset detector on the input data. A tolerance window can be defined around each hand-labeled onset and any detection that falls within this window can be considered correct [10, Chapter 2.5.2]. In addition to correct detections, errors can be divided into false positives and false negatives: the former are onsets that are detected outside a tolerance window, while the latter describe true onsets that are not detected. Duplicate detections (more than one onset detected in the same window) can either be counted separately or considered as false positives (except for the first correct detection). Different ratios can be calculated with these categories: Precision is the ratio of correct detections to the number of detected onsets, while Recall is the ratio of correct detections to the number of labeled onsets. Precision and Recall can be combined using the F1 score, which is defined as the harmonic mean of the two (Eq. 1).

$$F_1 = 2 \times \frac{P \times R}{P + R} \tag{1}$$

Step 3 consists in categorizing the parameters of the detector: the first classification can distinguish between fixed and free parameters, where the values of the former are given by the problem at hand and the latter can be optimized to obtain the best performance. Free parameters can be further divided into the following categories:

1. *A-parameters*: parameters that have an impact on detection latency
2. *B-parameters*: parameters that have no direct impact on detection latency.

Then, a set of combinations of interest of the A-parameters can be selected so that multiple single-objective optimization problems can be solved to find the values that give the best accuracy metric for each subproblem. The multi-objective optimization, which considers both detection accuracy and latency, is left for a subsequent step.

Step 4 involves optimizing the B-parameters for every combination of the A-parameters. For this task, an EC algorithm is proposed as this class of optimization methods can be very robust and requires no assumption on the problem or the input parameters (black-box optimization). EC algorithms model the candidate solutions of an optimization problem as individuals of a population, and evolve such individuals in a way that is inspired by natural evolution. For each generation, some individuals of the population are selected for breeding, recombination leads to the creation of an offspring, mutation can alter the nature of the generated solutions and finally, the population is replaced by the offspring. The individuals of an EC algorithm are described by their genotype, which is a set of values for the input parameters for the problem, and their phenotype, which is the manifestation of the genotype in the form of fitness of the solution (i.e., a measure of how close the solution is to the global optimum). This evolutionary process leads to the improvement of good solutions, which get closer to global optima, while weak solutions do not survive. Pseudocode for a generic EC algorithm is shown in Alg. 1 (inspired by [36]).

Algorithm 1: Evolutionary Algorithm

```
Generate initial random population
while generation <= max_generation do
    generation = generation+1
    calculate fitness of each individual with the evaluator
    select individuals depending on their fitness
    perform crossover with probability crossover_rate
    perform mutation with probability mutation_rate
    use replacement strategy to create the new population
end
```

In the proposed method, the genotype is a vector of B-parameter values. The phenotype is represented by the value of the accuracy metric of choice and is obtained using the algorithm designed in step 2. For each combination of A-parameters the EC returns the single solution with the highest detection accuracy obtained on the optimization dataset.

The fifth step is to take each solution obtained in the previous step and compute the set of optimal solutions with respect to both the success metric of choice (e.g., F1-score) and a measure of the detection latency (e.g., maximum value or variance). The optimal solutions are obtained by computing the Pareto front, which is the set that contains all the solutions for which none of the objective functions can be improved further, without reducing some of the other objective values. Therefore, all the solutions in the front, which are called non-dominated or Pareto optimal, are equally good from an objective standpoint, and offer different compromises between detection success and latency.

The final step is to select the single most desirable tradeoff between detection accuracy and latency among the optimal solutions in the Pareto front. The selection is subjective and should take into account the set of candidate tradeoffs and the problem domain.

4. EVALUATION

The proposed optimization procedure was evaluated by applying it to all the OD methods offered by the Aubio library. The solutions found by the EC algorithm were compared to the ones obtained by manual optimization.

The source code is made available on an online repository[4].

4.1. Input data

The input data used is a representative sample of a dataset composed of individual monophonic guitar sounds. The dataset was recorded with 6 acoustic guitars and 5 professional guitarists that were asked to play an exhaustive range of sounds with 8 common techniques, 4 of which produce percussive timbres.

The techniques that make use of the strings include using a plectrum over the sound hole or the near the bridge, palm mute and natural harmonics, whereas percussive sounds were produced by striking the guitar body using the following techniques:

1. "Kick" technique: producing a sound that resembles a kick drum by hitting the lower right part of the top of the guitar body.
2. "Tom" technique: producing a sound by hitting the area of the guitar body near the top of the end of the fretboard.
3. "Snare-A" technique: producing a sound by hitting the lower right side of the guitar body.
4. "Snare-B" technique: producing a sound by hitting the muted strings over the last part of the fretboard.

Each percussive sound was recorded at 3 dynamic levels (piano, mezzo-forte, forte) and repeated from 10 to 100 times, while the sounds for "pitched" techniques were recorded for every string on a selection of frets[5] between 0 and 20, and with 3 repetitions for each dynamic level.

The sound signal was recorded from a combination of a condenser microphone and a piezoelectric pickup embedded in each guitar (with the exception of one that included only a piezoelectric transducer) in WAV format, with bit-rate and bit-depth of 48000Hz and 24bit respectively.

For the optimization phase of this study, a sample of 1328 individual sounds was extracted from the main dataset with stratified random sampling. In the sampled dataset, each combination of guitarist and guitar is represented by a number of onsets that is proportional to the original dataset. Moreover, the sampling procedure produces the same number of sounds for each dynamic level. A second sample of 336 sounds was used for the testing phase, where the performance of the optimized detector was measured on data that was not used for the optimization. Finally, the recordings were labeled by one annotator with the open-source Audacity audio software, providing ground truth onset labels for the evaluation of the detector. The data used for this study is publicly available in the project repository.

4.2. Evaluation algorithm

Step 2 consists in developing a program that can measure the success of the onset detector on the input data.

Success was quantified using the average F1-score metric (Eq 1) for each playing technique in the dataset, considering as correct detections only onsets that were detected in a tolerance window of 20 ms that follows each hand-labeled onset (The non centered window starts at each labeled onset and finishes 20 ms later). The exact interval between each labeled onset and its time of detection was measured.

The evaluation program takes as input a set of parameter values for the detector and runs the aubioonset[6] executable that is provided with the library. The executable was modified to produce a text file with a list of the exact times at which onsets are detected. Along with the measure of detection success, the detection latency was recorded for each onset.

The program was developed using the Python programming language and the metrics were calculated using the R language. Finally, the program itself is designed to store any temporary file in a different folder that is created for each execution of the evaluation program, to allow multiple parallel instances to run at the same time without conflict.

4.3. Onset detector parameters

Step 3 involves distinguishing fixed parameters from free parameters. The fixed parameters, whose values are imposed by the

[4] https://github.com/domenicostefani/BioInspiredOnsetDetection

[5] Natural harmonics were played only on frets 5, 7 and 12, while most of the other techniques were recorded from the open string (fret 0) to at least the 15th fret, ending on a fret that depended on physical limitation of each guitar.

[6] https://aubio.org/manpages/latest/aubioonset.1.html

requirements of the problem at hand, were the hop size and the minimum inter-onset interval. The hop size is the number of samples between two consecutive analyses, and determines the rate at which detection is performed: a value of 64 samples was deemed adequate for our problem (64 samples at 48000 Hz equals 1.33 ms). The minimum inter-onset interval value is the shortest time gap between the onsets that the analysis can report, and is imposed to avoid double detections: a value of 20 ms was considered appropriate since it matches the real-time latency requirement described in Section 1, and was found not to affect the detection of successive, reasonably quick, guitar onsets.

Contrary to fixed parameters, free parameters must be optimized, and can be further divided into A and B-parameters. A-parameters are all the settings that directly affect latency, and in the case of Aubio, these are the buffer size and the actual OD function used. In contrast, the B-parameters for Aubio are the silence threshold (in dB) and the onset threshold. The size of the buffer and the algorithm used for detection directly influence latency by determining the number of computations performed at each analysis, while the silence and onset thresholds only modify the signal level considered as the noise floor and how strong a peak must be in order to be considered an onset.

Table 1: *Summary of Aubioonset parameters with the range considered for optimization, their category and whether they affect directly the detection latency or not.*

Aubio Parameter	Optimization Range	Category	Determines detection latency
Hop size	-	Fixed	Yes*
Min. i.o.i.**	-	Fixed	No
Buffer size	[64,2048]	A-par.	Yes
Method	Aubio methods	A-par.	Yes
Silence threshold	[-60 dB, -30 dB]	B-par.	No
Onset threshold	[0.1, 3.6]	B-par.	No

* Hop size affects the detection latency but it is not an A-parameter since it is constrained by the problem requirements (fixed).
** Minimum inter-onset interval.

4.4. Single-objective accuracy optimization

For step 4, a reasonable range of A-parameters was selected, consisting of buffer size values between 64 and 2048 samples (i.e., 64, 128, 256, 512, 1024, and 2048) and all 8 available onset methods, plus MKL with adaptive whitening disabled at initialization. Each combination of A-parameter values requires the optimization of the remaining B-parameters to obtain the best F1-score.

An initial optimization approach was to manually select different B-parameter values and measure performance using the evaluator script. The manual procedure consisted of a coarse grid search, followed by a refinement of the parameters near performance peaks in the search space. This technique was used because the brute-force approach of a fully automated grid search showed to be either too coarse or too time-consuming. Manual optimization proved to be time-consuming and required more than 2 workdays: given the large amount of human effort required and the limited scalability of the manual procedure, an automated EC algorithm was used.

The Inspyred Python was used to devise an EC optimizer. Only a few modifications were required to modify the EC algorithm from the library for the specifics of our problem. In particular, the individuals of the population were specified as vectors, each containing two values for the silence and onset thresholds. Furthermore, the program developed for step 2 was set as the evaluator of the EC algorithm, to be used to calculate the F1-score of each individual (fitness of the individual). The Inspyred library contains several standard evolutionary algorithms such as Genetic Algorithm, Evolution Strategy, and Simulated Annealing as well as a custom EC framework that allows different evolutionary operators to be composed. The custom framework was used, and it was found to work best for the problem at hand with the following settings:

- Population: 30 individuals, where each individual is a vector containing a value for each B parameter. A greater number of individuals could increase the possibilities for improving the solutions even with fewer generations, however, it would require more execution time.
- Evolution termination: automatic termination after 30 generations. The termination strategy can be defined by trying different values and evaluating the fitness plots: if fitness keeps increasing over time the termination deadline can be moved further away, while a stagnating behavior shows that termination can happen earlier.
- Selection strategy: tournament selection with size 4, which holds a "tournament" by randomly sampling n individuals (4 in this case) and choosing the one with the best fitness (see [37]).
- Crossover: Arithmetic and Laplace recombination operators (rate = 0.7), which are common choices for combining good solutions when using real-valued genotypes (see [38, 39]).
- Mutation: Gaussian mutation operator, which adds a random value from a Gaussian distribution to each element of an individual's genotype to produce a new offspring. The probability of performing the mutation (mutation-rate) was set to 0.7, while the mean and standard deviation of the Gaussian distribution were 0 and 3.0 respectively. Mutation helps candidate solutions to move away from potential local optima in the fitness landscape.
- Replacement strategy: Generational Replacement with elitism, meaning that the entire existing population is replaced by the offspring at the end of each generation, except for the best n solutions (in this case with 1 elite), which survive if they are better than the worst n offspring. Elitism helps to preserve the best individuals.

These parameter values were obtained by testing various combinations on a reduced number of generations. For more information on the parameters of EC algorithms, see the Inspyred documentation[7].

4.4.1. *Manual and Automatic optimization comparison*

Eleven parallel executions of the EC algorithm were executed on an Intel® Core™ i7-10750H CPU. The optimization instances for the three highest buffer size values were scheduled on the first nine parallel processes, while the remaining instances were executed on the last two processes (see Fig. 1). This schedule was devised to account for the longer time required for the evaluation of the performance of optimizers with high buffer size values (more overall

[7] `https://pythonhosted.org/inspyred/reference.html`

computations). The last process terminated 13 hours and 34 minutes after the beginning of the optimization.

The best F1-score values obtained in each instance were comparable to those obtained by manual optimization. For both manual and automated optimization, the F1 score was computed on the test dataset, to evaluate the performance of the best parameter configuration on new data. The mean of the difference between the F1-score obtained in the test dataset with the EC and the manual optimization process was 1.4×10^{-3} points and its standard deviation was 1.2×10^{-2}

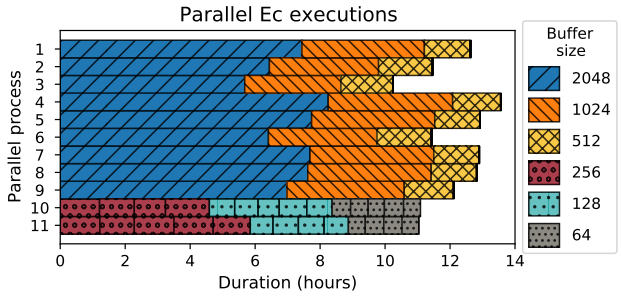

Figure 1: *Parallel EC process schedule. Each stack of bars represents the execution time of one of the parallel processes, while each individual bar indicates a single EC optimization instance.*

Table 2: *F1-score values (percentage) of the best solutions obtained from the EC algorithm for each combination of OD method and buffer size used. The highlighted values are the non dominated solutions of the Pareto front computed in Section 4.5 (Table 3).*

		Buffer size					
		64	128	256	512	1024	2048
Method	hfc	93.37	92.31	91.26	89.68	90.15	88.56
	energy	94.10	94.20	94.80	94.82	95.58	91.63
	complex	83.71	85.09	86.80	87.55	86.66	80.45
	phase	76.20	82.34	87.40	82.06	74.26	71.62
	specdiff	86.67	93.67	95.35	95.29	95.49	93.39
	kl	85.17	86.16	87.95	87.52	89.19	82.41
	mkl	84.84	85.90	87.22	86.96	88.18	88.14
	specflux	**84.49**	91.75	**92.43**	91.21	87.97	86.82
	mkl(noaw)*	**95.18**	97.08	**97.42**	97.38	97.30	96.30

* The MKL method with adaptive whitening disabled on initialization.

4.5. Multi-Objective Optimization

The result of step 4 is a set of solutions that are optimized for each combination of A-parameters, while step 5 is to perform multi-objective optimization to obtain the best overall performance in terms of both detection accuracy and latency.

There two main requirements for our target application are the following:

1. The maximum detection latency must be lower than 14 ms to comply with the end-to-end 20 ms deadline defined in Section 1 for the timbre recognition system. The system is composed of the OD algorithm and a classification algorithm that consistently takes 6 ms to execute, hence the remaining time interval of 14 ms that is assigned to the detection;
2. The variability of the latency distribution must be as low as possible. Thanks to this, it will be possible to estimate the time of the actual onset with high confidence by subtracting a fixed temporal interval from the detection time (e.g., average or maximum detection latency).

For this reason, both the maximum latency and its variability were calculated, in the form of upper Tukey fence and Interquartile Range (IQR) respectively. Tukey fences [40] are values that define the range of a data distribution while ignoring data points that differ significantly from other observations (i.e. outliers). Tukey fences are commonly used to determine the position of box plot whiskers, and are computed using quartile values (i.e., Q_1, Q_2, Q_3) and a constant k, with a value of 1.5 for outliers [40] (Eq. 2).

$$TF = \left[Q_1 - k(Q_3 - Q_1), Q_3 + k(Q_3 - Q_1)\right],\ k = 1.5 \quad (2)$$

On average, 95.7% of the detected onsets were within the upper and lower fences computed. The IQR was used because, unlike variance, it is expressed in the same unit of measure of the distribution data (milliseconds). IQR can also be multiplied by four in order to find the interval between the upper and lower Tukey fences.

Considering the problem as an instance of multi-objective optimization with F1-score and latency IQR as objectives, the Pareto front was found. This front is a set containing all the best tradeoff results that are not dominated by any other solution: this means that the points in the front describe solutions that are equally good for both objectives. All the solutions whose maximum latency was greater than the threshold defined (14 ms) were discarded a priori.

The solutions are shown in Fig. 2 along with the Pareto front (dotted line) and the discarded results, indicated by the larger red circle markers in the upper right area. A more detailed view of the front is shown in Fig. 3 and the solutions are listed in Table 3.

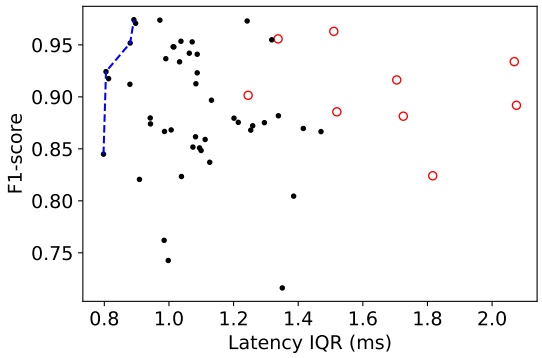

Figure 2: *Solutions of step 4 represented in the space described by the F1-score and the latency IQR objectives. The blue dotted line represents the Pareto front that minimizes IQR and maximizes the F1-score, while the solutions represented with the red hollow circles are the ones discarded because their maximum latency is over the maximum value allowed (14ms).*

Table 3: *Solutions in the pareto front of figures 2 and 3, with F1-score as the first objective and IQR of latency as the second.*

#	Method	Buffer Size	F1-score (%)	Low Tukey fence (ms)	Latency mean (ms)	High Tukey fence (ms)	IQR (ms)
a	specflux	64	84.49	2.2	3.8	5.3	0.80
b	specflux	256	92.43	3.1	4.8	6.3	0.81
c	mkl(noaw)	64	95.18	2.7	4.5	6.2	0.88
d	mkl(noaw)	256	97.42	4.2	6.0	7.7	0.89

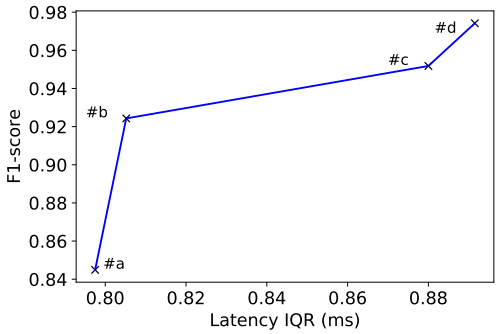

Figure 3: *Closer view of the Pareto front shown in figure 2.*

4.6. Choosing a solution

By definition, the Pareto front computed in step 5 contains solutions that offer different compromises on the two objectives, and it is used to support the subjective choice expressed by a human decision-maker. For this reason, any tradeoff belonging to the Pareto front can be chosen depending on the problem requirements and the shape of the curve.

For the problem at hand, solution *#d* (Fig. 3, Tables 2 and 3) was chosen since it provides the greatest F1-score value without having excessive latency IQR. Solution *#d* was obtained with the MKL method, no adaptive whitening (see Section 2.1), a buffer size of 256 samples, a silence threshold of -51.7 dB, and an onset threshold of 1.18. On the test dataset, solution *#d* obtained an F1-score of 97.33%, an IQR of 0.58 ms, an average latency of 6.0 ms, and lower and upper Tukey fences of respectively 4.8 and 7.2 ms. The results on the test set show that the chosen solution can perform well on new data.

A different set of optimal solutions may need a different choice, and the solution that has the best value according to a single objective may not be the most desirable tradeoff.

5. CONCLUSIONS

In this paper, we presented a procedure for the optimization of the performance of parametric onset detectors on a set of data of interest. The procedure focuses on both improving the detection accuracy, and reducing the latency measured between a true onset and its reporting.

The proposed method was successfully applied to the onset detectors of the Aubio library. An Evolutionary Computation algorithm was used to perform automated single-objective optimization of non-latency critical parameters, and it was compared to manual optimization. The proposed procedure offered a large reduction in the required human effort with respect to manual optimization, while producing comparable results. The reduction in the human effort required for optimization, along with the rather short execution time of the proposed procedure, offers the possibility of using larger datasets.

The F1-score values obtained with the evolutionary computation algorithm showed an average difference of 1.4×10^{-3} F1-score points and a standard deviation of 1.2×10^{-2} with respect to manual optimization. Moreover, the automated algorithm required 13 hours and 34 minutes to compute the best results, while the meticulous manual procedure required over two working days of human effort.

The procedure includes a subsequent step of multi-objective optimization to account for the detection latency, which is performed by defining the objectives of interest and finding the Pareto-optimal solutions. Finally, the best tradeoff between the optimized objectives is selected.

Using this procedure, we were able to significantly improve the success rate of a real-time onset detector while reducing the latency of detection. These improvements were measured on a separate test dataset.

The use of only a sample of an audio dataset can be a limitation: the proposed method could benefit from the use of the entire dataset of interest, however, both human labeling and optimization times would increase. Optimization in such situations can be accelerated by reducing the exploration to more relevant parameter ranges and increasing the number of parallel computations that can be executed. A higher degree of parallelism can be exploited by allowing the evolutionary algorithm to compute the fitness of more than one individual simultaneously, on multiple processing threads.

The proposed procedure is expected to extend to any parametric onset detector algorithm.

6. REFERENCES

[1] F. Eyben, S. Böck, B. Schuller, and A. Graves, "Universal onset detection with bidirectional long-short term memory neural networks," in *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR), Utrecht, The Netherlands*, 2010, pp. 589–594.

[2] J. Schlüter and S. Böck, "Musical onset detection with convolutional neural networks," in *Proceedings of the 6th international workshop on machine learning and music (MML), Prague, Czech Republic*, 2013, pp. 79–82.

[3] A. Roebel, C. Jacques, and A. Aknin, "Mirex 2018: Training cnn onset detectors with artificially augmented datasets," Accompanying abstract for two submissions to MIREX2018-Audio Onset Detection (AR3, AR4), https://www.music-ir.org/mirex/abstracts/2018/AR4.pdf.

[4] S. Böck, A. Arzt, F. Krebs, and M. Schedl, "Online real-time onset detection with recurrent neural networks," in *Proceedings of the 15th International Conference on Digital Audio Effects (DAFx-12), York, UK*, 2012.

[5] L. Turchet, "Hard real time onset detection for percussive sounds," in *Proceedings of the 21st International Conference on Digital Audio Effects (DAFx-18)*, 2018, pp. 349–356.

[6] L. Turchet, "Smart Musical Instruments: vision, design principles, and future directions," *IEEE Access*, vol. 7, pp. 8944–8963, 2019.

[7] L. Turchet, A. McPherson, and M. Barthet, "Real-time hit classification in a Smart Cajón," *Frontiers in ICT*, vol. 5, no. 16, 2018.

[8] B. C. J. Moore, *An introduction to the psychology of hearing*, Brill, 2012.

[9] P. M. Brossier, "Aubio, a library for audio labelling," http://aubio.piem.org, accessed March 23, 2021.

[10] P. M. Brossier, *Automatic annotation of musical audio for interactive applications*, Ph.D. thesis, Centre for Digital music, Queen Mary University of London, London, UK, 2006.

[11] D. Stowell and M. Plumbley, "Adaptive whitening for improved real-time audio onset detection," in *Proceedings of the International Computer Music Conference (ICMC)*, 2007, pp. 312–319.

[12] P. Masri, *Computer modeling of Sound for Transformation and Synthesis of Musical Signal*, Ph.D. thesis, University of Bristol, UK, 1996.

[13] C. Duxbury, J. P. Bello, M. Davies, and M. Sandler, "Complex domain onset detection for musical signals," in *Proceedings of the Digital Audio Effects Workshop*, 2003, vol. 1, pp. 6–9.

[14] J. P. Bello and M. Sandler, "Phase-based note onset detection for music signals," *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03).*, vol. 5, pp. V–441, 2003.

[15] J. Foote and S. Uchihashi, "The beat spectrum: a new approach to rhythm analysis," *IEEE International Conference on Multimedia and Expo (ICME)*, pp. 881–884, 2001.

[16] S. W. Hainsworth and M. Macleod, "Onset detection in musical audio signals," in *Proceedings of the International Computer Music Conference (ICMC)*, 2003.

[17] S. Dixon, "Onset detection revisited," in *Proceedings of the 9th International Conference on Digital Audio Effects (DAFx-06)*, 2006, vol. 120, pp. 133–137.

[18] D. Bogdanov, N. Wack, E. Gómez, S. Gulati, P. Herrera, O. Mayor, G. Roma, J. Salamon, J. R. Zapata, and X. Serra, "Essentia: an audio analysis library for music information retrieval," in *International Society for Music Information Retrieval Conference (ISMIR'13)*, Curitiba, Brazil, 04/11/2013 2013, pp. 493–498.

[19] B. McFee, C. Raffel, D. Liang, D. P. W. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th Python in Science conference (SciPy)*, 2015, vol. 8, pp. 18–25.

[20] S. Böck, F. Korzeniowski, J. Schlüter, F. Krebs, and G. Widmer, "madmom: A new python audio and music signal processing library," *Proceedings of the 24th ACM international conference on Multimedia*, 2016.

[21] S. Böck and G. Widmer, "Maximum filter vibrato suppression for onset detection," in *Proceedings of the of the 16th International Conference on Digital Audio Effects (DAFx-13). Maynooth, Ireland*, 2013, vol. 7.

[22] K. A. Yuksel, A. Ercil, and B. Bozkurt, "Digital sound synthesis via parallel evolutionary optimization," in *2012 20th Signal Processing and Communications Applications Conference (SIU)*, 2012, pp. 1–4.

[23] J. Manzolli, A. Maia, J. Fornari, and F. Damiani, "The evolutionary sound synthesis method," in *Proceedings of the 9th ACM International Conference on Multimedia*, New York, NY, USA, 2001, MULTIMEDIA '01, p. 585–587, Association for Computing Machinery.

[24] R. A. Garcia, "Growing sound synthesizers using evolutionary methods," in *Proceedings of the Workshop on Artificial Life Models for Musical Applications (ALMMA)*, 2001, pp. 99–107.

[25] C. G. Johnson, "Exploring sound-space with interactive genetic algorithms," *Leonardo*, vol. 36, no. 1, pp. 51–54, 2003.

[26] A. Moroni, J. Manzolli, F. V. Zuben, and R. Gudwin, "Vox populi: An interactive evolutionary system for algorithmic music composition," *Leonardo Music Journal*, pp. 49–54, 2000.

[27] M. Scirea, J. Togelius, P. Eklund, and S. Risi, "Metacompose: A compositional evolutionary music composer," in *International Conference on Computational Intelligence in Music, Sound, Art and Design*. Springer, 2016, pp. 202–217.

[28] V. M. Marques, C. Reis, and J. A. T. Machado, "Interactive evolutionary computation in music," in *2010 IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 2010, pp. 3501–3507.

[29] N. Tokui and H. Iba, "Music composition with interactive evolutionary computation," in *Proceedings of the 3rd International Conference on Generative Art*, 2000, vol. 17, pp. 215–226.

[30] I. Vatolkin, M. Preuß, G. Rudolph, M. Eichhoff, and C. Weihs, "Multi-objective evolutionary feature selection for instrument recognition in polyphonic audio mixtures," *Soft Computing*, vol. 16, no. 12, pp. 2027–2047, 2012.

[31] P. Faragó, C. Faragó, S. Hintea, and M. Cîrlugea, "An evolutionary multi-objective optimization approach to design the sound processor of a hearing aid," in *International Conference on Advancements of Medicine and Health Care through Technology; 5th–7th June 2014, Cluj-Napoca, Romania*. Springer, 2014, pp. 181–186.

[32] G. Pepe, L. Gabrielli, S. Squartini, and L. Cattani, "Evolutionary tuning of filters coefficients for binaural audio equalization," *Applied Acoustics*, vol. 163, pp. 107204, 2020.

[33] U. Rashid, I. K. Niazi, N. Signal, D. Farina, and D. Taylor, "Optimal automatic detection of muscle activation intervals," *Journal of Electromyography and Kinesiology*, vol. 48, pp. 103–111, 2019.

[34] M. Magda, A. Martinez-Alvarez, and S. Cuenca-Asensi, "Mooga parameter optimization for onset detection in emg signals," in *New Trends in Image Analysis and Processing – ICIAP 2017*, S. Battiato, G. M. Farinella, M. Leo, and G. Gallo, Eds. 2017, pp. 171–180, Springer International Publishing.

[35] A. Garrett, "inspyred: Bio-inspired algorithms in python," https://pypi.python.org/pypi/inspyred (accessed March 23, 2021).

[36] K. P. Ferentinos, K. G. Arvanitis, and N. Sigrimis, "Heuristic optimization methods for motion planning of autonomous agricultural vehicles," *Journal of Global Optimization*, vol. 23, no. 2, pp. 155–170, Jun 2002.

[37] A. Brindle, *Genetic algorithms for function optimization*, Ph.D. thesis, University of Alberta, 1980.

[38] Z. Michalewicz and J. Arabas, "Genetic algorithms for the 0/1 knapsack problem," in *International Symposium on Methodologies for Intelligent Systems*. Springer, 1994, pp. 134–143.

[39] K. Deep and M. Thakur, "A new crossover operator for real coded genetic algorithms," *Applied Mathematics and Computation*, vol. 188, no. 1, pp. 895–911, 2007.

[40] John W Tukey et al., *Exploratory data analysis*, vol. 2, Reading, Mass., 1977.