fGAAM: A fast and resizable genetic algorithm with aggressive mutation for feature selection

Izabela Rejer¹ · Jarosław Jankowski¹

Received: 19 January 2021 / Accepted: 5 June 2021 / Published online: 20 June 2021
© The Author(s) 2021

Abstract

The paper introduces a modified version of a genetic algorithm with aggressive mutation (GAAM) called fGAAM (fast GAAM) that significantly decreases the time needed to find feature subsets of a satisfactory classification accuracy. To demonstrate the time gains provided by fGAAM both algorithms were tested on eight datasets containing different number of features, classes, and examples. The fGAAM was also compared with four reference methods: the Holland GA with and without penalty term, Culling GA, and NSGA II. Results: (i) The fGAAM processing time was about 35% shorter than that of the original GAAM. (ii) The fGAAM was also 20 times quicker than two Holland GAs and 50 times quicker than NSGA II. (iii) For datasets of different number of features, classes, and examples, another number of individuals, stored for further processing, provided the highest acceleration. On average, the best results were obtained when individuals from the last 10 populations were stored (time acceleration: 36.39%) or when the number of individuals to be stored was calculated by the algorithm itself (time acceleration: 35.74%). (iv) The fGAAM was able to process all datasets used in the study, even those that, because of their high number of features, could not be processed by the two Holland GAs and NSGA II.

Keywords Genetic algorithms · Feature selection · Mutation · Wrappers

Mathematics Subject Classification 68W50

1 Introduction

A genetic algorithm (GA) is a heuristic optimization tool used for a variety of applications. One area where it is frequently applied is feature selection [1–3]. A genetic algorithm used as a feature selection method usually behaves as a wrapper [4, 5], which means that it wraps the selection process around the classification scheme. To be more exact, it uses a classifier to evaluate the quality of each candidate feature subset created during the exploration of the space of possible solutions. The main advantage of GAs used for feature selection is that during the exploration process, they evaluate a set of solutions in parallel rather than evaluating

solutions one by one [3]. Moreover, they are not prone to getting stuck at local minima, and they do not need to make assumptions about the interactions between features [6–8]. For these reasons, they are often used to find feature sets with high discrimination capabilities in multidimensional and nonlinear problems.

Although a few different genetic approaches have been proposed to solve the feature selection task, usually the simplest approach, based on the classic Holland GA with a special coding scheme, is used in practice [1, 3, 6, 8, 9]. With this approach, the GA's individuals encode the existence of individual features in a given feature subset. That means that i) the number of genes in an individual is always equal to the number of features in the feature space, and ii) each gene can take the value 1 or the value 0, meaning that the feature exists (1) or does not exist (0) in the feature subset encoded in this individual. A classification accuracy is often used to evaluate the quality of individuals in the Holland GA [4]. This approach is easy to implement but has one weakness: because it uses pure classification accuracy as a fitness function, the algorithm tries to find individuals providing 100%

✉ Izabela Rejer
Izabela.Rejer@zut.edu.pl
Jarosław Jankowski
Jaroslaw.Jankowski@zut.edu.pl

¹ West Pomeranian University of Technology in Szczecin,
Faculty of Computer Science and Information Technology,
Żołnierska 49, 71-210 Szczecin, Poland

classification accuracy, regardless of their number of features. Since the majority of classifiers take new parameters with each additional input dimension and as a result usually obtain higher accuracy than classifiers built over a smaller input space [10], individuals coding more features provide more flexibility and hence are favored in the optimization process. As a result, the GA tends to look for individuals coding large feature sets. Such a situation is undesirable for two reasons: i) classifiers of higher input dimension usually have lower generalization capabilities; ii) more inputs elongates the training time. One solution to deal with this problem is to modify the fitness function of the Holland GA by adding a penalty term penalizing the individuals coding too many features [11]. In theory, this solution fulfills the goal, but in practice, the process of reducing the number of features encoded in the GAs' individuals is incredibly slow.

Other genetic approaches that are often used for feature selection are NSGA (nondominated sorting genetic algorithm) and its later modifications: NSGA II [12–16] and NSGA III [17, 18]. NSGA is equipped with binary individuals (the same as in the classic Holland algorithm) and with a specific fitness function based on the domination principle. It applies a two-step evaluation process on individuals from the current population. In the first step, the whole population (composed of the mother individuals and their offspring) is sorted based on the non-domination of individuals. In the second step, each individual is assigned a fitness rank equal to its non-domination level. The best (from the multi-criteria point of view) individuals are assigned rank 1, the next best are assigned rank 2, etc. Usually two competing objectives are used here [16]: the minimization of the number of features and the maximization of the class separation measure (such as classification accuracy). Since NSGA, like the classic Holland GA with penalty term, i) starts from a random population composed of individuals coding approximately 50% of possible features and ii) tries to compromise between these two competing objectives, it also suffers from a long processing time. Although in NSGA II some mechanisms for accelerating the algorithm were introduced (like the crowding distance concept), the algorithm still needs a long time to find a satisfactory solution composed of a small number of features.

In both of the approaches discussed so far, an individual contains one gene for each feature. Hence, when the task at hand is described by a feature set composed of hundreds or thousands of features, the classifiers trained at early iterations take many unnecessary inputs, which prolongs their training time and deteriorates their generalization capabilities. Meanwhile, in many tasks, the ultimate goal of the feature selection is not to discard some unnecessary features out of thousands of features but to choose a few features of the highest discrimination capabilities. For such tasks, the better approach is to start the selection process not from

approximately half of all the possible features (like in the previously discussed GAs) but from the number of features that is appropriate for the given task (taking into account the number of observations, the rate of nonlinearity, and other conditions). Such an approach might save a lot of processing time.

One GA applying such an approach is the Culling GA, proposed by Baum et al. [19]. The distinct feature of this algorithm is its selection strategy. While in the Holland GA, a new population is created by choosing individuals of high fitness value, in the Culling GA, a new population is created by discarding the worst individuals. Yom-Tov and Inbar have adopted this general strategy for a feature selection task [20]. Their algorithm starts with the random partitioning of a set of P features into M subsets (GA's individuals) of N feature indexes (genes), where $N = P/M$. Next, individuals are ordered with respect to their classification accuracy, and the predetermined percentage of individuals of the lowest accuracy is discarded. Finally, the new population of M individuals of N genes each is created by selecting features (random selection with repetitions) from the remaining feature subsets. This approach assumes that the sets containing features of the highest discrimination capabilities are more likely to survive the selection process, and hence these features will be returned by the algorithm. Although this assumption is true in regard to the whole feature subsets, it is not true for the individual features. As we showed in [21], due to discarding entire individuals during successive iterations, features significant for the classification process might be discarded at the initial stages of the algorithm if they are incorporated into a set of insignificant features. As a result, the classification accuracy for the feature sets returned by the Culling algorithm is usually significantly lower than for other GAs.

In 2013, we proposed another genetic algorithm, called GAAM (genetic algorithm with aggressive mutation), that is capable of leading the search process with feature sets containing a given number of features [22]. In [23], we showed that in a selection problem where 6 out of 324 features had to be selected our strategy outperformed three other popular feature selection methods in terms of classification accuracy (GAAM—94.29%, forward selection—91.43%, Lasso—90.71%, ReliefF—85.71%). We also showed (in [21]) that it provided more accurate results and fewer features in the final feature sets than the two genetic search strategies (classic Holland encoding the existence of features and Culling). In [11, 24], we showed the same result in regard to classic Holland with penalty term, and NSGA, respectively. Results of some other experiments with GAAM can be found in [25–29]. The good performance of GAAM can be attributed to the specific mutation scheme that enables a multidirectional search in the local surroundings of the best feature subsets found in each algorithm iteration. Moreover, due to

leading a search process among the subsets of a fixed size, the search space is significantly reduced, allowing for a more exhaustive search.

The GAAM method provides feature sets of high discrimination capabilities; however, its original version is computationally demanding. The aim of this paper is to propose a modified version of GAAM, called fGAAM (fast GAAM), that (i) reduces the GAAM computational time and (ii) makes the algorithm resizable—it is independent of the number of features encoded in the individuals. To reach this goal, we changed the evaluation process of individuals produced in successive generations and introduced two additional subtle modifications (in the mutation scheme and selection process).

The main problem with GAAM is the same as with other wrappers - to assess the quality of the feature set encoded in an individual, the classifier has to be built and trained. That makes the process of searching the solution space quite long. The advantage of GAAM over other GAs is that the classifiers are trained with the smallest required number of features, which makes the process quicker than in the case of other genetic approaches such as Holland or NSGA. However, the process is still incredibly slow, and any acceleration is most welcome. As we will show in the paper, the fGAAM algorithm accelerates the computational time of the original GAAM without deteriorating its precision.

The main component of the computational cost of wrapper algorithms for feature selection is the cost of training the classifier for each candidate feature subset. The cost of training a single classifier differs significantly depending on the classifier model, its parameters, and its number of inputs. Although the cost might sometimes be quite high, usually it is possible to tackle when only one classifier has to be built. The problem is that with genetic algorithms, hundreds of classifiers or more are built in each of hundreds of generations. The classifier training costs add up, slowing the algorithm significantly. One possible solution to mitigate this drawback is to decrease the number of classifiers trained in each generation. Since the GA reproduction process is unsupervised and hence the same individuals might reappear in successive generations, we can obtain such an effect by storing the fitness values of the individuals from the last few populations and using a stored fitness value every time a stored individual is found in the following populations. Although this modification will not affect the algorithm speed in the first few populations, where most new individuals differ from the previous ones, it should significantly increase it at later stages.

Assuming that this hypothesis is correct, the question that arises is how many individuals should be stored to obtain a significant increase in the algorithm's speed. A first impression is that the more individuals the algorithm stores, the higher the acceleration of the algorithm will be. As we will

show in this paper, this impression is true but only until some critical point. If the number of stored individuals exceeds this point, the gains stemming from storing the additional individuals gradually decrease, and finally, the costs of storing the additional individuals exceed the benefits of training a smaller number of classifiers. In fact, the function describing the algorithm's computational cost in terms of the number of stored individuals has a U-shape with the right arm going to infinity. At first, the computational cost drops as the number of individuals increases, and after getting past the saturation point, it increases linearly. This means that the number of individuals stored for later populations cannot be arbitrarily chosen but should be carefully selected for the task at hand, taking into account the training cost of a single classifier and the cost of comparing individuals from the current population with those stored previously. Since both factors are not known in advance but are specific for the task, we decided to leave the problem of calculating the optimal number of individuals to store for the algorithm itself.

There is one more GAAM feature that should be corrected in order to extend the domain of its possible applications. The main idea of the algorithm that distinguishes it from other GAs for feature selection is that it looks for a feature set composed of a strictly defined number of features. Due to this feature, the search space is significantly reduced, allowing for a more exhaustive search. In a majority of real tasks, we look for feature sets composed of only a few, or at the most tens of, features. Small feature sets are beneficial because i) the classifier can be trained more quickly and easily, ii) the probability of overfitting is much smaller, and iii) the interpretability of the classifier structure is significantly higher. Although having a small feature set is usually beneficial, for complex problems, we often have to use high-dimensional feature sets simply because a few features is not enough to obtain a model of satisfying generalization capabilities (underfitting occurs). For such problems, we might be interested in finding feature sets composed of hundreds or thousands of features. Using the current version of GAAM in such case might be not a right option. The problem is that the computational complexity of GAAM increases linearly with each additional feature introduced to the feature set. Usually, a linear complexity is not a problem, but in the case of GAAM, each additional feature means that we have to train additional classifiers per individual in each population. Hence, assuming that one population with 10 individuals of 10 features is processed in 1 s, when we increase the number of features to 1000, the same population will be processed in 100 s. Keeping in mind that during each GA hundreds of populations are processed, the total increase of the computational time might not be acceptable.

To solve the problem described above, the number of offsprings created in the mutation process should be limited. The simplest but efficient way to deal with this task is to

introduce a new algorithm parameter, commonly used in the Holland algorithm, called the probability of mutation. As we will show later in this paper, by using this parameter, we can achieve an acceptable algorithm speed even for large feature sets, and in this way, we can make the GAAM resizable. However, we would like to underline here that by limiting the number of mutated individuals, we also weaken the main advantage of the GAAM, which stems from the aggressive mutation—a multidirectional search in the local surroundings of the best feature subsets found in each algorithm generation [30]. That is why, although we added the mutation probability parameter to the algorithm, we think that in most real problems it is better to limit the number of features to be found, rather than limit the number of mutated individuals.

The final modification introduced in the fGAAM is very subtle. We simply changed the method of selecting individuals to the mother population from tournament to rank selection. We decided on this modification because the overall scheme of the GAAM, where first the mother population is significantly extended as a result of aggressive mutation and then is reduced in the selection process, is quite competitive by itself and does not need additional enhancement. Hence, the tournament selection that we originally used in the selection process does not provide any benefits. It only significantly increases the computation time.

The paper is structured as follows. In the next section, we describe shortly the original version of the GAAM and then provide a detailed description of all the modifications that lead us to the fGAAM. Section 3 introduces eight datasets that we used to compare the fGAAM with the original GAAM and also with the four other GAs used for feature selection: classic Holland, Holland with penalty term, Culling, and NSGA II. The next section presents and discusses results obtained during the experiments. Finally, the Conclusions section closes the paper.

2 Methods

2.1 Original GAAM

The pseudocode for the original GAAM algorithm is given in Algorithm 1. The algorithm starts by setting four parameters: N , M , T , $tourN$. The first parameter N corresponds to the number of features that are searched by the algorithm. Since each feature is encoded by one gene, the same parameter also corresponds to the number of genes contained in an individual. The next two parameters, M and T , correspond to the number of individuals and number of generations, respectively. The last parameter $tourN$ encodes how many individuals will take part in each tournament in the selection step.

After setting the algorithm parameters, the initial mother population is drawn. The population is composed of M randomly chosen individuals, where each individual consists of N genes. A single gene of an individual corresponds to one feature from the set of possible features. The encoding process is straightforward—a gene takes an integer value equal to the feature index.

The main algorithm loop starts with two reproduction operations performed on the individuals from the mother population (*motherP*). The first operation is a classic one-point crossover, well known from the classic Holland GA, performed with a fixed probability equal to 1. The offsprings born during the crossover operation create a new population called *crossedP*. The second operation is an aggressive mutation—a concept specific to the GAAM. During aggressive mutation, each gene in each individual is mutated individually by setting its value to the index of one randomly chosen feature from the feature set. As a result of the mutation operation, one parent individual has a set of N offsprings, each created by mutating another gene of that individual. The pseudocode of the mutation scheme is presented in Algorithm 2 [23].

Algorithm 1: The GAAM's pseudocode; N – number of genes, M – number of individuals, T – number of generations, $tourN$ – number of individuals in tournament

```

1 Set  $N$ ,  $M$ ,  $T$ ,  $tourN$ 
2  $motherP$  = DrawInitialPopulation( $M$ )
3 for  $generation = 1$  to  $T$  do
4    $crossedP$  = OnePointCrossover( $motherP$ )
5    $mutatedP$  = AggressiveMutation( $motherP$ )
6    $currentP$  = [ $motherP$   $mutatedP$   $crossedP$ ]
7    $fitness$  = FitnessEvaluation( $currentP$ )
8   [ $motherP$ ,  $fitness$ ] = TournamentSelection( $currentP$ ,  $fitness$ ,  $tourN$ )
9  $theBestIndividual$  = MaxFitness( $motherP$ ,  $fitness$ )

```

Next, the *motherP* is concatenated with the *mutatedP* and the *crossedP* to create the *currentP*, composed of *M* mother individuals, *M* (or *M-1* for odd *M*) off-springs from the *crossedP*, and *N*M* off-springs from the *mutatedP*. At the next step, all individuals from the *motherP* are evaluated with a chosen classifier model. The evaluation process is similar to evaluation processes in other wrappers. The feature subset encoded in an individual is introduced as

the input to the classifier that is then trained on the available dataset. The classifier's accuracy is assigned to the individual as its fitness value. Since the algorithm permits duplicated features (that might appear as a result of genetic operations or random initialization of the initial population), before the classifier training all identical genes (apart from one) are set to zero.

Algorithm 2: The pseudocode for the aggressive mutation function;
P – the total number of features in the feature space

```

1  j = 0
2  for i = 1 to M do
3      newIndividual = motherP(i, 1:N)
4      for g = 1 to N do
5          newIndividual(g) = Random({0, 1, ..., P})
6          j++
7      mutatedP(j) = newIndividual

```

Finally, in the last step of the main GAAM loop, *M* individuals are selected for the next mother population with a tournament selection method. After repeating the main loop *T* times, the individual of the highest fitness value (highest accuracy) from the last *motherP* is returned as the GAAM output.

2.2 fGAAM

To reduce the number of classifiers trained in each generation of the GAAM, we modified the main algorithm loop and the function used for evaluating the individuals and also introduced a new function *HowManyToStore()*. The fGAAM pseudocode is given in Algorithm 3. As the figure shows, the first generation of the fGAAM, where the number of stored individuals (*storedN* parameter) is determined, is slightly different from the rest.

Algorithm 3: The fGAAM's pseudocode; *N* – number of genes, *M* – number of individuals, *T* – number of generations, *probM* – probability of mutation

```

1  Set N, M, T, probM
2  motherP = DrawInitialPopulation(M)
3  for generation = 1 to T do
4      crossedP = OnePointCrossover(motherP)
5      mutatedP = AggressiveMutation(motherP)
6      currentP = [motherP mutatedP crossedP]
7      if generation = 1 then
8          storedN = HowManyToStore()
9      else
10         currentP = Sort(currentP by genes)
11         currentP = Unique(currentP)
12         fitness = fGAAMFitnessEvaluation(currentP, storedN)
13     [motherP, fitness] = RankSelection(currentP, fitness)
14  theBestIndividual = MaxFitness(motherP, fitness)

```

The first modification introduced in the fGAAM is to leave only individuals with unique phenotypes in the population. This requires a two-step procedure. First, the genes' order is unified by sorting individuals' genes (the operation: $\text{Sort}(\text{currentP}$ by genes)). Next, a unique function is used on the current population to detect and discard individuals with repeating phenotypes. As a result of this modification, only individuals of unique phenotypes remain in the population. It should be emphasized here that the ability to sort genes in individuals, which allows us to detect repeating phenotypes in a very simple way, is a distinct feature of the fGAAM (or GAAM). Such a procedure cannot be applied in the genetic approaches using binary individuals coding existence of features, where the features that should be introduced to the classifier are given directly as gene indexes.

The second part of the algorithm that was changed was the function used for the evaluation of individuals. We changed it by introducing the concept of storing individuals (and their fitness values) born in the last few populations. The pseudocode for a modified version of the evaluation function ($\text{fGAAMFitnessEvaluation}()$) is given in Algorithm 4. To perform this function, a value for the parameter containing information about the upper limit of stored individuals (storedN) has to be provided. This value is

established by the algorithm according to the procedure that will be described later in this section.

The $\text{fGAAMFitnessEvaluation}()$ starts by comparing all the individuals from the current population (currentP) with the individuals stored in a set storedInd . For the first algorithm generation, the set storedInd is empty, and then, with each new generation it is gradually extended (up to a limit defined by the storedN parameter) by taking new individuals from the mother populations. The subfunction $\text{CompareGenotypes}()$ is performed on the currentP and returns two sets of individuals: evaluatedInd and newInd . While the set evaluatedInd contains individuals evaluated and stored in previous fGAAM generations, the set newInd is composed of individuals that have not been evaluated yet. In the next step, the individuals from the newInd set are evaluated. The evaluation process is the same as in the original GAAM—a chosen classifier model is trained for each individual. After evaluation, the genotypes and fitness values of the individuals from the newInd set are added at the beginning of the set storedInd . If, after this operation, the number of individuals in the storedInd set exceeds the value of the storedN parameter, the redundant individuals from the end of the set are discarded.

Algorithm 4: The pseudocode for the function $\text{fGAAMFitnessEvaluation}()$; storedN – the upper limit of individuals to be stored in the storedInd set

```

1  $[\text{newInd}, \text{evaluatedInd}, \text{storedInd}] = \text{CompareGenotypes}(\text{currentP}, \text{storedInd})$ 
2 for  $i = 1$  to  $\text{size}(\text{newInd})$  do
3    $\text{fitness}(i) = \text{TrainClassifier}(\text{newInd}(i))$ 
4  $\text{storedInd} = \text{AddIndividuals}(\text{newInd}, \text{storedInd})$ 
5  $\text{storedInd} = \text{storedInd}(1:\text{storedN})$ 

```

The upper limit of individuals that are stored in the storedInd set is determined by the function $\text{HowManyToStore}()$, presented in Algorithm 5. This function takes in all individuals created during the reproduction step and uses them to evaluate the computational cost of two operations: i) an average classifier's training cost of an individual (TC) and ii) an average cost of comparing a single individual with all individuals from the population (CC). To calculate TC , all individuals from the currentP are evaluated with the same classifier model as used for other generations, and the total training cost is divided by the number of evaluated individuals. The average value for CC is determined by the function $\text{CompareIndividual}()$, which compares the last individual from the current population with all individuals from the same population. Since the cost of a single comparison is very small (on the order of 10^{-4}), to ensure a stable result,

the comparison is performed inside a loop with 1000 iterations, and then the output is averaged. Next, the value of the storedN parameter is calculated. We assumed here that the number of individuals stored should be such that the formula $CC * \text{storedN} = 0.01 * TC$ will be satisfied. Hence, the storedN parameter is calculated as:

$$\text{storedN} = 0.01 * (TC/CC) \quad (1)$$

With this equation, we ensure that the comparison process can take at the most 1% of classifier training time. Finally, the same operations that are used on the remaining generations are performed: i) the unique individuals from the current population are chosen, and ii) they are stored in the storedInd set. Additionally, before the individuals are stored in the storedInd set the vector fitness is adjusted to contain only fitness values of unique individuals.

Table 1 The demography of the datasets used in the survey

Dataset	No. of features	No. of classes	No. of examples	Source
Postoperative	8	3	90/ 86	https://archive.ics.uci.edu [31]
Adult	14	2	48842/ 45222	http://archive.ics.uci.edu [31]
Dermatology	34	6	366/ 358	https://archive.ics.uci.edu [31]
Humanactivity	60/ 57	5	24075	file available in Matlab R2019a [32]
SMK-CAN-187	19993/ 19983	2	187	http://featureselection.asu.edu [33, 34]
Gisette	5000/ 4891	2	7000	https://archive.ics.uci.edu [31, 35]
Yale_64x64	4096/ 3304	15	165	http://www.cad.zju.edu.cn [36]
Coil100	1024	100	7200	http://www.cad.zju.edu.cn [37, 38]

Values written with a bold font refer to the number of features and examples after the preprocessing stage

Table 2 The description of the datasets

Dataset	Description
Postoperative	This dataset contains 90 cases of postoperative patients. Each case belongs to one of three classes concerning where the patient should be sent from the recovery area: class 0 – to the intensive care unit; class 1 – home; class 2 – to the general hospital floor. Eight attributes corresponding mainly to body temperature measurements are used to describe the cases (1 – internal temperature, 2 – surface temperature, 3 – oxygen saturation, 4 – blood pressure, 5 – stability of surface temperature, 6 – stability of core temperature, 7 – stability of blood pressure, 8 – perceived comfort)
Adult	This dataset contains 48842 records of adults concerning whether a given person makes more than 50K a year (class 1) or not (class 0). Each record is described by 14 attributes (mostly nominal), carrying information about the person's age, work class, education degree, years of education, marital status, occupation, relationship, race, sex, capital gain, etc
Dermatology	This dataset contains 366 cases of Eryhemato-Squamous Disease, classified into 6 types: psoriasis (class 1), seboreic dermatitis (class 2), lichen planus (class 3), pityriasis rosea (class 4), chronic dermatitis (class 5), and pityriasis rubra pilaris (class 6). Each case is described by 12 clinical and 22 histopathological attributes
Humanactivity	This dataset contains 24075 observations of five different physical human activities: 1 – sitting, 2 – standing, 3 – walking, 4 – running, 5 – dancing. Each observation is described by 60 features extracted from acceleration data measured by smartphone accelerometer sensors. The dataset covers acceleration, GPS, gyroscope, light, magnetic field, and sound level data
SMK-CAN-187	This dataset compares the gene expression data (199993 features) recorded from two classes of smokers: 1 – smokers with lung cancer, 2 – smokers without lung cancer
Gisette	This dataset contains 7000 images of the handwritten digits '4' and '9' (6000 in the training set and 1000 in the validation set) described by 5000 features. The feature set is composed of i) 2500 true features, it is pixels sampled randomly from the middle-top part of a fixed-size image of dimension 28x28 and higher order features created as products of these pixels and ii) 2500 distractor features having no predictive power
Yale_64x64	This dataset contains 165 grayscale face images in GIF format of 15 individuals (15 classes). There are 11 images per subject, one per different facial expression or configuration: center light, wearing glasses, happy, left light, wearing no glasses, normal, right light, sad, sleepy, surprised, and wink. Each feature represents the gray level of one pixel from the 64x64 image
Coil100	This dataset contains images of 100 different objects (one object corresponds to one class). The images of each object were taken 5 degrees apart as the object was rotated on a turntable (72 images were taken for each object). The size of each image is 32x32 pixels (with 256 gray levels per pixel); hence, each image is represented by a 1024-dimensional vector

Algorithm 5: The pseudocode for the function `HowManyToStore()`;
 TC – the average training cost, CC – the average comparison cost,
 $storedN$ – the upper limit of individuals to be stored in the $storedInd$ set

```

1  $[TC, fitness] = \text{TrainClassifier}(currentP)$ 
2  $CC = \text{CompareIndividuals}(currentP)$ 
3  $storedN = 0.01 * (TC/CC)$ 
4
5  $currentP = \text{Sort}(currentP \text{ by } genes)$ 
6  $[currentP, indexes] = \text{Unique}(currentP)$ 
7  $fitness = fitness(indexes)$ 
8  $storedInd = currentP$ 

```


The two other changes made in the algorithm structure are rather minor, but they enhance the algorithm's capabilities. The first change, replacing the tournament selection with a rank selection, significantly accelerates the algorithm (fewer generations are needed to find a feature subset of the same accuracy). The second change, introducing the mutation probability, enhances the algorithm's scalability and enables us to use it efficiently even for problems described in a high-dimensional feature space.

3 Experiment setup

The study implemented to test fGAAM performance was divided into four stages. The objective of the first stage was to show that regardless of the characteristics of the data set, some individuals of the same phenotypes might appear in successive algorithm generations. The experiments performed during this stage also demonstrate the time benefits stemming from evaluating only individuals with unique phenotypes. The experiments performed during the second stage were designed to show that the time needed to complete the algorithm could be further reduced by storing the evaluated individuals from the last few populations. Here, we also show that if we store individuals from too many generations, the processing time increases linearly. The next stage was dedicated to verifying our approach to find the number of individuals that should be stored in the individuals set. Finally, in the last stage, we compared the fGAAM, in terms of accuracy and processing time, with the four other GAs used for feature selection: classic Holland, Holland with penalty term, Culling, and NSGA II.

In all stages, eight datasets with different characteristics were used: *Postoperative*, *Adult*, *Dermatology*, *Humanactivity*, *SMK-CAN-187*, *Yale_64x64*, *Gisette*, and *Coil100*. These datasets differ in three dimensions: number of features, number of classes, and number of examples. The detailed demography of the chosen group of datasets is presented in Table 1, and their short descriptions are given in Table 2. Before we started the experiment, the individual datasets were preprocessed according to the following rules: a) *Postoperative*: 4 cases (3 cases containing *NaN* data and a single case of class '0') were removed, leaving 86 records and 2 classes. b) *Adult*: 3620 cases containing *NaN* data were removed, leaving 45222 records. c) *Dermatology*: 8 cases containing *NaN* data were removed, leaving 358 records. d) *Gisette*: 29 features that had the same value for each record were removed, leaving 4971 features. e) For each dataset, the pairs of features with a linear correlation exceeding 99% were identified, and one feature from each pair was removed from the feature set. This preprocessing rule influenced four datasets: *Humanactivity* (3 features were removed, leaving 57 features); *SMK-CAN-187* (10 features were removed,

leaving 19983 features); *Gisette* dataset (80 features were removed, leaving 4891 features); and *Yale_64x64* (856 features were removed, leaving 3304 features).

In all four stages, the two fGAAM/GAAM parameters, the number of generations (T) and the number of individuals in a mother population (M), were set at the same levels: $T = 100$ and $M = 10$. Also, the probability of mutation ($probM$) was kept the same (equal to one) for all but one dataset. For *Coil100*, we had to decrease it because as a result of a high number of classes (100 classes), the evaluation process of a single individual lasted about 5 s (compared to some milliseconds in other cases), and the time needed to evaluate all individuals in 100 generations exceeded 102 hours. Therefore, we decided to limit the number of individuals born in the mutation process to 10%. As a result, the number of individuals evaluated in each generation decreased from 740 to about 90, and the time needed to complete the algorithm decreased from 102 to 12.5 hours.

The last algorithm parameter, the number of genes in an individual (N), was determined individually for each dataset. For the five datasets with relatively small number of examples (compared to the number of classes), we tried to apply the rule provided by Raudys and Jain, which says that at least 10 times more observations than features per class should be used in the classification task to limit the overfitting phenomena [39]. For three datasets (*Postoperative*, *Dermatology*, and *SMK-CAN-187*), we managed to apply this rule in its original meaning by setting the N parameter to 5 for *Postoperative*, 6 for *Dermatology*, and 9 for *SMK-CAN-187*. However, for the two remaining datasets (*Yale_64x64* and *Coil100*), we had to violate the rule significantly because it created too sparse feature set to obtain reasonable classification results. Hence, for *Yale_64x64*, we set N to 10, and for *Coil100*, we set N to 72, which corresponded to one example per class and feature.

Although the Raudys and Jain rule provided much more freedom for the N value for the three remaining datasets (*Adult*, *Humanactivity*, and *Gisette*), there were other reasons for keeping it low. In the case of the *Adult* dataset, we limited the number of genes to 10 because of a small number of potential features describing the problem (only 14 features). For the *Humanactivity* dataset, we first tried to use 50 genes; however, the algorithm stopped after a few generations, returning the feature set providing classification accuracy of 100%. We tried to run the algorithm a few times with a smaller number of genes, and finally for $N = 10$ the algorithm completed 100 epochs and then returned a feature set providing a somewhat smaller accuracy of 97%. The same situation repeated for the last dataset, *Gisette*—however, this time we used slightly more genes (20) and obtained an accuracy of 94%.

The individuals' evaluation process was done with an LDA (linear discriminant analysis) classifier. This classifier

Fig. 1 The comparison of GAAM and fGAAM (with $storedN = 0$) in terms of time needed to complete each generation; **a** *Postoperative*, **b** *Adult*, **c** *Dermatology*, **d** *Humanactivity*, **e** *SMK-CAN-187*, **f** *Gisette*, **g** *Yale_74x74*, **h** *Coil100*

Table 3 The comparison of results obtained with GAAM and fGAAM (with *storedN* = 0); the average time needed to complete one generation (columns 4 and 5), the time saved (column 6), the average number of individuals excluded from the evaluation process in fGAAM (column 7), and the classification accuracy obtained with a feature set returned by both algorithms (columns 2 and 3)

Dataset	Accuracy [%]		Average time [s]		Gains [%]	
	GAAM	fGAAM	GAAM	fGAAM	Time	Individuals
Postoperative	73.18	73.39	1.03	0.63	38.65	41.08
Adult	79.54	79.54	68.47	55.05	19.59	19.77
Dermatology	95.00	94.88	1.69	1.48	12.11	11.71
Humanactivity	96.92	97.01	62.76	59.22	5.64	5.83
SMK-CAN-187	84.73	82.55	2.02	1.90	5.98	3.68
Gisette	94.33	94.25	32.94	32.27	2.02	2.04
Yale_64x64	91.80	92.17	2.73	2.62	4.11	3.00
Coil100	90.17	89.97	808.92	760.38	6.00	1.68
Mean					11.76	11.10

uses two simple rules to discriminate classes: maximization of between-class variance and minimization of within-class variance. To carry out the classification process a MATLAB function *classify* with a linear discriminant function was applied. The function performs a multi-class classification by fitting a multivariate normal density function to each class. In the training process, the 10 cross-fold validation scheme was used. The classifier's accuracy, reported in the Results section, was calculated as the mean accuracy from 10 validation sets. In all experiments, the same stop conditions were used: completing T generations or localizing the feature set providing an accuracy of more than 99%. All experiments were performed on a machine with the following parameters: Processor – Intel CORE i7-6700HQ CPU @ 2.60 GHz, 16GB RAM, Windows 10 x64.

4 Results and discussion

4.1 Stage 1

To estimate the influence of our first modification, evaluating individuals with unique phenotypes, we ran both algorithms (GAAM and fGAAM) on the eight datasets described in Tables 1 and 2. To unify the GAAM with the fGAAM, we changed the selection procedure in the GAAM from tournament to rank selection, and to focus only on the time benefits gained by evaluating individuals with unique phenotypes, we temporarily disabled the function *HowManyToStore()* in the fGAAM and set the *storedN* parameter to 0. The comparison of both algorithms in terms of time needed to perform all necessary computations for each generation is presented in Fig. 1, and the averaged results are presented in Table 3.

As shown in Table 3, individuals of the same phenotypes appeared for all analyzed datasets. Of course, the number of redundant individuals was different for datasets with different number of features and genes—in general, the more features in the feature set and genes in an individual, the less individuals of the same phenotypes. However, even for

SMK-CAN-187, which contains thousands of features, the redundant individuals appeared. The average amount of time saved as a result of detecting and excluding redundant individuals from the evaluation process was almost 12%.

In Table 3, we also present the average classification accuracy achieved by the classifiers using feature sets returned by the algorithms. Since both algorithms share the same evaluation, reproduction, and selection methods, the classification accuracies presented in the second and third columns of Table 3 are also very similar for each dataset.

4.2 Stage 2

To show that the time needed to complete the algorithm can be further reduced by storing the individuals evaluated in the last few populations, we ran the fGAAM on each dataset with different *storedN* values. Because different numbers of genes were used for different datasets, the population size (after the reproduction) also differed significantly among the datasets, from 70 individuals for *Postoperative* and *Yale_64x64* to 220 individuals for *Gisette*. Hence, instead of setting the same *storedN* level for each dataset, we fixed the number of populations to store. With this setting, the *storedN* parameter was different for each dataset, but the number of stored populations (*storedP*) was the same.

The algorithm was run four times for each dataset. In each run, a different level of the *storedP* parameter was used: 1, 3, 10, and 100 (*storedP* equal to 100 meant that all new individuals from all populations were stored). Such a choice of parameters allowed us to compare the time differences in the cases of a small vs a high number of stored individuals. Fig. 2 presents the comparison of the times needed to process each generation of the original GAAM and fGAAM ran for the two most extreme cases (*storedP* = 1 and *storedP* = 100). The results obtained in Stage 1 for the original GAAM were treated as a baseline for calculating the time differences for different *storedP* levels, shown in Table 4.

As shown in Table 4, for five out of eight datasets, each value of *storedP* greater than zero evoked an improvement

Fig. 2 Time needed to complete each generation of the original GAAM and fGAAM with the two most extreme *storedP* levels (1 and 100); **a** *Postoperative*, **b** *Adult*, **c** *Dermatology*, **d** *Humanactivity*, **e** *SMK-CAN-187*, **f** *Gisette*, **g** *Yale_74x74*, **h** *Coil100*

Table 4 The average time (in seconds) needed to complete one generation of fGAAM with four different *storedP* levels (1, 3, 10, and 100) and the maximum number of individuals stored for *storedP* = 100. The time gain was calculated in regard to the average time needed to complete one generation of the original GAAM (reported in Table 3)

Dataset	Mean time [s] for <i>storedP</i> equal				Time gain [%] for <i>storedP</i> equal			
	1	3	10	100	1	3	10	100
Postoperative	0.15	0.04	0.04	0.04	85.10	95.55	96.12	96.12
Adult	43.47	31.98	8.82	8.24	36.50	53.29	87.11	87.96
Dermatology	1.31	1.13	0.74	0.60	22.07	32.79	55.80	64.37
Humanactivity	55.21	51.77	43.60	35.14	12.03	17.51	30.53	44.01
SMK-CAN-187	1.87	1.86	2.03	2.54	7.40	7.82	-0.60	-25.52
Gisette	31.25	30.67	31.21	34.22	5.13	6.90	5.26	-3.86
Yale_64x64	2.49	2.46	2.59	3.20	8.53	9.71	4.92	-17.27
Coil100	775.37	711.40	712.23	689.18	4.14	12.05	11.95	14.80
Mean					22.61	29.45	36.39	32.58

in the fGAAM speed. For two datasets (*Gisette* and *Yale_64x64*), the improvement was observed for *storedP* equal to 1, 3, or 10 but not for *storedP* equal to 100. And for the last dataset (*SMK-CAN-187*), only the two smallest levels of *storedP* induced an improvement in the algorithm's speed. From the table, it is clear that the extent of the improvement is correlated with the number of features in the datasets. The highest improvement was observed for the first four datasets (maximum 57 features); the improvement was much smaller for the remaining datasets containing many more features. This behavior is quite easy to explain – the higher the number of features, the higher the diversity of the population.

The plots presented in Fig. 2 confirm our hypothesis that the optimal *storedN* level depends on the characteristics of an analyzed dataset. At the beginning of our study, we also assumed that for each dataset the dependency between the number of stored individuals and the processing time would have a U-shape. That is, we expected that for *storedP* equal to 100 we should observe the moment when the time curve reversed its direction and started to grow. This hypothesis, however, turned out to be true only for some datasets – those composed of a sufficiently high number of features. For the datasets with relatively small number of features (maximum 57 in our survey), the algorithm was fastest when all new individuals from all generations were stored in the individuals set.

4.3 Stage 3

The goal of the third stage of our study was to show that the proposed approach to calculate the *storedN* level increased the fGAAM speed regardless of the characteristics of the analyzed domain, such as the size of the feature space, the number of classes, and the number of observations. To achieve this goal, we ran the fGAAM with the function for calculating the *storedN* level (*HowManyToStore*) for each of the eight datasets and compared the algorithm speed to the speed obtained for the original GAAM. The results are presented in Fig. 3 and in Table 5.

As shown in Table 5, by using the function that calculates the number of individuals that should be stored in the individuals set, we obtained an increase in the algorithm speed for each dataset. The smallest acceleration was obtained for *Gisette* (3.29%) and the highest for postoperative (95.74%). However, the most meaningful time savings (3.5 hours) were obtained for **Coil100**. The time acceleration averaged across all eight datasets was equal to 35.74%.

To compare the results from all three stages of the study, we prepared a table presenting the time gains obtained with different numbers of individuals stored in the individuals set (Table 6). The first group of columns in Table 6 (columns 2-7) presents the time gains obtained for different *storedP* levels (*storedP* = 0, 1, 3, 10, 100) and also for the dataset-specific *storedN* value (calculated in Stage 3) with regard to the original GAAM. The second part of the table (columns 8-12) presents the same data but after removing the time savings obtained as a result of our first modification (evaluating individuals of unique phenotypes). Hence, the second part of Table 6 shows the acceleration induced exclusively by storing some amount of previously evaluated individuals for further generations.

As can be noticed in Table 6, on average each analyzed level of *storedP*, as well as the individual level of *storedN* calculated by the algorithm itself, provided an additional acceleration of the fGAAM processing time. The smallest acceleration of about 11% was obtained when only individuals from one population were stored in the individuals set (*storedP* = 1); the highest acceleration (of about 25%) was obtained when 10 populations of individuals were stored. It should be also emphasized here that when the number of individuals to store was calculated by the algorithm itself, the acceleration was only 1% smaller (about 24%) than this highest value.

From Table 6, we can also see that for different datasets, a different number of stored populations provided the highest acceleration. For the first four datasets, because of a small number of features, the best solution was to store all new individuals from all generations. For the next three

Fig. 3 The comparison of the time needed to complete each generation of GA in the case of the original GAAM and fGAAM with *storedN* calculated with the function *HowManyToStore()*; **a** *Postoperative*, **b** *Adult*, **c** *Dermatology*, **d** *Humanactivity*, **e** *SMK-CAN-187*, **f** *Gisette*, **g** *Yale_74x74*, **h** *Coil100*

Table 5 The comparison of results obtained with GAAM and fGAAM (with *storedN* = 0); the average time needed to complete one generation (columns 4 and 5), the time saved (column 6), the average number of individuals excluded from the evaluation process in fGAAM (column 7), and the classification accuracy obtained with a feature set returned by both algorithms (columns 2 and 3)

Dataset	Accuracy [%]		Stored	storedN	Average time [s]		Gains [%]
	GAAM	fGAAM			GAAM	fGAAM	
Postoperative	73.18	73.04	205	495	1.03	0.04	95.74
Adult	79.54	79.53	1513	3546	68.47	8.56	87.48
Dermatology	95.00	94.84	124	124	1.69	1.25	26.03
Humanactivity	96.92	97.04	3062	3062	62.76	37.27	40.60
SMK-CAN-187	84.73	86.06	159	159	2.02	1.76	12.76
Gisette	94.33	93.81	884	884	32.94	31.86	3.29
Yale_64x64	89.80	93.43	129	129	2.73	2.59	5.09
Coil100	90.17	90.07	5828	57762	808.92	687.99	14.94
Mean							35.74

Table 6 The time gains (in %) obtained with different numbers of individuals stored in the individuals set; columns 2-7: the time gains calculated in regard to the average time needed to complete one generation of the original GAAM; columns 8-12: the time gains calculated in regard to fGAAM with *storedP* = 0

Dataset	In regard to original GAAM [%]						storedN
	storedP = 0	1	3	10	100		
Postoperative	38.65	85.10	95.55	96.12	96.12		95.74
Adult	19.59	36.50	53.29	87.11	87.96		87.48
Dermatology	12.11	22.07	32.79	55.80	64.37		26.03
Humanactivity	5.64	12.03	17.51	30.53	44.01		40.60
SMK-CAN-187	5.98	7.40	7.82	− 0.60	− 25.52		12.76
Gisette	2.02	5.13	6.90	5.26	− 3.86		3.29
Yale_64x64	4.11	8.53	9.71	4.92	− 17.27		5.09
Coil100	6.00	4.14	12.05	11.95	14.80		14.94
Mean	11.76	22.61	29.45	36.39	32.58		35.74
Postoperative	–	46.45	56.90	57.47	57.47		57.09
Adult	–	16.91	33.70	67.52	68.37		67.89
Dermatology	–	9.96	20.68	43.69	52.26		13.92
Humanactivity	–	6.39	11.87	24.89	38.37		34.96
SMK-CAN-187	–	1.42	1.84	− 6.58	− 31.50		6.78
Gisette	–	3.11	4.88	3.24	− 5.88		1.27
Yale_64x64	–	4.42	5.60	0.81	− 21.38		0.98
Coil100	–	− 1.86	6.05	5.95	8.80		8.94
Mean	–	10.85	17.69	24.62	20.81		23.98

datasets (*SMK-CAN-187*, *Gisette* and *Yale_64x64*), the better solution was to store a rather small number of individuals (the highest acceleration was obtained for *storedP* = 3). And finally, for the last dataset (*Coil100*), the situation was very different than in the previous cases. Here, as a result of the long evaluation time of a single individual (about 5 s), storing too small number of individuals (*storedP* = 1) even increased the algorithm processing time compared to the fGAAM evaluating only individuals with unique phenotypes (*storedP* = 0).

The analysis of the results obtained for individual datasets also shows that although the fGAAM with an individual level of *storedN* did not provide the best acceleration for each dataset, its results were stable across all datasets, meaning that the processing time was shorter or at most the same as the time of the fGAAM with *storedP* = 0. For this

reason, although the fGAAM with *storedP* = 10 provided slightly higher average acceleration than the fGAAM with an individual level of *storedN* (23.98% vs 24.62%), we think that it is better to allow the fGAAM to calculate the number of individuals by itself rather than to use a fixed level of *storedP* or *storedN*. We should mention here that stable results were also obtained for *storedP* = 3, but for this case the average acceleration of the algorithm was about 6.5% smaller.

4.4 Stage 4

In the last stage of our study, we ran four GAs commonly used for feature selection – Holland, Holland with penalty term, Culling, and NSGA II – over the eight datasets described in Tables 1 and 2. To ensure a fair comparison

Table 7 A comparison of the average time needed to process one algorithm generation and the accuracy and number of genes of the best individual returned by fGAAM and four other GAs

Dataset	fGAAM	Holland	Holland with penalty	Culling	NSGA II
Accuracy of the best individual [%]					
Postoperative	73.04	73.60	73.05	56.66	72.58
Adult	79.53	79.53	79.52	78.46	79.07
Dermatology	94.84	98.17	97.67	79.57	98.21
Humanactivity	97.04	97.82	97.57	89.83	97.88
SMK-CAN-187	86.41	-	-	83.68	-
Gisette	93.81	-	-	91.25	-
Yale_64x64	79.50	-	-	64.47	-
Coil100	90.07	94.07	94.01	81.87	93.89
Mean	86.78	88.64	88.36	78.22	88.33
Average time needed to process one generation [s]					
Postoperative	0.04	1.09	0.92	0.03	2.98
Adult	8.56	79.12	45.23	0.42	163.41
Dermatology	1.25	2.22	1.90	0.12	7.38
Humanactivity	37.27	194.22	131.47	1.82	531.21
SMK-CAN-187	1.94	-	-	36.64	-
Gisette	31.86	-	-	33.69	-
Yale_64x64	1.16	-	-	13.45	-
Coil100	687.99	8739.73	8967.29	169.57	27045.78
Mean	96.26	1803.28	1829.36	31.97	5550.15
Number of genes in the best individual					
Postoperative	3	2	2	3	1
Adult	10	8	8	5	3
Dermatology	6	28	17	5	12
Humanactivity	10	40	30	4	26
SMK-CAN-187	9	-	-	9	-
Gisette	20	-	-	12	-
Yale_64x64	5	-	-	5	-
Coil100	72	523	534	72	472
Mean	17	120	118	14	103

The best result for each data set is marked with a bold font

among the algorithms, for three methods (Holland, Holland with penalty term, and NSGA II), we set the number of evaluated individuals for each dataset at the same level as in fGAAM: *Postoperative* – 70, *Adult* – 120, *Dermatology* – 80, *Humanactivity* – 120, *SMK-CAN-187* – 100, *Gisette* – 220, *Yale_64x64* – 70, *Coil100* – 90. We could not do the same for the Culling algorithm, since it calculates the number of individuals by itself with regard to the number of genes and features. The other parameters of the algorithms were as follows:

- Holland – probability of crossover: 1, probability of mutation: 0.1, selection method: tournament selection, tournament size: 2 individuals;

- Holland with penalty term – probability of crossover: 1, probability of mutation: 0.1, selection method: tournament selection, tournament size: 2 individuals, fitness function: function composed of two terms, a classic accuracy term and a term penalizing individuals containing a large number of genes/features ($0.5Accuracy + 0.5NumberOfGenes$);
- Culling – percentage of discarded individuals: 10%.

Table 7 presents the results obtained in this stage of the study. As shown in the table, not all methods produced results for all eight datasets. Both Holland algorithms and the NSGA II algorithm encountered the same problem when evaluating the individuals' fitness – because of the high number of features contained in individuals (*SMK-CAN-187*: about 10000; *Gisette*: about 2500; *Yale_64x64*: about 1500), the corresponding covariance matrices did not meet the positive definiteness condition. For these three datasets, only the fGAAM and the Culling algorithm, where the number of genes can be determined by a user, were able to produce results. And, as shown in the table, for all these datasets, the classifiers based on individuals returned by the fGAAM provided higher accuracy than those returned by the Culling algorithm.

Comparing the results of the five remaining datasets in terms of accuracy, we should underline that almost all the algorithms (apart from the Culling algorithm) obtained a similar accuracy for most datasets. The greatest differences in the accuracy (about 4%) happen for the *Dermatology* and *Coil100* datasets, but they should be attributed to a significantly smaller number of genes used in the case of the fGAAM.

While the classification accuracy was similar for almost all algorithms, the number of genes/features and the time needed to perform the algorithm were not. Both conditions definitely favored first, the Culling algorithm and then, the fGAAM algorithm. The difference in time was huge: while the Culling algorithm and the fGAAM needed on average less than 100 s to process one generation, both Holland algorithms took almost 20 times longer (1803 s and 1829 s), and NSGA II took 55 times longer (5550 s). In the case of the number of genes/features, the differences among algorithms were much smaller, but still, in the case of both Holland algorithms and NSGA II 6–7 times more genes/features were used than in the case of the fGAAM and the Culling algorithm.

5 Conclusions

The aim of this paper was to present a modified version of a genetic algorithm with aggressive mutation (GAAM). The new version, called fGAAM (fast GAAM), improved the original algorithm in two aspects: shorter processing time

and wider range of application. As we shown in the paper, the fGAAM processing time, calculated for eight data sets of different number of features, classes, and examples, was about 35% shorter than that of the original GAAM.

Moreover, when compared to other GAs (genetic algorithms) used for feature selection, the fGAAM provided similar accuracy, but because of using significantly smaller number of genes in the individuals, it was incomparably quicker. The average acceleration of the processing time reached a factor of 20 (compared to two Holland GAs) and a factor of 50 (compared to NSGA II (nondomination sorting GA)). It should be also emphasized here that a smaller number of genes allowed the fGAAM to process the datasets (*SMK-CAN-187*, *Gisette*, *Yale_64x64*) that could not be processed by the three GAs using binary coding.

Addressing the second feature, range of application, a new parameter (mutation probability) was added to the algorithm to adapt it to data sets of any number of features/examples. Due to this parameter, the fGAAM was capable of processing data sets of higher number of features in shorter time than the original GAAM (e.g., for *Coil100* composed of 1024 features and 7200 examples, the time reduction from 102 to 12.5 hours was obtained with mutation probability set to 10%).

The proposed approach to accelerate the GA processing time by storing a given number of individuals (determined by the algorithm) brought significant time savings. We believe that similar approaches, where the new individuals are compared to individuals evaluated previously, can also have similar benefits in other GAs or in other heuristic methods used for feature selection.

Funding This work was supported by the National Science Centre, Poland, Grant No. 2017/27/B/HS4/01216.

Availability of data and material Only open datasets were used in the paper.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Code availability Code available on demand from the corresponding author.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not

permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Deon G, Peterson David A, Anderson Charles W, Thaut Michael H (2003) Comparison of linear, nonlinear, and feature selection methods for EEG signal classification. *IEEE Trans Neural Syst Rehabil Eng* 11(2):141–144
2. Marcin K, Andrzej M, and Remigiusz JR (2011) A new method of EEG classification for BCI with feature extraction based on higher order statistics of wavelet components and selection with genetic algorithms. In *Proceedings of International Conference on Adaptive and Natural Computing Algorithms*, pages 280–289
3. Peterson DA, Knight JN, Kirby MJ, Anderson CW, Thaut MH (2005) Feature selection and blind source separation in an EEG-based brain-computer interface. *EURASIP J Appl Signal Process* 19:3128–3140
4. Vafaie H, Imam IF, et al (1994) Feature selection methods: genetic algorithms vs. greedy-like search. In *Proceedings of the international conference on fuzzy and intelligent control systems*, 51: 28
5. Zhao M, Fu C, Ji L, Tang K, Zhou M (2011) Feature selection and parameter optimization for support vector machines: A new approach based on genetic algorithm with feature chromosomes. *Expert Syst Appl* 38(5):5197–5204
6. Heba L, Conway BA (2007) Understanding intention of movement from electroencephalograms. *Expert Syst* 24(5):295–304
7. Whitley D, Beveridge JR, Guerra-Salcedo C, and Graves C (1997) Messy Genetic Algorithms for Subset Feature Selection. *Proceedings of the 7th International Conference on Genetic Algorithms*, pages 568–575
8. Yang J and Honavar V (1998) Feature subset selection using genetic algorithm. In *Feature Extraction, Construction and Selection. The Springer International Series in Engineering and Computer Science*, volume 453, pages 117–136. Springer
9. Riccardo Leardi (2000) Application of genetic algorithm-PLS for feature selection in spectral data sets. *J Chemom* 14(5–6):643–655
10. Xu W, Zhang M, Sawchuk AA, Sarrafzadeh M (2012) Robust human activity and sensor location corecognition via sparse signal representation. *IEEE Trans Biomed Eng* 59(11):3169–3176
11. Rejer I, Lorenz K (2015) Classic genetic algorithm vs. genetic algorithm with aggressive mutation for feature selection for a brain-computer interface. *Przegląd Elektrotechniczny* 91(2):98–102
12. Kalyanmoy D, Amrit P, Sameer A, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans Evol Comput* 6(2):182–197
13. Hamdani TM, Won JM, Alimi AM, Karray F (2007) Multi-objective feature selection with NSGA II. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 240–247
14. Huang B, Buckley B, Kechadi TM (2010) Multi-objective feature selection by using NSGA-II for customer churn prediction in telecommunications. *Expert Syst Appl* 37(5):3638–3646
15. Soyel H, Tekguc U, Demirel H (2011) Application of NSGA-II to feature selection for facial expression recognition. *Comput Elect Eng* 37(6):1232–1240
16. Tekguc U, Soyel H, & Demirel H (2009) Feature selection for person-independent 3D facial expression recognition using NSGA-II. In *2009 24th International Symposium on Computer and Information Sciences, ISCIS 2009*, pages 35–38

17. Cui Z, Chang Y, Zhang J, Cai X, Zhang W (2019) Improved nsga-iii with selection-and-elimination operator. *Swarm Evol Comput* 49:23–33
18. Ishibuchi H, Imada R, Setoguchi Y, and Nojima Y (2016) Performance comparison of nsga-ii and nsga-iii on various many-objective test problems. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 3045–3052. IEEE
19. Garrett EB, Baum D, Boneh C (2001) Where Genetic Algorithms Excel. *Evol Comput* 9(1):93–124
20. Elad YT, Inbar Gideon F (2002) Feature selection for the classification of movements from single movement-related potentials. *IEEE Trans Neural Syst Rehabil Eng* 10(3):170–177
21. Rejer I (2015) Genetic algorithms for feature selection for brain-computer interface. *Int J Pattern Recognit Artif Intell* 29(5):155900
22. Rejer I (2013) Genetic algorithms in EEG feature selection for the classification of movements of the left and right hand. In *Proceedings of the 8th International Conference on Computer Recognition Systems CORES*, pages 579–589. Springer Berlin Heidelberg
23. Izabela Rejer (2015) Genetic algorithm with aggressive mutation for feature selection in BCI feature space. *Pattern Anal Appl* 18(3):485–492
24. Krzysztof L and Izabela R (2015) Feature selection with NSGA and GAAM in EEG signals domain. In *Proceedings - 2015 8th International Conference on Human System Interaction, HSI 2015*, pages 94–98
25. Augustyniak P (2014) Subject-optimized feature selection for accurate classification of cardiac beats. *Comput Cardiol* 41:617–620
26. Augustyniak P (2016) Accurate classification of ECG patterns with subject-dependent feature vector. In *Proceedings of the 9th International Conference on Computer Recognition Systems CORES 2015*, pages 533–541
27. Violi ME, Bruni DN, Duy P, Beraldi A (2018) The Mahalanobis Distance for Feature Selection Using Genetic Algorithms: An Application to BCI. In *In New Trends in Emerging Complex Real Life Problems*, pages 73–81. Springer, Cham
28. Noori FM, Naseer N, Qureshi NK, Nazeer H, Khan RA (2017) Optimal feature selection from fNIRS signals using genetic algorithms for BCI. *Neurosci Lett* 647:61–66
29. Noori FM, Qureshi NK, Khan RA and Naseer N (2016) Feature selection based on modified genetic algorithm for optimization of functional near-infrared spectroscopy (fNIRS) signals for BCI. In *Proceedings of 2016 2nd International Conference on Robotics and Artificial Intelligence, ICRAI 2016*, pages 50–53. IEEE
30. Rejer I, Twardochleb M (2018) Gamers' involvement detection from fig EEG data with cGAAM—a method for feature selection for clustering. *Expert Syst Appl* 101:196–204
31. Graff D, Dua C (2019) UCI Machine Learning Repository [<http://archive.ics.uci.edu/ml>]
32. Helou El A (2020) Sensor HAR recognition App
33. Spira A, Beane JE, Shah V, Steiling K, Liu G, Schembri F, Gilman S, Dumas YM, Calner P, Sebastiani P, Sridhar S, John B, Carla L, Timothy A, Norman G, Joseph K, Marc EL, Jerome SB (2007) Airway epithelial gene expression in the diagnostic evaluation of smokers with suspect lung cancer. *Nat Med* 13(3):361–366
34. Gustafson AM, Soldi R, Anderlind C, Scholand MB, Qian J, Zhang X, Cooper K, Walker D, McWilliams A, Liu G, Szabo E, Jerome B, Massion PP, Lenburg ME, Stephen L, Bild AH, Avrum S (2010) Airway PI3K pathway activation is an early and reversible event in lung cancer development. *Sci Transl Med* 2(26):1–25
35. Isabelle Guyon, Steve Gunn (2004) Hur Asa Ben, Dror Gideon (2003) feature selection challenge. *Advances in Neural Information Processing Systems* 17(545–552)
36. Cai D, He X, Hu Y, Han J and Huang T (2007) Learning a spatially smooth subspace for face recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1–7
37. Huang TS, Cai D, He X, Han J (2010) Adaptive graph regularized nonnegative matrix factorization for data representation. *IEEE Trans Pattern Anal Mach Intell* 33(8):1548–1560
38. S Nene, S Nayar, and H Murase. Columbia Object Image Library (COIL-20), 1996
39. Raudys SJ and Jain AK (1991) Small Sample Size Effects in Statistical Pattern Recognition: Recommendations for Practitioners

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.