Cartesian Genetic Programming Is Robust Against Redundant Attributes in Datasets

Henning Cui[a] and Jörg Hähner[b]
University of Augsburg, 86159 Augsburg, Germany
{henning.cui, joerg.haehner}@uni-a.de

Keywords: Cartesian Genetic Programming, CGP, Noisy Attributes, Duplicate Attributes.

Abstract: Real world datasets might contain duplicate or redundant attributes—or even pure noise—which may not be filtered out by data preprocessing algorithms. This might be problematic, as it decreases the performance of learning algorithms. Cartesian Genetic Programming (CGP) is able to choose its own input attributes by design. Thus, we hypothesize that CGP should be able to ignore redundant or noise attributes. In this work, we empirically show that CGP is indeed able to handle such problematic datasets. For this task, six different datasets are extended with different kinds of redundancies: Duplicated-, duplicated and noised-, and pure noise attributes. Different numbers of unwanted attributes are examined, and we present our results which indicate that CGP is robust against additional redundant or noisy attributes in a dataset. We show that there is no decrease in performance as well as no change in CGP's convergence behaviour.

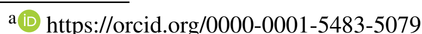
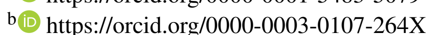
1 INTRODUCTION

Any kind of imperfection in a dataset might decrease the final performance of a learning algorithm. Such flaws might occur in real world datasets, as they could contain inconsistencies, redundant-, noisy-, or duplicate attributes. Preprocessing or data mining algorithms try to improve the quality of a given dataset by *feature-* or *instance selection* techniques, for example. These algorithms reduce the dimensionality of data by removing redundant or conflicting attributes respectively (García et al., 2015). However, most algorithms assume independent and identically distributed data. If this precondition is not given, unneeded attributes might not be filtered out (Rong et al., 2019). This can slow down the training time needed of machine learning algorithms (Hall and Smith, 1997) or decrease their accuracy (Duangsoithong and Windeatt, 2009). The other way around, redundant features might not even impede machine learning algorithms. Duangsoithong and Windeatt found that removing redundant features can decrease the accuracy of ensemble learning methods (Duangsoithong and Windeatt, 2009). Thus, choosing algorithms that remove every instance of redundancy is not always the best choice.

We believe that *Cartesian Genetic Programming* (CGP) should be able to ignore unwanted attributes through its representation and evolutionary mechanisms. CGP consists of nodes in a grid which are partially connected. By being able to evolve its connections, it might learn to not connect to unwanted attributes—which means that those inputs are ignored. As a result, CGP might not be negatively affected by duplicated or noisy attributes. This means that CGP might be a great choice to consider for datasets which could not be preprocessed perfectly.

Motivated by this hypothesis, we investigate the effects of *additional duplicated-*, *duplicated and noised-*, and *pure noise attributes* in datasets on CGP. For this reason, six UCI (Kelly et al.,) datasets are used and extended with different levels of artificial and unwanted attributes. We examine its effects on CGP's performance and behaviour by empirical means and try to give an answer to our hypothesis.

Based on these goals, we provide a quick overview of related work in the following Section 2. Section 3 then reintroduces CGP. We also discuss our hypothesis more in-depth. Afterwards, Section 4 presents the experimental design of this work. This is followed by Section 5, where we report our results and discuss our research questions as well as our hypothesis. At last, Section 6 summarizes our findings and discusses future research directions.


[a] https://orcid.org/0000-0001-5483-5079

[b] https://orcid.org/0000-0003-0107-264X

Cui, H. and Hähner, J.
Cartesian Genetic Programming Is Robust Against Redundant Attributes in Datasets.
DOI: 10.5220/0012974600003837
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 16th International Joint Conference on Computational Intelligence (IJCCI 2024)*, pages 108-119
ISBN: 978-989-758-721-4; ISSN: 2184-3236
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

2 RELATED WORK

Various previous works investigated the effects of redundant data on algorithms. However, to the best of our knowledge, we are the first to investigate its influence on CGP. Nevertheless, various other articles laid out the foundation for this work.

The investigation of data preprocessing mechanisms is a major research subject in the field of data mining. There are numerous algorithms for different kinds of preprocessing tasks (García et al., 2015).

Feature selection is another important topic in the realm of data mining, as the goal of these algorithms is to reduce the dimensionality of data. This can be achieved, among other things, by using genetic algorithms (Tiwari and Singh, 2010; Xu et al., 2009). It is also possible to use fuzzy genetic algorithms, as was done by Fung et al. (Fung et al., 1997). Other possibilities include the application of differential evolution algorithms, as was done by Bidgoli et al. (Bidgoli et al., 2019).

Instance selection is another technique that is used in combination with feature selection. Here, the goal is to remove faulty data. Again, genetic algorithms can be considered. Tsai et al. used genetic algorithms for both feature and instance selection (Tsai et al., 2013). They also examined the effects of performing only instance-, or only feature selection, as well as performing both. Both feature- and instance selection can also be performed simultaneously by using genetic algorithms (Albuquerque et al., 2020).

3 CARTESIAN GENETIC PROGRAMMING

Cartesian Genetic Programming is a supervised learning algorithm invented in 1999 by Miller (Miller, 1999). In this section, we reintroduce CGP's representation, its standard evolutionary operators, and explain our hypothesis.

3.1 Representation

The standard CGP version we are using in this work is represented by a *directed, acyclic and feed-forward graph*. It is a grid which consists of partially connected *nodes*. Originally, it was conceptualized with a $c \times r$ grid with $c \in \mathbb{N}_+$ and $r \in \mathbb{N}_+$. However, today's standard consists of a CGP model with only one row for most applications (Miller, 2011). Furthermore, CGP's representation allows for an *arbitrary amount* of program inputs and outputs.

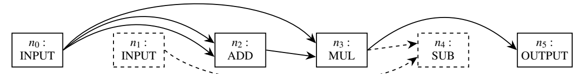


Figure 1: Example graph defined by a CGP genotype. The dashed node and connections are *inactive* due to not contributing to the output.

These aforementioned nodes can be categorized into *input-*, *output-*, and *computational* nodes. The first type, input nodes, directly receive the program input to relay them to other nodes. Output nodes redirect the output of an input- or computational node. Both types—input and output nodes—do not change their respective ingoing value. As for the last category: Computational nodes do change their inputs. They are represented by one function- and a connection genes, with $a \in \mathbb{N}_+$ being the maximum *arity* of one function in the whole function set. Function genes encode the function of a node, while the connection genes define the nodes respective input. This is done by defining a path between a previous and the current node.

Another important distinction is the difference between *active* and *inactive* nodes—both input- and computation nodes can be grouped into one of these two categories. On the one hand, active nodes are part of a path to one or multiple output nodes. Because of that, they contribute to the program's final output. On the other hand, inactive nodes are not part of a path to output nodes. Hence, they do not contribute to the program's final output. While there are methods to enforce all nodes to be active, the existence of inactive nodes contributes to an improvement in CGP's evolutionary search. This allows for *neutral genetic drift* (Miller and Smith, 2006; Turner and Miller, 2015), which may lead to better fitness values and/or faster convergence.

An illustrative example of a graph defined by CGP can be seen in Figure 1. It depicts the genotype with two input-, three computational- and one output node. Active nodes are drawn with a solid line, while inactive nodes are marked by dashed lines. The first two nodes are input nodes, which correspond to a respective input attribute. They are followed by three computational nodes, and one output node at the end. In this example, only the first input is used to calculate an output. The first attribute is taken and added to itself at node n_2. Afterwards, this result is taken and multiplied by the first attribute—with its outcome being the result of this program. Input node n_1 and computational node n_4 are not part of a path to an output node. As a result, they do not contribute to the programs final output and are classified as inactive.

To simplify the description of a CGP configura-

tion in the following work: When we mention a graph defined by CGP with $n \in \mathbb{N}_+$ nodes, this graph will have only one row and n computational nodes. Furthermore, it also contains additional input- and output nodes corresponding to the given learning task.

3.2 Evolutionary Algorithms

In this work, we use an *elitist* $(\mu+\lambda)$ evolution strategy (ES) with $\mu = 1$ and $\lambda = 4$, as is standard in most CGP variants (Miller, 2020). In addition, *neutral search* is included into the $(1+4)$-ES to improve CGP's convergence time and fitness value (Yu and Miller, 2001; Turner and Miller, 2015). That means: When an offspring has the same or better fitness value than the parent, this offspring is always chosen as the next parent. This leads to *neutral drift*, which enables a better exploration of different genotypes (Miller, 2020).

As for the mutation operator, we use one proposed by Goldman and Punch called *Single* (Goldman and Punch, 2013). It works by selecting and mutating random nodes until one active node is mutated. This has the benefit that a change in CGP's phenotype is enforced. When a standard point or probabilistic mutation strategy is used, it is possible that only inactive genes are mutated (Goldman and Punch, 2013; Goldman and Punch, 2015). As a consequence, the quality of the newly mutated individual cannot be evaluated. This might lead to more training iterations needed as well as being stuck at local optima. By enforcing a change in CGP's phenotype with *Single*, no wasted evaluations are performed. It also has the benefit that it does not rely on a mutation rate (Goldman and Punch, 2013).

CGP does not profit from standard crossover operators (Miller, 2011; Cai et al., 2006; Kalkreuth et al., 2017). This is why we also do not include it in this work.

3.3 Ignoring Redundant Attributes

As already mentioned in Section 1, real world datasets might contain duplicate attributes (Hernández and Stolfo, 1998) or unimportant ones (Kumar and Chaurasiya, 2019). This can negatively affect learning algorithms.

We believe that CGP should be able to handle some amount of unnecessary attributes in a dataset. As already mentioned in Section 3.1, the nodes used in CGP are able to mutate their ingoing connection genes. Therefore, the differentiation between active and inactive nodes are important—because some input- or computational nodes are not part of a path to any output nodes (see Figure 1). Because of that, an *inactive input node* means that its corresponding attribute is not used to generate an output. This is why we believe that CGP should handle redundant attributes well. Some nodes may obtain their inputs by being connected to unwanted attributes. Via CGP's evolutionary mechanisms, a node should be able to mutate such connections to use more meaningful inputs. Over time, input nodes corresponding to these unwanted attributes should become inactive. Thus, they do not contribute to the program's output—and do not affect its final fitness value.

4 EXPERIMENTAL DESIGN

In this section, our whole experimental setup is described. We present the datasets used as well as methods to add redundancies into them. Afterwards, a brief introduction into *Bayesian data analysis* and a description of our hyperparameter study is given.

4.1 Problem Sets

As we try to answer our hypothesis empirically, the choice of the right datasets is important. Six classification datasets downloaded from the *UCI Machine Learning Repository* (Kelly et al.,) were chosen according to the recommendations from the genetic programming community (White et al., 2013). We include: *Abalone*, *Credit Approval* (*Credit*), *Statlog Shuttle* (*Shuttle*), *Breast Cancer Wisconsin Diagnosis* (*Cancer*), *Page Blocks Classification* (*Page Blocks*), and *Waveform Version 1* (*Waveform*) (Kelly et al.,). They were chosen to cover different number of instances, number of attributes, and number of classes to predict. These specific values, among others, are shown in Table 1.

Concerning the pre-processing of the datasets, we *standardized* each one. In addition, entries with missing values were removed—as was the case for Credit, for example.

In order to answer our hypothesis, we must gauge CGP's ability to deal with redundant data. Therefore, redundant data is added incrementally to observe CGP's performance differences. The number of additional data is added with respect to the datasets number of attributes. This means that we increase the dataset's size by a fixed, predefined percentage: 20 %, 40 %, $\cdots$, 100 %. These values were chosen in order to gain significant insight into CGP's behaviour without cluttering our results (e.g. using a percentual step size of 10 %) or having too unrealistic values (e.g. more than 100 % redundancies). For example:

Given a dataset with 10 attributes and an increase in its size by 40 %. That means, 4 additional redundant attributes are added, increasing the datasets total number of attributes to 14.

Please note that we do not include symbolic regression benchmarks such as *Korns-12*, which is also one of the recommended benchmarks to use for evaluation purposes (White et al., 2013). Its peculiarity is that five input variables are defined but only two variables are used to generate an output. The goal of Korns-12 is to test if an algorithm is able to ignore unimportant variables. While it fits our scenario, we believe that using it would distort our results. There is also no proposed method to remove or add unimportant variables. As we cannot remove variables, it is not possible to create a baseline without any unused variables. In addition, as additional unimportant variables cannot be added, different magnitudes of unimportant variables can also not be examined. This would strongly limit our evaluation, as we could not compare it to anything.

4.2 Adding Redundancies into Datasets

In this work, three different ways of adding redundancies are examined: *Duplicating attributes*, *duplicating attributes and noising them*, and adding *pure noise drawn from a Gaussian distribution*. Please note: In order to avoid repetitions, phrases like unwanted attributes, redundancies, etc. are used synonymously.

4.2.1 Duplicate Attributes

The first method randomly *copies* attributes and inserts them into random positions *without changing them*. This operation leads to attributes that should be easily detected and removed without repercussion during the data pre-processing phase of training a model. Thus, this method should be viewed as a second baseline—next to CGP trained without any added redundancies—to evaluate CGP's ability to handle attribute redundancies.

To give a more formal expression of copying random attributes and inserting them into random positions: Let $D \in \mathcal{D}^{n \times a} := (d_{ij})_{\substack{i=1,\cdots,n \\ j=1,\cdots,m}}$ be a dataset containing n entries and a attributes. Furthermore, $\mathcal{D}$ is a set of numbers or a set of categorical values. Additionally, we introduce a parameter $r \in \mathbb{R}_+$ which defines the percentage of additional attributes added to bloat the dataset. This means, we increase the size of a dataset D by $s := \lceil a \cdot r \rceil$.

Expanding D works by drawing random indices $u_1, \cdots, u_s$ at first, with $u_k \in \{1, 2, \cdots, a\}$ for $k = 1, \cdots, s$. These indices $u_1, \cdots, u_s$ define which

Algorithm 1: First redundancy method: Duplicate and insert random attributes.

```
Data: Dataset D, percentage of additional
      attributes r ∈ ℝ+
t ← 0;
s ← ⌈|a · r⌉;
D′ ← Clone(D);
U ← {u1, ··· , us} random indices from
  {1, 2, ··· , a};
foreach u ∈ U do
    d′ ← D[:, u];
    v ← randomly drawn number from
      {1, 2, ··· , a + t};
    expand D′ by shifting all elements after v
      one dimension to the right and inserting
      d′ into D[:, v];
    t ← t + 1;
end
return D′
```

attribute columns in D will be copied. In order to finally expand our dataset, we must first create a copy of D called D', which we will expand upon and add redundancies. Then, for each index u_k with $k = 1, \cdots, s$, we copy a set $d'_{u_k} = \{d_{1,u_k}, \cdots, d_{n,u_k}\} \in D^n$. At last, we draw a random index $v \in \{0, \cdots, a'\}$ and insert d'_{u_k} into the vth column of D', with a' being D''s current number of attributes. That means, we copy the u_kth attribute column in D, shift all elements after a random column position v to the right, and insert it into position v in D'. To further clarify our approach, we include its pseudocode in Algorithm 1.

4.2.2 Duplicate Attributes and Add Noise

Our second method works by *duplicating attributes* and *adding noise* before inserting them into the dataset. This method is a more realistic version of redundant attributes in a dataset. Sensor readings might drift and/or fluctuate. Due to this reason, for example by placing two sensors close to each other, their readings should not lead to the exact same value.

To perform this second method, similar steps compared to Algorithm 1 have to be performed. We only differ at the last steps: For each index u_k with $k = 1, \cdots, s$, we copy a set $d'_{u_k} = \{d_{1,u_k}, \cdots, d_{n,u_k}\} \in D^n$. However, before we insert d'_{u_k} into the new dataset D', it must be noised. In this work, noising each attribute means that it is value changes by increasing or decreasing it by up to ten percent. Hence, for each value in d'_{u_k}, we draw a uniformly distributed value $x_i \sim \mathcal{U}_{[-0.1,0.1]}$ for $i = \{0, \cdots, n\}$. Afterwards, noise

Table 1: The full name of datasets used in this work, the dataset's size (Size), number of classes to predict (# Classes), number of attributes (# Attrib.), and its number of additional attributes given a specific percentage of redundancy (x %).

Dataset	Size	# Classes	# Attrib.	20 %	40 %	60 %	80 %	100 %
Abalone	4,177	28	8	+2	+4	+5	+7	+8
Breast Cancer Wisconsin Diag.	569	2	30	+6	+12	+18	+24	+30
Credit Approval	690	2	15	+3	+6	+9	+12	+15
Page Blocks Classification	5,473	5	10	+2	+4	+6	+8	+10
Statlog (Shuttle)	58,000	7	9	+2	+4	+6	+8	+9
Waveform Version 1	5,000	3	21	+5	+9	+13	+17	+21

Algorithm 2: Second redundancy method: Duplicate attribute and add noise.

Data: Dataset D, percentage of additional attributes $r \in \mathbb{R}_+$
$t \leftarrow 0$;
$s \leftarrow \lceil |a \cdot r \rceil$;
$D' \leftarrow Clone(D)$;
$U \leftarrow \{u_1, \cdots, u_s\}$ random indices from $\{1, 2, \cdots, a\}$;
foreach $u \in U$ **do**
  $d' \leftarrow D[:, u]$;
  $v \leftarrow$ randomly drawn number from $\{1, 2, \cdots, a+t\}$;
  foreach $d'_i \in d'$ **do**
    $x \sim \mathcal{U}_{[-0.1, 0.1]}$;
    $d'_i \leftarrow d'_i + d'_i \cdot x$;
  end
  expand D' by shifting all elements after v one dimension to the right and inserting d' into $D[:, v]$;
  $t \leftarrow t + 1$;
end
return D'

is added for each d'_{i,u_k}:

$$d'_{i,u_k} \leftarrow d'_{i,u_k} + d'_{i,u_k} \cdot x_i$$

This noised attribute set is then inserted into D' into a random attribute index v after all elements after v are shifted to the right. Again, to further clarify our approach, we refer to Algorithm 2.

4.2.3 Add Pure Noise

For our last method to evaluate our hypothesis, we *only add Gaussian distributed noise* as redundant attributes. As already mentioned in Section 4.1, we standardize our data. That means, each dataset has a mean of zero and a standard deviation of one. Thus, we are able to draw from a Gaussian distribution with a mean of zero and standard deviation of one. As a result, we generate truly redundant attributes—which would be equivalent of using faulty or wrongly configured sensors, for instance.

Algorithm 3: Third redundancy method: Insert random noise-attributes.

Data: Dataset D, number of D's attributes a, percentage of additional attributes $r \in \mathbb{R}_+$
$t \leftarrow 0$;
$s \leftarrow \lceil a \cdot r \rceil$;
$D' \leftarrow Clone(D)$;
repeat s **times**
  $v \leftarrow$ randomly drawn number from $\{1, 2, \cdots, a+t\}$;
  $p = \{p_1, \cdots, p_n\}$ with $p_i \sim \mathcal{N}(0, 1)$ and $i = \{1, \cdots, n\}$;
  expand $\mathcal{D}'$ by shifting all elements after v to the right and inserting p into $\mathcal{D}[:, v]$;
end
return $\mathcal{D}'$

Adding redundant attributes needs similar steps to Algorithm 1. Again, we must define similar parameters and sets: Our dataset D, parameter r to define the percentage of a datasets increase in attributes, and a cloned dataset D'. We differ from the first two approaches as we do not rely on D to generate our redundant data. Instead, we insert pure noise. For this approach, a noise vector p is generated by drawing from a Gaussian distribution: $p = \{p_1, \cdots, p_n\}$ with $p_i \sim \mathcal{N}(0, 1)$ and $i = \{1, \cdots, n\}$. Please note that each value in p is drawn *independently*. Then, a random index v is drawn. It represents the position of D', into which our noise vector p is added. At last, p is added into D' at the attribute position v. This works by shifting all elements after v to the right and inserting p into the position v. Again, Algorithm 3 depicts this process for further clarification.

4.3 Bayesian Data Analysis

In order to gauge the effects of redundant data, our results must be ranked according to their respective *final fitness value*. As this number cannot be neg-

ative, common statistical tests—such as *Student's t-test*, which uses a *Student's t-distributions*—should *not* be used. The reason is that such distributions cannot be expected to model the data well (Kruschke, 2013). On that account, we perform a *Bayesian data analysis* for the posterior distributions of our results. The model to compare the algorithms is based on the *Plackett-Luce model* described by Calvo et al. (Calvo et al., 2018). It allows the computation of a set of ranked options by estimating the probabilities of each of the options to be the one with the highest rank. For this task, we use the Python library *cmpbayes* (Pätzel, 2023) for all statistical models. As is standard practice, prior sensitivity analyses were conducted to ensure the robustness of all models. For more informations regarding the models, we refer to Kruschke (Kruschke, 2013) and Pätzel (Pätzel, 2023).

4.4 Configuration of CGP and Its Training

In our experiments, we used a standard CGP configuration. That means: No crossover, a modified $(1+4)$-ES as described in Section 3, and *Single* (Goldman and Punch, 2013) mutation. The only hyperparameter that must be optimized in our setting is CGP's number of computational nodes n. In order to have a fair comparison, n was optimized for each combination of: Dataset; no redundant attributes, or additional redundant attributes with respect to one of the three redundancy types introduced in this work and given a specific percentage of redundancy.

We investigated $n \in \{50, 100, \cdots, 2000\}$ for each aforementioned combination. As the datasets mentioned in Section 4.1 do not contain a train/test split, k-fold cross-validation with $k = 5$ was employed to generate a training- and a test dataset. Each configuration was tested 20 times with independent repetitions and completely random seeds. Afterwards, to find the best n, we ranked them according to the *Plackett-Luce model* described by Calvo et al. (Calvo et al., 2018) with respect to their final test fitness value. Please note: The final hyperparameters found and used are listed in our results.

Because all datasets can be categorized as classification tasks, we use the same fitness metric during the training of all datasets. We chose the *Balanced Accuracy*, which should be used for imbalanced datasets. It is defined by calculating the average of recall obtained on all classes. The reason is that some datasets (e.g. Shuttle) are heavily unbalanced. As a result, a standard accuracy metric would not reflect CGP's fitness accurately.

A single run has a budget of 100,000 iterations. That means, a run is stopped after the given budget. Additionally, a run is stopped preliminary when the fitness value of the training data reaches a value less than 0.01. In this case, we classify a dataset as *solved*.

To generate our *final results*, each configuration used the best n found. The tests were run again for 50 times, again, with independent repetitions and different random seeds. Furthermore, a standard 5-fold cross-validation was used to generate the test fitness values.

5 EVALUATION

In order to find the effects of unnecessary attributes in datasets on CGP, we conducted an empirical study[1]. We try to answer the following three research questions to find a solution to our hypothesis:

Q1: How does having redundant attributes in a dataset affect CGP? Especially regarding its

- Number of iterations until a solution is found (*I2S*),
- Fitness value, and
- Number of active nodes.

Q2: Does CGP manage to ignore redundant attributes?

Q3: How do unnecessary attributes affect CGP's behaviour?

To increase readability, we will introduce the following abbreviations: A CGP model trained on a dataset without noise will be called *baseline*; a CGP model trained on a dataset with duplicated attributes (see Section 4.2.1) will be called CGP+DA; a CGP model trained on a dataset with duplicated and noised attributes (see Section 4.2.2) will be called CGP+DA&NOISE; and finally, a CGP model trained on a dataset that has additional Gaussian distributed noise (see Section 4.2.3) is called CGP+NOISE.

5.1 Results of Redundant Attributes on CGP

We show our results in the Appendix in Table 2, Table 3 and Table 4. They show the results for CGP+DA, CGP+DA&NOISE and CGP+NOISE respectively on all datasets, as well as their baselines. We show the percentage of additional attributes (% Add.), the number of nodes (Nodes), number of mean

[1] Implementation and datasets can be found at: https://github.com/CuiHen/redundant_attributes_with_CGP

active nodes (Active), the mean and standard deviation of iterations until a dataset is solved or stopped ($I2S Mean \pm Std$), the mean and standard deviation of achieved test fitness ($Fit Mean \pm Std$), the mean percentage of redundant attributes that are used to generate an output (% Red), and the probability of a solution being the best per dataset with respect to its test fitness ($p(best)$).

Regarding the *I2S*, all datasets except Cancer are not classified as solved—because they all were stopped after 100,000 iterations. Hence, only the Cancer dataset can be solved with the given budget. In most cases, the mean *I2S* is relatively equal. Thus, regarding the effect of redundant attributes on CGP—given the Cancer dataset—there is the trend that this has *no effect on CGP's time to solution*. However, as this conclusion is drawn by evaluating only a single dataset, this outcome should be treated with reservations.

Similarly, there is no clear correlation between levels of noise and computational nodes needed. This statement also applies to the mean number of active nodes.

As for CGP's fitness values, a similar conclusion can be drawn. For a given dataset, the mean fitness values and their standard deviations are relatively similar. This is also reflected in their probabilities of being the best solution per dataset. There is no clear winner, given the calculated probabilities. All probabilities are relatively similar, with no configuration dominating over the other. That means that adding redundant attributes into a dataset *will probably not* affect CGP's fitness value.

Please note: For better readability and understandability, our three methods of adding unwanted attributes are separated into three tables respectively. However, all three redundancy methods are compared/ranked against the same baseline. On all three methods, similar results can be seen. This means: These three types of additional noise *do probably not affect* CGP's *I2S* or fitness value, regardless of *their percentage of additional attributes*.

Another interesting fact is that CGP *should* be able to ignore redundant attributes. Considering the Waveform dataset, when attributes are duplicated, or duplicated and noised, CGP is able to ignore most of unwanted attributes. Only 2 % to 7 % of redundant attributes are used. In the case of CGP+NOISE, it will only use 8 % to 13 % of noise to calculate an output, given the Shuttle dataset. As there is little to no difference in their respective fitness values, we can conclude that CGP should be able to *ignore redundant attributes*. However, this is not the case for all datasets. Given the Shuttle dataset, for example, the

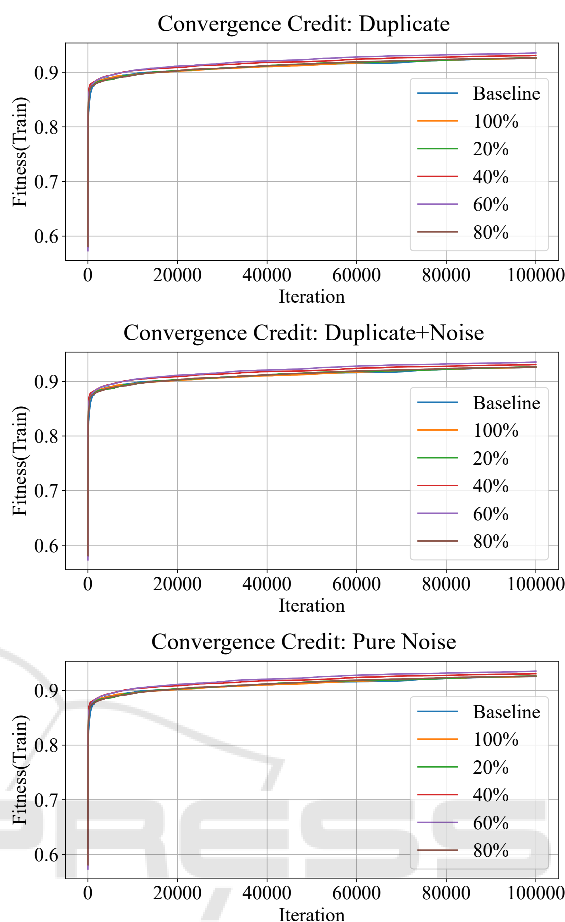


Figure 2: Convergence plots for all three types of data redundancy on the Credit dataset.

percentage of duplicate input attributes of CGP+DA and CGP+DA&NOISE are up to 42 %. Still, relatively high fitness values are achieved for this dataset. Another more prominent example is given in Table 4, when CGP+NOISE is considered. Given the Credit dataset, CGP's inputs are up to 50 % pure Gaussian noise. Thus, depending on the given learning problem, CGP may include redundant information into calculating its final program output with no obvious effect. A reason might be that the redundant information are not meaningfully included into the calculation of the output. Such might happen when, for example, a noise attribute is added to another value during an intermediate step but subtracted immediately afterwards. This noise attribute is then listed as used but it does not actually contribute to the programs final output.

5.2 Convergence Behaviour

To better understand the convergence behaviour of CGP, convergence plots for all three types of data redundancy were considered. We also classified the different behaviours according to Stegherr et al. (Stegherr. et al., 2023). On that account, we investigate the progression of the mean fitness value of the train split.

Our experiments show that each configuration depicts the same convergence behaviour: *Fast to Slow*. CGP's fitness improves drastically during the first iterations. However, its progression slows down and high numbers of iterations are needed for small performance increases. Interestingly, this behaviour can be seen for the baseline as well as CGP+DA, CGP+DA&NOISE, and CGP+NOISE. Furthermore, the percentage of added attributes do not affect CGP's convergence. This leads us to the following conclusion: A bloated dataset *does not affect CGP's convergence behaviour*.

An illustrative example of CGP's convergence on all three types of redundancy is depicted in Figure 2. We only exemplarily show the Credit dataset. The reason is that the other datasets depict the exact same behaviour.

6 CONCLUSION

In this work, we investigated the effects of additional duplicated-, duplicated and noised-, and purely noise attributes in datasets on CGP. Six different datasets and five levels of noise were examined. They were also compared against a baseline, which describe a CGP model trained on a dataset without any type of additional artificial redundancy.

Considering these three types of additional attributes, we found that they do not affect CGP's achieved fitness values in our testing. In addition, there is also no effect on CGP's convergence behaviour. We classified CGP's behaviour according to Stegherr et al. (Stegherr. et al., 2023) and found, that each configuration shows the same convergence behaviour: *Fast to slow*. When we examine CGP's number of iterations until a solution is found (*I2S*), no clear answer can be given. Five out of six datasets could not be classified as solved within its given budget. Thus, they cannot be used to answer this research question. Still, *in one case*, it can be seen that additional attributes do not effect CGP's *I2S*.

Another research question is: Does CGP manage to ignore redundant attributes? In some cases, CGP is able to almost completely ignore them. This suggests that CGP is indeed able to use only relevant information to generate an output. However, there are also cases where 50 % of inputs are purely noise—without an effect on its fitness value.

Furthermore, these results are valid for all levels of additional attributes. Thus, we can conclude that CGP is *robust* to additional duplicated-, duplicated and noised-, and purely noise attributes in datasets. These types of additional, unwanted attributes *do not affect* CGP's performance or behaviour.

As for future work, there are still various different settings that could be examined to further investigate CGP. More datasets and different types of additional attributes could be investigated. Another possibility is to integrate different preprocessing methods. A dataset can be extended with attributes that are not filtered out by said preprocessing methods. This adds another level of difficulty, as the additional attributes can truly not be distinguishable from the real data. Including such data might or might not influence CGP.

In addition, our bloated datasets should be evaluated with various other learning algorithms. These results should then be compared with our findings to show if CGP is a valuable choice on noised datasets.

ACKNOWLEDGEMENTS

The authors would like to thank the German Federal Ministry of Education and Research (BMBF) for supporting the project SaMoA within VIP+ (grant number 03VP09291).

REFERENCES

Albuquerque, I. M. R., Nguyen, B. H., Xue, B., and Zhang, M. (2020). A novel genetic algorithm approach to simultaneous feature selection and instance selection. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 616–623.

Bidgoli, A. A., Ebrahimpour-Komleh, H., and Rahnamayan, S. (2019). A novel multi-objective binary differential evolution algorithm for multi-label feature selection. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 1588–1595.

Cai, X., Smith, S. L., and Tyrrell, A. M. (2006). Positional independence and recombination in cartesian genetic programming. In Collet, P., Tomassini, M., Ebner, M., Gustafson, S., and Ekárt, A., editors, *Genetic Programming*, pages 351–360, Berlin, Heidelberg. Springer Berlin Heidelberg.

Calvo, B., Ceberio, J., and Lozano, J. A. (2018). Bayesian inference for algorithm ranking analysis. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '18, page 324–325,

New York, NY, USA. Association for Computing Machinery.

Duangsoithong, R. and Windeatt, T. (2009). Relevant and redundant feature analysis with ensemble classification. In *2009 Seventh International Conference on Advances in Pattern Recognition*, pages 247–250.

Fung, G., Liu, J., Chan, K., and Lau, R. (1997). Fuzzy genetic algorithm approach to feature selection problem. In *Proceedings of 6th International Fuzzy Systems Conference*, volume 1, pages 441–446 vol.1.

García, S., Luengo, J., Herrera, F., et al. (2015). *Data preprocessing in data mining*, volume 72. Springer.

Goldman, B. W. and Punch, W. F. (2013). Reducing wasted evaluations in cartesian genetic programming. In Krawiec, K., Moraglio, A., Hu, T., Etaner-Uyar, A. Ş., and Hu, B., editors, *Genetic Programming*, pages 61–72. Springer Berlin Heidelberg.

Goldman, B. W. and Punch, W. F. (2015). Analysis of cartesian genetic programming's evolutionary mechanisms. 19(3):359–373.

Hall, M. A. and Smith, L. A. (1997). Feature subset selection: a correlation based filter approach.

Hernández, M. A. and Stolfo, S. J. (1998). Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37.

Kalkreuth, R., Rudolph, G., and Droschinsky, A. (2017). A new subgraph crossover for cartesian genetic programming. In McDermott, J., Castelli, M., Sekanina, L., Haasdijk, E., and García-Sánchez, P., editors, *Genetic Programming*, pages 294–310. Springer International Publishing.

Kelly, M., Longjohn, R., and Nottingham, K. The UCI machine learning repository. https://archive.ics.uci.edu.

Kruschke, J. K. (2013). Bayesian estimation supersedes the t test. *Journal of Experimental Psychology: General*, 142(2):573–603.

Kumar, S. and Chaurasiya, V. K. (2019). A strategy for elimination of data redundancy in internet of things (iot) based wireless sensor network (wsn). *IEEE Systems Journal*, 13(2):1650–1657.

Miller, J. and Smith, S. (2006). Redundancy and computational efficiency in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation*, 10(2):167–174.

Miller, J. F. (1999). An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 2*, GECCO'99, page 1135–1142, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Miller, J. F. (2011). *Cartesian Genetic Programming*. Springer Berlin Heidelberg.

Miller, J. F. (2020). Cartesian genetic programming: its status and future. *Genetic Programming and Evolvable Machines*, 21(1):129–168.

Pätzel, D. (2023). cmpbayes. https://github.com/dpaetzel/cmpbayes.

Rong, M., Gong, D., and Gao, X. (2019). Feature selection and its use in big data: Challenges, methods, and trends. *IEEE Access*, 7:19709–19725.

Stegherr., H., Heider., M., and Hähner., J. (2023). Assisting convergence behaviour characterisation with unsupervised clustering. In *Proceedings of the 15th International Joint Conference on Computational Intelligence - ECTA*, pages 108–118. INSTICC, SciTePress.

Tiwari, R. and Singh, M. P. (2010). Correlation-based attribute selection using genetic algorithm. *International Journal of Computer Applications*, 4(8):28–34.

Tsai, C.-F., Eberle, W., and Chu, C.-Y. (2013). Genetic algorithms in feature and instance selection. *Knowledge-Based Systems*, 39:240–247.

Turner, A. J. and Miller, J. F. (2015). Neutral genetic drift: an investigation using cartesian genetic programming. 16(4):531–558.

White, D., Mcdermott, J., Castelli, M., Manzoni, L., Goldman, B., Kronberger, G., Jaśkowski, W., O'Reilly, U.-M., and Luke, S. (2013). Better gp benchmarks: Community survey results and proposals. 14:3–29.

Xu, S., Zhou, X., and Sun, Y.-n. (2009). A genetic algorithm-based feature selection method for human identification based on ground reaction force. In *Proceedings of the First ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, GEC '09, page 665–670, New York, NY, USA. Association for Computing Machinery.

Yu, T. and Miller, J. (2001). Neutrality and the evolvability of boolean function landscape. In Miller, J., Tomassini, M., Lanzi, P. L., Ryan, C., Tettamanzi, A. G. B., and Langdon, W. B., editors, *Genetic Programming*, pages 204–217. Springer Berlin Heidelberg.

SCIENCE AND TECHNOLOGY PUBLICATIONS

APPENDIX

Results for Duplicated Attributes

Table 2 shows our results for all datasets when attributes are *duplicated*, according to Algorithm 1.

Results for Duplicated and Noised Attributes

Table 3 shows our results for all datasets when attributes are *duplicated and then noised*, according to Algorithm 2.

Results for Adding Pure Noise

Table 4 shows our results for all datasets when the additional attributes are *pure noise*, according to Algorithm 3.

Table 2: Our results for all datasets when attributes are duplicated. We show the percentage of additional attributes (% Add.), the number of nodes (Nodes), number of mean active nodes (Active), the mean and standard deviation of iterations until a dataset is solved or stopped ($I2SMean \pm Std$), the mean and standard deviation of achieved test fitness ($FitMean \pm Std$), the mean percentage of redundant attributes that are used to generate an output (% Red), and the probability of a solution being the best per dataset with respect to its test fitness ($p(best)$). Results are ranked according to $p(best)$.

Dataset	% Add.	Nodes	Active	$I2SMean \pm Std$	$FitMean \pm Std$	% Red.	$p(best)$
Abalone	60	1,800	371	100k ± 0k	0.15 ± 0.02	0.28	0.2
	Baseline	900	289	100k ± 0k	0.14 ± 0.02	-	0.19
	80	2,000	395	100k ± 0k	0.14 ± 0.02	0.21	0.19
	100	1,500	352	100k ± 0k	0.14 ± 0.02	0.17	0.15
	40	1,850	381	100k ± 0k	0.14 ± 0.02	0.27	0.15
	20	1,400	331	100k ± 0k	0.14 ± 0.03	0.56	0.12
Cancer	20	1,400	49	72k ± 36k	0.95 ± 0.02	0.07	0.2
	Baseline	850	42	70k ± 31k	0.95 ± 0.02	-	0.19
	40	1,400	47	74k ± 33k	0.94 ± 0.03	0.03	0.17
	100	1,450	42	80k ± 29k	0.94 ± 0.02	0.01	0.15
	60	1,700	39	85k ± 27k	0.94 ± 0.02	0.02	0.14
	80	950	38	79k ± 30k	0.95 ± 0.02	0.01	0.14
Credit	20	1,200	65	100k ± 0k	0.86 ± 0.02	0.25	0.23
	40	1,600	71	100k ± 0k	0.86 ± 0.03	0.13	0.19
	80	1,150	68	100k ± 0k	0.85 ± 0.03	0.06	0.17
	Baseline	1,900	68	100k ± 0k	0.85 ± 0.03	-	0.15
	60	1,050	75	100k ± 0k	0.85 ± 0.02	0.09	0.13
	100	1,650	73	100k ± 0k	0.85 ± 0.03	0.06	0.12
Page Blocks	60	950	93	100k ± 0k	0.72 ± 0.04	0.18	0.19
	Baseline	1,500	96	100k ± 0k	0.72 ± 0.03	-	0.18
	100	1,700	98	100k ± 0k	0.73 ± 0.04	0.12	0.18
	20	1,400	99	100k ± 0k	0.72 ± 0.05	0.4	0.18
	40	1,000	89	100k ± 0k	0.72 ± 0.04	0.27	0.14
	80	1,600	100	100k ± 0k	0.72 ± 0.03	0.14	0.13
Shuttle	100	1,250	110	100k ± 0k	0.82 ± 0.04	0.12	0.2
	40	300	78	100k ± 0k	0.82 ± 0.06	0.2	0.18
	Baseline	1,550	116	100k ± 0k	0.82 ± 0.06	-	0.18
	60	1,300	109	100k ± 0k	0.81 ± 0.06	0.15	0.17
	20	550	89	100k ± 0k	0.81 ± 0.06	0.42	0.15
	80	1,650	124	100k ± 0k	0.81 ± 0.06	0.14	0.12
Waveform	Baseline	1,650	50	100k ± 0k	0.6 ± 0.01	-	0.19
	40	600	40	100k ± 0k	0.6 ± 0.01	0.05	0.19
	100	1,450	42	100k ± 0k	0.6 ± 0.01	0.03	0.18
	60	950	38	100k ± 0k	0.6 ± 0.01	0.03	0.16
	20	1,750	48	100k ± 0k	0.6 ± 0.01	0.1	0.16
	80	950	40	100k ± 0k	0.6 ± 0.01	0.03	0.12

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

Table 3: Our results for all datasets when attributes are duplicated and then noised. We show the percentage of additional attributes (% Add.), the number of nodes (Nodes), number of mean active nodes (Active), the mean and standard deviation of iterations until a dataset is solved or stopped ($I2SMean \pm Std$), the mean and standard deviation of achieved test fitness ($FitMean \pm Std$), the mean percentage of redundant attributes that are used to generate an output (% Red), and the probability of a solution being the best per dataset with respect to its test fitness ($p(best)$). Results are ranked according to % Add.

Dataset	% Add.	Nodes	Active	$I2SMean \pm Std$	$FitMean \pm Std$	% Red.	$p(best)$
Abalone	100	1,750	369	100k ± 0k	0.14 ± 0.02	0.2	0.22
	60	2,000	401	100k ± 0k	0.14 ± 0.03	0.25	0.18
	Baseline	900	289	100k ± 0k	0.14 ± 0.02	-	0.17
	20	1,150	328	100k ± 0k	0.14 ± 0.02	0.47	0.15
	40	1,400	345	100k ± 0k	0.14 ± 0.02	0.27	0.15
	80	1,250	327	100k ± 0k	0.13 ± 0.03	0.19	0.14
Cancer	Baseline	850	42	70k ± 31k	0.95 ± 0.02	-	0.2
	100	1,050	34	79k ± 32k	0.95 ± 0.02	0.01	0.18
	40	1,200	38	74k ± 32k	0.95 ± 0.02	0.03	0.18
	20	550	34	70k ± 32k	0.95 ± 0.02	0.05	0.17
	80	950	37	79k ± 30k	0.95 ± 0.02	0.01	0.16
	60	1,500	41	79k ± 28k	0.94 ± 0.03	0.02	0.11
Credit	100	1,400	64	100k ± 0k	0.87 ± 0.02	0.05	0.26
	20	1,150	63	100k ± 0k	0.86 ± 0.02	0.2	0.18
	Baseline	1,900	68	100k ± 0k	0.85 ± 0.03	-	0.15
	40	1,350	68	100k ± 0k	0.85 ± 0.03	0.12	0.14
	60	1,500	66	100k ± 0k	0.85 ± 0.03	0.08	0.14
	80	1,750	71	100k ± 0k	0.85 ± 0.03	0.06	0.12
Page Blocks	80	1,100	86	100k ± 0k	0.73 ± 0.03	0.13	0.21
	20	1,900	110	100k ± 0k	0.72 ± 0.04	0.36	0.18
	40	1,050	88	100k ± 0k	0.73 ± 0.04	0.2	0.18
	Baseline	1,500	96	100k ± 0k	0.72 ± 0.03	-	0.16
	100	1,900	114	100k ± 0k	0.72 ± 0.04	0.11	0.13
	60	1,150	87	100k ± 0k	0.72 ± 0.04	0.16	0.13
Shuttle	20	200	66	100k ± 0k	0.82 ± 0.05	0.26	0.21
	80	750	96	100k ± 0k	0.83 ± 0.04	0.11	0.21
	40	1,650	120	100k ± 0k	0.81 ± 0.05	0.26	0.15
	60	1,950	127	100k ± 0k	0.82 ± 0.08	0.14	0.15
	Baseline	1,550	116	100k ± 0k	0.82 ± 0.06	-	0.14
	100	1,750	129	100k ± 0k	0.81 ± 0.06	0.12	0.14
Waveform	20	600	37	100k ± 0k	0.6 ± 0.01	0.07	0.19
	80	650	31	100k ± 0k	0.6 ± 0.01	0.02	0.19
	40	1,150	48	100k ± 0k	0.6 ± 0.01	0.05	0.18
	Baseline	1,650	50	100k ± 0k	0.6 ± 0.01	-	0.17
	100	1,400	38	100k ± 0k	0.6 ± 0.01	0.02	0.11
	60	500	32	100k ± 0k	0.6 ± 0.01	0.03	0.15

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

Table 4: Our results for all datasets when the additional attributes are pure noise. We show the percentage of additional attributes (% Add.), the number of nodes (Nodes), number of mean active nodes (Active), the mean and standard deviation of iterations until a dataset is solved or stopped ($I2SMean \pm Std$), the mean and standard deviation of achieved test fitness ($FitMean \pm Std$), the mean percentage of redundant attributes that are used to generate an output (% Red), and the probability of a solution being the best per dataset with respect to its test fitness ($p(best)$). Results are ranked according to % Add.

Dataset	% Add.	Nodes	Active	$I2SMean \pm Std$	$FitMean \pm Std$	% Red.	$p(best)$
Abalone	Baseline	900	289	100k ± 0k	0.14 ± 0.02	-	0.27
	20	700	260	100k ± 0k	0.13 ± 0.02	0.63	0.24
	40	400	207	100k ± 0k	0.13 ± 0.03	0.76	0.19
	60	1,050	301	100k ± 0k	0.12 ± 0.03	0.8	0.15
	80	950	284	100k ± 0k	0.11 ± 0.03	0.84	0.09
	100	2,000	396	100k ± 0k	0.11 ± 0.03	0.86	0.07
Cancer	Baseline	850	42	70k ± 31k	0.95 ± 0.02	-	0.26
	40	1,500	43	85k ± 25k	0.94 ± 0.02	0.2	0.19
	20	1,400	42	79k ± 29k	0.94 ± 0.03	0.2	0.18
	60	800	41	81k ± 27k	0.94 ± 0.02	0.2	0.14
	80	1,600	41	85k ± 27k	0.94 ± 0.03	0.18	0.13
	100	1,600	43	91k ± 19k	0.94 ± 0.03	0.16	0.1
Credit	20	1,750	78	100k ± 0k	0.85 ± 0.03	0.53	0.21
	Baseline	1,900	68	100k ± 0k	0.85 ± 0.03	-	0.19
	100	1,200	72	100k ± 0k	0.85 ± 0.04	0.5	0.18
	40	1,250	74	100k ± 0k	0.85 ± 0.03	0.53	0.16
	60	1,000	75	100k ± 0k	0.85 ± 0.02	0.55	0.15
	80	1,300	71	100k ± 0k	0.84 ± 0.03	0.5	0.12
Page Blocks	80	1,100	83	100k ± 0k	0.72 ± 0.04	0.24	0.2
	Baseline	1,500	96	100k ± 0k	0.72 ± 0.03	-	0.18
	20	1,850	103	100k ± 0k	0.72 ± 0.04	0.29	0.17
	60	1,550	96	100k ± 0k	0.72 ± 0.04	0.27	0.17
	100	1,550	91	100k ± 0k	0.71 ± 0.05	0.3	0.14
	40	2,000	106	100k ± 0k	0.71 ± 0.05	0.31	0.13
Shuttle	100	1,600	108	100k ± 0k	0.83 ± 0.06	0.13	0.19
	20	250	71	100k ± 0k	0.83 ± 0.05	0.08	0.18
	40	250	68	100k ± 0k	0.84 ± 0.06	0.08	0.17
	60	800	93	100k ± 0k	0.82 ± 0.05	0.13	0.16
	80	950	94	100k ± 0k	0.83 ± 0.05	0.13	0.16
	Baseline	1,550	116	100k ± 0k	0.82 ± 0.06	-	0.13
Waveform	40	1,100	43	100k ± 0k	0.6 ± 0.01	0.19	0.25
	Baseline	1,650	50	100k ± 0k	0.6 ± 0.01	-	0.19
	80	1,500	44	100k ± 0k	0.6 ± 0.01	0.17	0.16
	60	1,450	40	100k ± 0k	0.6 ± 0.01	0.16	0.14
	100	500	31	100k ± 0k	0.6 ± 0.01	0.11	0.14
	20	1,150	42	100k ± 0k	0.6 ± 0.01	0.16	0.13

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS