An Iterative Algorithm for Increasing Accuracy of a CNN-based Selective Sweep Detection Tool

STEFAN VAN GURP, University of Twente, The Netherlands

Selective sweeps are regions of reduced mutations caused by a favoured mutation in DNA genome sequences. Finding these selective sweeps has been under much research for the past decade. While the tools created from this research have proved to be successful at finding selective sweeps, accurately pinpointing the locations of sweeps remains an open problem in these tools, such as RAiSD. It is the goal of this research paper to continuously decrease the initial fixed window size to narrow the focus of these tools, thereby improving their accuracy. In this research, we used RAiSD-AI to examine a method to fulfil this goal. This method should more accurately predict the locations of the selective sweeps and possibly locate the base pair positions of these sweeps.

Additional Key Words and Phrases: Selective sweep detection, Positive selection, RAiSD-AI.

1 INTRODUCTION

The DNA genome sequence has been discovered since 1953 [5]. Since then, efforts have been made to identify genes affected by positive selection in the evolution of species [14]. Zhao et al. [23] describe that with positive selection, an allele is favoured by natural selection and it becomes more common in a population. A 'selective sweep' occurs when a favoured allele sweeps away other genetic diversity in a locus [23]. Detecting these selective sweeps has been under much research for the past couple decades, where more advanced approaches emerge such as SweeD [18], SweepFinder2 [8], OmegaPlus [3], ASDEC [24] and the tool this research utilises, RAiSD [2]. Accurately detecting the presence of selective sweeps boosts our understanding of the genetic basis of evolution and adaptation in both humans and other organisms [15].

The tool this research focuses on, RAiSD, is an open-source software tool which combines the computationally inexpensive sliding-window algorithm with a memory-aware approach for parsing input data [2]. RAiSD is rather lightweight, as it only allocates a few megabytes of memory [2] and it can scan whole genomes to detect multiple sweep signatures. The sliding window algorithm is used to detect sweep signatures by using a CNN classifier. In order to achieve this, the DNA is encoded into binary and then converted to a grayscale image [23]. The sliding window then parses over pieces of the data, equal to the size of the window, and passes this chunk of data, called a window, to the CNN to give a prediction whether or not there is a sweep in this window.

The issue with sweep detection tools, such as RAiSD, is the input window width. If this width is too large, the precision of finding the selective sweep suffers as the location of the sweep is unknown in the large window. While it is beneficial to know sweeps exist in a particular section, the precise location is more desirable. On the other hand, if the window size is too small, its accuracy suffers as it does not have enough information to accurately determine the sweep's location. This leads us to the purpose of this research paper, namely using increasingly smaller window sizes in order to more accurately determine where sweep sequences are.

1.1 Research Question

To achieve the goal of this research paper, the following research questions were formulated:

> **Main RQ**: What is the effect of incrementally decreasing the window size of a CNN-based selective sweep detection tool on its accuracy compared to the initial fixed window size?
> **Sub RQ1**: What percentage of the top most likely points provides the highest accuracy for the Minimisation Method?
> **Sub RQ2**: What is the most effective initial window size?

To answer these research questions, a new method has been developed that utilises the RAiSD-AI tool. RAiSD has two versions, RAiSD-ZLIB and RAiSD-AI. The difference between these, is that the ZLIB versions uses a GZ file format for processing, whereas the other does not. The proposed method, named the Minimisation method, first scans a DNA dataset, then iteratively creates smaller windows based on the highest scores of the windows from the previous iteration. This results in a small window width that has the highest likelihood of a selective sweep existing in this window.

This paper is organised as follows: Section 2 details the necessary knowledge required to understand this research. Followed by section 3 which outlines some tools that already exist and have the same goal as RAiSD. Afterwards, section 4 explains the details of the Minimisation method's implementation. The results are discussed in section 5 and lastly section 6 concludes this research.

2 BACKGROUND

In this section, we will describe the necessary biological concepts and terminology to understand the data and thus the application of the Minimisation method. This paper assumes the reader is familiar with basic Computer Science related terms and has some knowledge in this field, such as being familiar with some machine learning concepts.

Chromosomes are large DNA molecules that contain the genetic information for an organism [16], which can be found in the centre of cells, the nucleus, as they are essential for DNA to replicate and segregate during cell division [7]. A gene is a piece of DNA on the chromosome and an allele is simply a variation of that [9]. DNA is made up of four bases, that is: adenine (A), cytosine (C), guanine (G) and thymine (T) [10]. These bases join together, A always with T and G always with C to make base pairs [4]. The datasets used by RAiSD are binary representations of these base pair positions. The first, labelled as, 'Mild Bottleneck' and the easiest dataset to work with

TScIT 41, July 5, 2024, Enschede, The Netherlands

[19]. A DNA bottleneck occurs when a population suddenly declines [20], thus resulting in little DNA variation of the population when it begins to grow again. The second dataset, named 'Severe Bottleneck', is the hardest dataset out of the three [19]. The difference between this and the first is the degree of severity of the decline of the population. The last dataset this research utilised is called 'Old Migration' [19]. This refers to the change of DNA after a population has migrated from its original home and stayed in a new region for some period of time.

To create the data in a readable format, RAiSD converts these datasets into SNP files. This is a custom file format used for further processing but is unable to be opened by a regular text editor. Biologically, an SNP is a single nucleotide polymorphism and is the most common type of genetic variation for humans [13]. Each SNP represents a difference in a single base, for example, an SNP may replace cytosine (C) with thymine (T) in a certain stretch of DNA [13]. The SNP files generated by RAiSD is an SNP vector, which is an entire alignment column [2]. When loaded into Python, a three-dimensional matrix is created and the base pair target position of the SNP file. The three-dimensional matrix can be separated into three two-dimensional matrices. The first two are identical and are used by RAiSD-AI for verification and the last for classification. The contents of the first two matrices represent rows of integers, while the last matrix represents rows of binary.

Previously, we mentioned the use of a sliding window algorithm. A window is simply a section of data, a sliding window will move its focus over the data. When it moves, it discards the data it previously kept and adds the new data it will focus on to the window [6]. In the case of RAiSD, the sliding window moves over the datasets to create SNP files which are then passed to a CNN model for classification.

In each dataset, there are multiple versions of DNA, these are referred to as simulations. An image of such simulation is a part of it, i.e. a window. A simulation can have multiple images, while a stride determines how far each image is from each other. For example, if an image starts at index 0 and with a stride of 5, the next image would start at index 5.

Lastly, a CNN model, or Convolutional Neural Network model, is a deep learning model that can be used to identify spatial patterns in a robust manner [1]. CNNs are often used for computer vision and classifying images. RAiSD has two different versions, one named SweepNET [22] and the other FastNN [25]. As the name suggests, FastNN is significantly faster than SweepNET but fails when running window sizes smaller than 50. Therefore, this research utilises a combination of both SweepNET and FastNN where SweepNET is used for the smaller than 50 window sizes.

3 EXISTING SOLUTIONS

Research to detect selective sweeps has been carried out for the past couple of decades. As a result, several methods have been developed, such as SweepFinder, SweeD, OmegaPlus and iHS [17].

SweepFinder is a program that implements a likelihood-based method to detect selective sweeps [8]. It performs a composite likelihood ratio test for positive selection [12] where the likelihood of the null hypothesis is calculated from the neutral frequency spectrum and the likelihood of the alternative hypothesis is modelled in a way where the neutral spectrum was altered by a recent selective sweep [8]. SweepFinder2 improves upon this by increasing its sensitivity and robustness to the effects of mutation rate variation and background selection [8]. However, SweepFinder2 suffers from increased computational and memory requirements, which took 5 hours to scan 1000 simulated datasets [2].

Similarly, OmegaPlus suffers from this, where it fails to scan the first chromosome of the human genome [2]. OmegaPlus is a tool to detect selective sweeps based on linkage distribution [3]. OmegaPlus implements the ω statistic which can be applied to the whole genome [3]. The ω statistic is calculated by analysing the ratio of genetic diversity in two regions, one where there is a known sweep and the other in a region without one [11]. This statistic can be used to accurately localise selective sweeps [11].

As described by Pavlidis et al. [2013], SweeD, or Sweep Detector, is a tool to quickly detect selective sweeps in whole genomes by building upon SweepFinder and analysing site frequency spectra [18]. It does this by calculating the Site Frequency Spectrum, SFS, analytically for demographic models that comprise of a number of population size changes and, optionally, an exponential growth as the most recent event [18].

Another sweep detection tool, named ASDEC, a neural-network-based framework that can scan whole genomes for selective sweeps [24]. It employs a CNN consisting of three layers, a a convolutional layer, a pooling layer and a dense layer [24]. ASDEC achieves similar accuracy as other convolutional neural network-based classifiers that rely on summary statistics, but is much faster [24]. ASDEC was used to scan the chromosome of the Yoruba population, identifying nine known candidate genes [24].

Efforts have also been made to improve the pre-processing of data, as described by Zhao and Alachiotis [2023], where they use a series of pixel-rearrangement algorithms to increase CNN classification accuracy for selective sweep detection [21]. From this, an increase of about 24.55% in accuracy was observed, compared to the pre-processing techniques previously used [21].

4 METHOD

This section will focus on the Minimisation method, aptly named as it focuses on dividing windows into smaller and smaller pieces to enhance the accuracy of RAiSD-AI. For the purposes of this research, we chose windows of sizes: 1000, 750, 500, 200, 100, 50 and 10. These were arbitrarily chosen, except window size 10 as this is the smallest window that RAiSD-AI can support without any errors. Before Minimisation can begin, we pre-trained models based on these window sizes to decrease the total time needed to run. These models were trained on 20 images per simulation for each dataset.

The first step necessary is to generate the starting windows of sizes of 1000. This is done by utilising RAiSD's scan function on one of the datasets mentioned in section 2. After this is complete, the result is moved into a folder and renamed to replicate the input RAiSD expects.

After this, the iterative process can begin. First, the folder of the previous window size is tested using RAiSD-AI. The results from which are deposited into two text files. First, named, 'PredResult-sneturalTE', containing the results for the neutral SNP files and the

other, 'PredResultssweepTE', containing the results for the sweep SNP files. In both text files, the name of the SNP file and its score is displayed. These text files are loaded into Python and then sorted, keeping only the top X% of files for the sweep SNP files and the bottom X% scores for the neutral SNP files. A more detailed explanation can be found in section 4.2. From here on, when the top X% is mentioned, we mean the top X% for sweep SNP files and the bottom X% for the neutral files.

Next, if this iteration is not the last in the list, in other words, the size is not 10, each SNP file which was retained from the sorting is divided into smaller windows, as described in section 4.3. These smaller windows are added to a new folder which also replicates the input required by RAiSD-AI. The iterative process then continues using this new folder with the smaller window sizes.

Conversely, if the iteration does reach the last size, 10, the process stops and passes the SNP file names to calculate the base pair positions, as detailed in section 4.4.

4.1 SNP File Names

Firstly, it is important to explain the naming used by RAiSD and subsequently, Minimisation. Each SNP file's name generated by RAiSD has the format such as, '400_6_0.snp'. The first integer, '400', represents the simulation the SNP file belongs to, the second represents the image and the last integer is redundant to our method. When new windows are created by our method, it appends the image number to this name. For example, if we create two images from the above file, they would now be called '400_6_0_0' and '400_6_0_1'. Additionally, the size of the window is added to this name, 750 in this case as the file above would be of size 1000, therefore the final name for the first image would be '400_6_0_0_750.snp'. Some placeholder should be here, as in section 4.2 it is vital this placeholder exists. The window size was used here, however, when creating new windows, this integer is removed before adding the image number to the new file name.

4.2 Sorting SNPs

The goal of sorting through the SNP files is to take the top X% of the child (smaller) windows with the best scores generated from RAiSD from the parent (larger) windows. This is done to decrease the computation time of the method and attempt to fulfil Sub-Research Question 1. It would be incorrect to simply take the top X% of the entire data as this would neglect some simulations with lower scores.

First, the file names are loaded into Python, using the 'PredResultsneturalTE' or 'PredResultssweepTE' files. Then, iterating over the SNP filenames, the filename is split into an array based on the '_' character. This leaves us with the name of the larger file. Consider the example in section 4.1, '400_6_0_0_750.snp' and '400_6_0_1_-750.snp' were created from the file, '400_6_0.snp'. By removing the last two indices, we see both of the child windows have the same name as well, which is,'400_6_0'. After deriving the parent window's name, we subset the files beginning with this name and then take the top X% of these if we loaded the names from the 'PredResultssweepTE' or bottom X% if the 'PredResultsneutralTE' was loaded. We round up this calculation, so there must be at least 1 file. It is then joined into a list with the other results. A simple formula for this would be $\lceil S * X \rceil$ where S is the subset of files beginning with the same name and X is the top X% represented in decimal format.

4.3 Creating New Windows

The first step in creating a new window is to remove any padding that was added in a previous iteration, more details are in section 4.3.1. We need padding to ensure that the size of each window is the same due to the requirements of the CNN model. There is a possibility we could omit padding by overlapping the windows, however, that would be biased to the data in the centre of the window as the edges of the window will not be processed as much. After this fist step, the necessary padding is calculated and then added to the existing window. This may seem counter-intuitive, to remove and then add padding but there could be a case where a large amount of padding has been added to the window already and by removing this, the next window size may only need a much smaller amount of padding. For example, in Figure 1, the padding gets removed from the initial window (1), which results in a window size smaller than the intended child window size (2). However, if we retained the initial padding, a new window would span mostly padding and its score from RAiSD-AI would suffer. For further explanation, please read section 4.3.1.

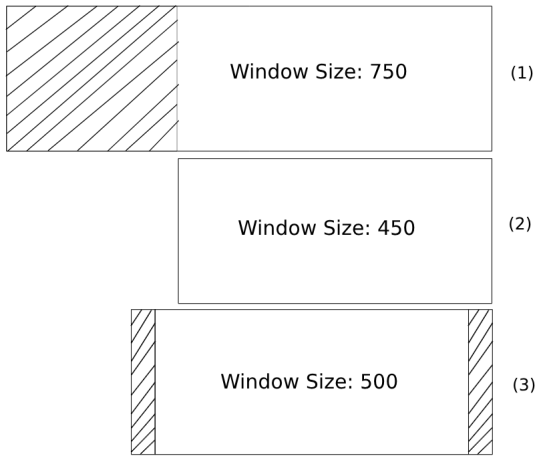

Fig. 1. Depiction of padding being removed and then added again. The hashed area represents the padding.

Next, we simply iterate i times over the parent window, where i is defined as: parent window size (O) divided by the stride (S) rounded up => $\lceil O/S \rceil = i$. Then the matrix is partitioned based on the third index of the parent window using the formula: $i * S$ for the start of the smaller window and $i * S + N$ for the end of the window, where N represents the intended child window size. As discussed in section 2, when an SNP file is loaded into Python, a 3d matrix is created. The 3d matrix can be thought of as three 2d matrices. Therefore, the first index of the 3d matrix represents a 2d matrix, the second index represents the rows of that matrix and the last index represents the columns. As we wish to partition based on the columns, we use the last index. If a partition is less than the size of the new window, the

iterative process is broken and the new windows are moved to the proper folder to be used in the next iteration of Minimisation.

The stride here is a whole number but it is dynamically calculated for each window size iteration. The input stride is a percentage and using this, the stride in the above calculation is calculated. For example, if the stride is set to 15%. when creating windows of size 750 the stride is 112.5, rounded up to 113. The formula for this would be: $\lceil s * N \rceil$ where s is the percent stride in decimal format, and N is the new window size.

4.3.1 Padding. Padding should be added equally to both the front and end of the window. This ensures that during the creation of new windows, there is an equal spread of padding and data. If all padding was added to the front or end, the result of that window would be lower in comparison to the other windows. From here, when 'matrix' is referred to, we mean the first 2d matrix of the window. When referring to adding padding to the window, we mean each of the 2d matrices has padding added to them, based on the columns. For example, after adding 500 padding, the shape of the 1000 matrix will transform from (3,128,1000) to (3,128,1500).

To calculate the padding, first removing any padding from previous iterations as described in section 4.3 and seen in Figure 1 step (1), we must then check if the length of the matrix is less than the intended size of the child window. If it is, we add padding to the front and end of the window equal to $(N - C)/2$ where C is the current size of the matrix, as shown in step (3) in Figure 1. After, if the stride is less than the intended child window size, padding equal to $(O\%C) + S$ is added to both sides of the window. Otherwise, we calculate the padding using the formula:

$$\frac{(\lceil (O/S) \rceil * C - O) \mod N}{2}$$

This gives the padding needed to be added per side. After this is complete, the slicing of the window can commence, as described in section 4.3.

4.4 Calculating Base Pair Positions

After Minimisation is complete, we can proceed to calculate the base pair positions of the smallest windows. Both the original window, of size 1000, and the smallest window, size 10, are loaded into Python and we will proceed to locate the index of the smallest window in the original, size 1000, window. First, any padding in the smaller window is removed, and then the first integer in its matrix is used to locate the indices where it occurs in the larger window, as shown in Figure 2. For example, if the first integer in the first 2d matrix of the smaller window is 127, we locate all indices in the larger windows 2d matrix where 127 is located. Then we slice smaller sections of the larger window using these indices as the staring point and the length of these subsections equal to the length of the small window. This can be seen in Figure 2, where the subsections from the Window of 1000 are called Copy Size. Then we check if these pieces are equal to the original window size 10, if they are their indices are saved, otherwise they are discarded. This approach is called Pattern Matching. We save all indices found and leave it up to the user to determine which is the correct position, an alternative method is described in section 6.1 which should locate the exact base pair position.

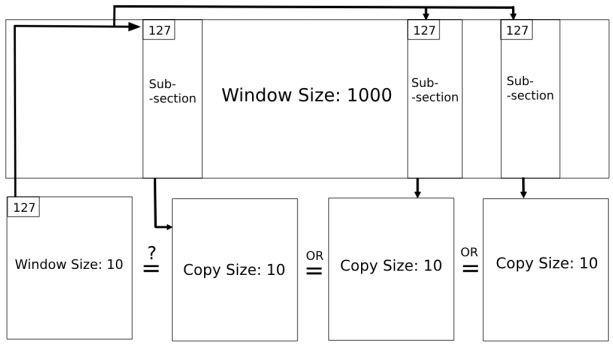

Fig. 2. Diagram depicting the location and slicing of smaller window sizes from the large window.

When the large windows are generated, they each have a target position. This target position is at its centre, therefore at index 500. Using the saved indices of the smaller window, we can calculate how far away the smaller window is from the centre using $i - 500$ where i is the index of the smaller window. If this value is negative, it means the smaller window is to the left of the centre, and conversely, if it is positive, the smaller window is located to the right of the centre.

Next, we must load the positions saved in the datasets. This is done by scanning the original dataset file for lines starting with 'position' and loading this line into a list. The result is a matrix, where the rows represent the simulation and the columns the base pair position values. We first find the list belonging to the large windows simulation by using the first integer in the large windows name. For example, 400 is the simulation for the large window as described in section 4.1. Next, we must search this list for the index of the target position, which we will call T. To find the distance of the smaller window to this target position we add them, $T + (i - 500)$. Using this value we find the value in the list located at this new index. Lastly, we must multiply this by the size of the region in base

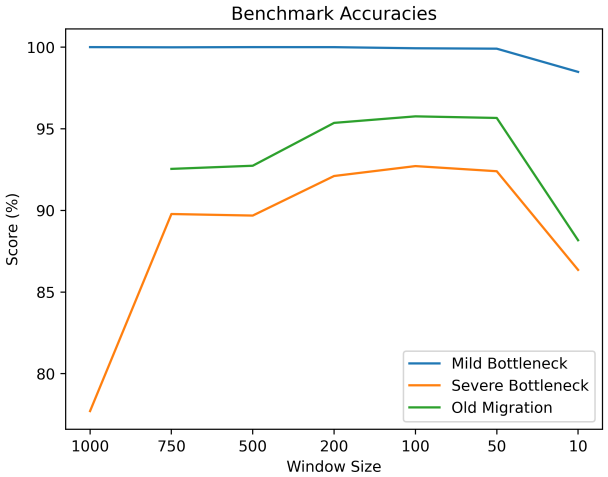

Fig. 3. Benchmark accuracies for each dataset.

pairs, which is usually set to 100000 by RAiSD-AI and was also set to this value in this research. To better understand this, the following formula is used:

$$P[T + (i - 500)] * L$$

Where P is the list of base pair positions of the corresponding simulation, T is the index of the target position, i is the index of the smaller window and L is the size of the region in base pairs.

This calculation is done for all possible indices found in the pattern matching and then saved to a text file.

5 RESULTS AND DISCUSSION

In this section, the results of the research will be shown and discussed. In order to assess if the Minimisation method improves the accuracy of RAiSD-AI, we must first run a benchmark using the same window widths. Naturally, the higher the accuracy, the better the result.

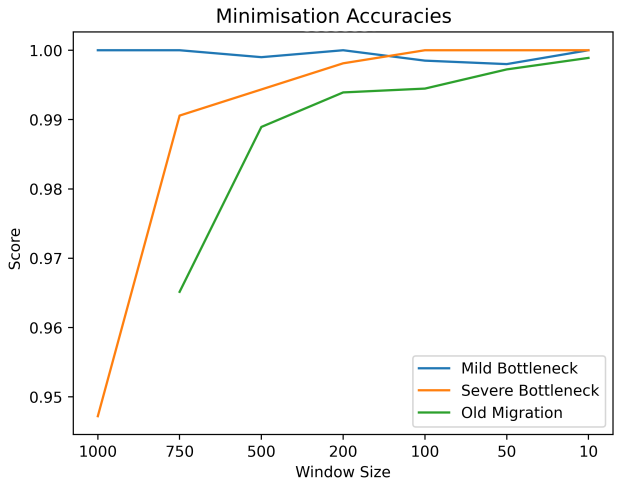

Fig. 4. Accuracies produced by the Minimisation Method for each dataset.

5.1 Benchmark

To create these results, the models, as used in the Minimisation method, were also used to assess RAiSD-AI's accuracy without the method. As described at the beginning of section 4, these models were trained on simulations with 20 images each. Similarly, the testing data also contains 20 images per simulation. The values as seen in Figure 3 is the output accuracy of RAiSD-AI. These tests were run independently of one another and the score of one does not affect the score of another.

It is important to note that the Old Migration dataset is not large enough to encompass the initial 1000 window, therefore this has been omitted and it starts at index 750 instead.

As shown in Figure 3, we see RAiSD-Ai performs very well with the Mild Bottleneck Dataset. For the other two datasets, the smaller window sizes between, and including, 200 and 50 seem to perform better than their larger counterparts. There is a noticeable dip in accuracy for all three datasets at window size 10, which can likely be explained by the transition from FastNN to SweepNET as SweepNET is only used for this smallest window size. It seems that even with several images, RAiSD-AI can struggle with more difficult datasets. the highest accuracy for the Old Migration dataset being about 96% at window size 50 and at the same size, roughly 92% accuracy for the Severe Bottleneck dataset.

5.2 Minimisation Results

Here the results of applying the Minimisation results are shown. These accuracy scores are calculated by taking the maximum score of each simulation per window size iteration after sorting the SNP files, as described in 4.2. Next, we assess if the model predicts the window to have a sweep or not by judging if the score of the window is > 0.5. By using the ground truth of these files, we calculate the accuracies per iteration. The stride percentage for the results found in Figure 4 is 15% and it takes the top 5% of SNP files per iteration.

At first glance, all three datasets seem to perform well when the Minimisation method is applied to them. The most surprising is that the most difficult dataset, the Severe Bottleneck, seems to perform better than the other two at some window sizes. Surprisingly, the Mild Bottleneck seems to suffer slightly as Minimisation occurs.

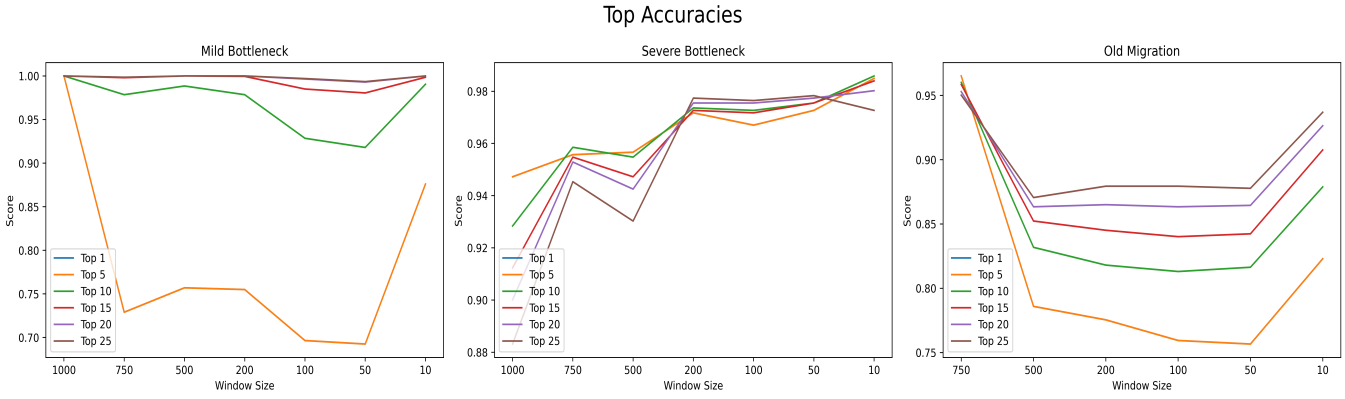

Fig. 5. Accuracies for each dataset with varying top X percentages, using stride of 100%.

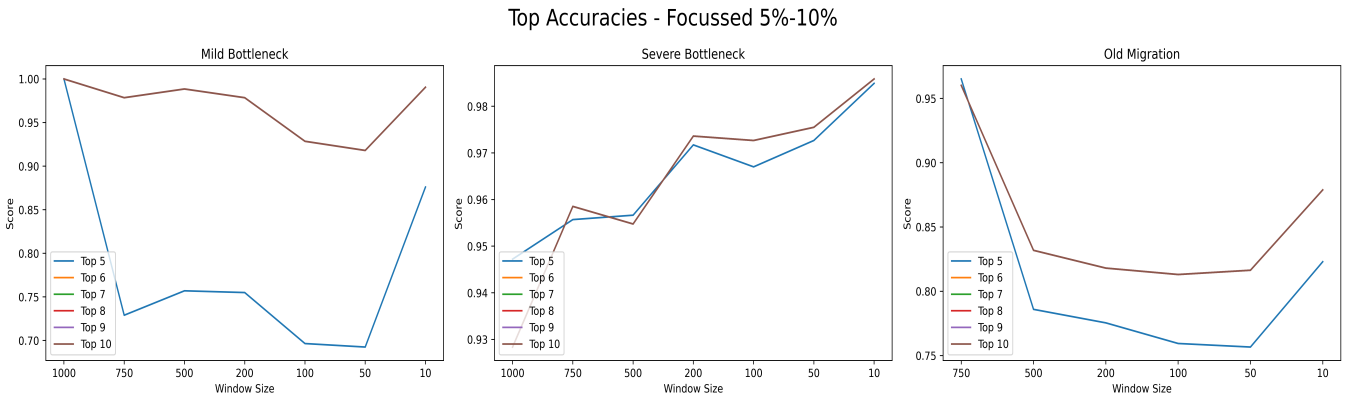

Fig. 6. Accuracies for each dataset focusing on 5%-10%.

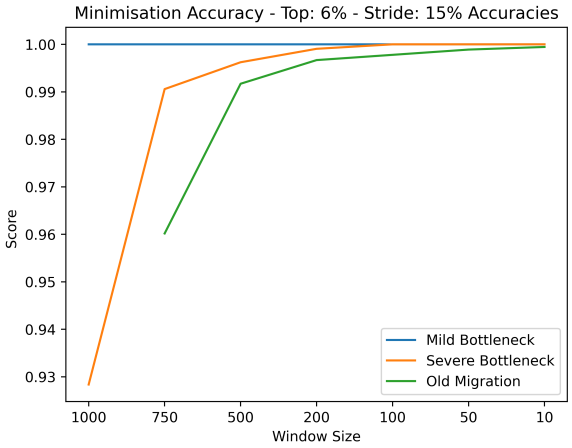

Fig. 7. Accuracy of each dataset with Minimisation applied with a X% of 6% and stride of 15%.

Dips can be seen at window size of 500, 100 and 50 before improving again at window size 10. This may be due to the noise introduced by the padding. While not all datasets managed to reach 100% accuracy, this could be achieved with a more fine grained stride or a different top percentage. Alternatively, adjusting some other parameters such as the top percentage, as discussed in section 5.3, or adjusting the starting window, as described in section 5.4.

5.3 Top Percentage

In this section, we test the Minimisation method with various top X percentages. The stride percent for these experiments was set to 100, therefore the output varies slightly from section 5.2.

The results for this experiment, as shown in Figure 5, have more variation than expected. Firstly, the Severe Bottleneck has a much different result than the other two, In the sense that the top percent does not make too much of a difference. This is especially the case for window sizes of 200 and 50. This may explain the stronger performance of the Severe Bottleneck seen in Figure 4. For all three datasets, the Top 1% line can not be seen. Upon investigation of the output values, it shows that the Top 1% and Top 5% have the same accuracy scores. This is likely due to the rounding used, for example $20 * 0.01 = 0.2$, rounded up to 1, which is the same as $20 * 0.05 = 1$, where 20 is the number of files with the same parent window. Surprisingly, especially for the Mild Bottleneck, the difference between the Top 5% and 10% lines is quite substantial. In order to confirm this as the reason, Figure 6 has been created which focuses on the percentages 5 to 10.

Once again, there are only two lines visible from this experiment. After investigating the output values, it determines that the split is between the Top 5% line and all five others are joined. That is, percentages from 6 to 10 all have the same accuracy scores. However, it is still surprising that at the later window sizes (100, 50, 10), there is no deviation. At those later iterations, there are many more files than it started with, therefore we should expect some variation between the top X% at this point.

After adapting our initial parameters used in Figure 4, we see some considerable changes, as shown in Figure 7. Firstly the Mild Bottleneck does not vary and maintains a constant 100% accuracy, unlike the results shown in Figure 4, where it would drop to about 99.8% accuracy at window size 100 and 99.9% for window size 50. While the Old Migration still does not reach an accuracy of 100%. it gets even closer to having a final accuracy of 99.9% in Figure 7. As for the Severe Bottleneck, despite the worse accuracy at window size 1000, starting at 92.9% in 7, in comparison to Figure 4, starting at 99.8%, it still manages to reach an accuracy score of 100% at the same window size of 100 as in Figure 4.

5.4 Starting Window

Here are the results of starting the Minimisation method with various window sizes. It uses this new starting window to scan the data and then proceed with the remaining window sizes. Until now, it has always been started with a window size of 1000. In Figure 8, it shows the result of this experiment, the other parameters for these results are the Top 1% and a stride of 100%.

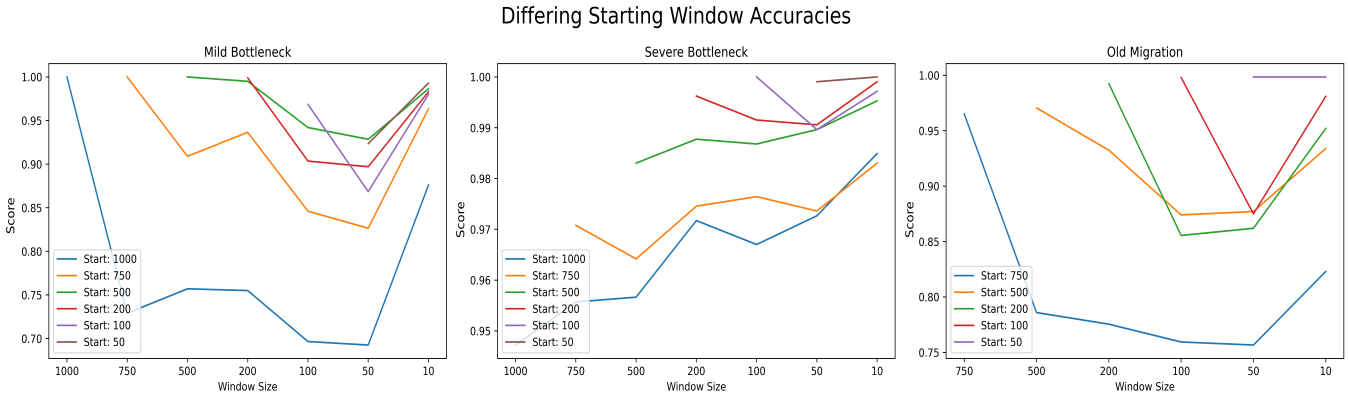

Fig. 8. Accuracies for each dataset starting with various window sizes.

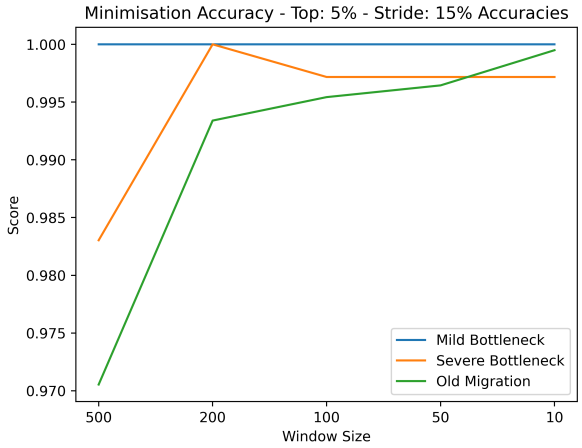

Fig. 9. Accuracy of each dataset with Minimisation applied with a X% of 5% and stride of 15%, starting at window size 500 for each dataset.

It is immediately evident that a smaller window size is preferred. Most notably the large differences between window size 1000 and 750 for the Mild Bottleneck, as well as window size 750 and 500 for the Old Migration dataset. Surprisingly, this does not seem to be the case for the Severe Bottleneck, While there is some difference between sizes of 1000 and 750, they seem to have roughly the same accuracy when they finish.

When applying a window size to the datasets with the same parameters as Figure 4, it reveals some interesting variations. Like Figure 7, the Old Migration dataset does relatively well, ending with a 99.9% accuracy. Here it seems to outperform Figure 7 at each window size. Similarly, the Mild Bottleneck dataset does not deviate from 100%. The most interesting case, the Severe Bottleneck, seems to have done worse than the previous experiments, failing to reach a 100% end accuracy. It seems that for more complex datasets, a starting wider starting window is preferred. This may explain why the Old Migration does note fare as well as the others, as the starting window is not the maximum for this dataset.

6 CONCLUSION

The goal of this paper was to use increasingly smaller window sizes in order to more accurately determine the location of selective sweeps, so that a wider scope may be used by sweep detection tools without losing accuracy or precision. Throughout this paper, we have not only discussed the pre-existing solutions to detect selective sweeps but also the details of the Minimisation method which attempts to achieve this goal.

The main research question of this paper was to investigate the effect of incrementally decreasing the window size compared to a fixed window size. As we have seen, the Minimisation method fulfills this requirement and does does seem to be an improvement in its accuracy. As shown in Figure 4, the accuracies for all three datasets did improve to almost 100% for all three. While with varying parameters; the accuracy can worsen during the computation of the Minimisation method, as shown in Figure 4. However, a general improvement in accuracy can be seen, as even after the first sorting has taken place, the Minimisation method does achieve a higher accuracy than the benchmark. This can be seen in Figure 3, window size 1000 and the same window size in Figure 4.

The second research question was to investigate while top X% provides the highest accuracy. From the figures shown in section 5.3, it is generally better to use a higher percentage, especially when a large stride is used as used for Figures 5 and 7. However, this does lead to a longer computation time as the Minimisation method must create significantly more windows, especially if a small stride is introduced. Due to the rounding used when taking the top X%, it seems that every 5% increases in this parameter only changes the accuracy scores, that is percentages 1 to 5 will have the same accuracy scores and 6 to 10 will have the same accuracy scores.

Lastly, the last research question was to investigate which was the most effective initial window size. To determine what the most effective initial window size seems to be a complex problem. As seen in section 5.4, the smaller window sizes seem to perform better than their larger counterparts. As shown in Figure 9, a smaller window

size is preferred for less complex datasets, whereas a wider window is preferred for more complex ones. For future researchers, they may want to keep a large window size as their initial starting window size as other parameters, such as the stride and top X%, can be used to increase the accuracy, as seen in Figure 5 for the top X% and Figures 5.2 and 7 for a smaller stride. In the cases of the Mild and Severe Bottleneck datasets, their lengths were enough to satisfy an initial window size of 1000, but future researchers should adjust this to a size almost equal to their datasets. Some margin should be left in order to take multiple images of the dataset, with some stride between each new image.

6.1 Future Work

There are several ways in which the Minimisation method could be improved. Firstly, a stride of 1 would give an extremely fine grained search throughout the entire window, unfortunately, this is not possible with the current implementation of Minimisation. While there is no limit on the quantity of the input, Minimisation calculates the entirety of one iteration and then proceeds with the next. Therefore, a sliding window algorithm could improve Minimisation's accuracy further.

While optimisation was taken into account during the development of the Minimisation method, it can still take considerable computation time. Therefore, it would benefit from further research into the optimisation of this method. From observation during this research, the most time occurs during the creation of the windows, hence efforts should be focused here.

Furthermore, some variations in accuracy from the Minimisation method was observed, such as in Figure 5, despite the Mild Bottleneck starting at 100% at window size 1000, the accuracy drops in the next iteration. This is not limited to the Mild Bottleneck dataset, as the Old Migration's accuracy in Figure 5, there is little variation between window sizes 500 and 50. Some investigation should be done to understand this using explainable AI tools.

Lastly, as mentioned in section 4.4, there is a way to calculate the exact base pair positions. This could be done by utilising the naming scheme utilised by the Minimisation method. As described in section 4.1, the image of each window is added to the name. For example the first is labelled as 0, the second 1 etc. Using these, the calculation of the index of the window is possible. Then, using the formula, as explained in section 4.4, the calculation of the exact base pair position of the window would be possible. The stride of the window must be taken into account, as well as the padding added when creating the child windows.

ACKNOWLEDGMENTS

Special thanks to Sjoerd van den Belt for his help during this research by answering questions about RAiSD-AI and pointing out mistakes in the methodology as it was implemented.

REFERENCES

[1] Samit Ahlawat. 2022. *Reinforcement learning for finance* (1 ed.). APress, Berlin, Germany. 139–140 pages.

[2] Nikolaos Alachiotis and Pavlos Pavlidis. 2018. RAiSD detects positive selection based on multiple signatures of a selective sweep and SNP vectors. *Communications. Biology.* 1, 1 (June 2018), 79. https://doi.org/10.1038/s42003-018-0085-8

[3] Nikolaos Alachiotis, Alexandros Stamatakis, and Pavlos Pavlidis. 2012. OmegaPlus: a scalable tool for rapid detection of selective sweeps in whole-genome datasets. *Bioinformatics* 28, 17 (Sept. 2012), 2274–2275. https://doi.org/10.1093/bioinformatics/bts419

[4] Bruce Alberts, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts, and Peter Walter. 2002. *The structure and function of DNA.* Garland Science, London, England.

[5] Maddox B. 2002. *Rosalind Franklin: The Dark Lady of DNA.* HarperCollins.

[6] Vladimir Braverman. 2015. Sliding window algorithms. In *Encyclopedia of Algorithms.* Springer Berlin Heidelberg, Berlin, Heidelberg, 1–6. https://doi.org/10.1007/978-1-4939-2864-4_797

[7] Janine E Deakin, Sally Potter, Rachel O'Neill, Aurora Ruiz-Herrera, Marcelo B Cioffi, Mark D B Eldridge, Kichi Fukui, Jennifer A Marshall Graves, Darren Griffin, Frank Grutzner, Lukáš Kratochvíl, Ikuo Miura, Michail Rovatsos, Kornsorn Srikulnath, Erik Wapstra, and Tariq Ezaz. 2019. Chromosomics: Bridging the gap between genomes and chromosomes. *Genes (Basel)* 10, 8 (Aug. 2019), 627. https://doi.org/10.3390/genes10080627

[8] Michael DeGiorgio, Christian D Huber, Melissa J Hubisz, Ines Hellmann, and Rasmus Nielsen. 2016. SweepFinder2: increased sensitivity, robustness and flexibility. *Bioinformatics* 32, 12 (June 2016), 1895–1897. https://doi.org/10.1093/bioinformatics/btw051

[9] Nicole Gleichmann. 2020. Gene vs allele: Definition, difference and comparison. https://www.technologynetworks.com/neuroscience/articles/gene-vs-allele-definition-difference-and-comparison-331835. Accessed: 2024-6-17.

[10] Balwinder Kaur and Rajendra Srivastava. 2014. Synthesis of ionic liquids coated nanocrystalline zeolite materials and their application in the simultaneous determination of adenine, cytosine, guanine, and thymine. *Electrochim. Acta* 133 (July 2014), 428–439. https://doi.org/10.1016/j.electacta.2014.04.019

[11] Yuseob Kim and Rasmus Nielsen. 2004. Linkage disequilibrium as a signature of selective sweeps. *Genetics* 167, 3 (July 2004), 1513–1524. https://doi.org/10.1534/genetics.103.025387

[12] Yuseob Kim and Wolfgang Stephan. 2002. Detecting a local signature of genetic hitchhiking along a recombining chromosome. *Genetics* 160, 2 (Feb. 2002), 765–777. https://doi.org/10.1093/genetics/160.2.765

[13] MedlinePlus. 2020. Noonan Syndrome; Genomic Research: SNP. https://medlineplus.gov/genetics/condition/noonan-syndrome/ and https://medlineplus.gov/genetics/understanding/genomicresearch/snp/. Noonan Syndrome: [updated 2020 Jun 18; reviewed 2018 Jun 01; cited 2020 Jul 1; about 5 p.], Accessed: 2024-6-17.

[14] Tomoko Ohta. 1996. The current significance and standing of neutral and neutral theories. *Bioessays* 18, 8 (Aug. 1996), 673–7; discussion 683. https://doi.org/10.1002/bies.950180811

[15] Manjit Panigrahi, Divya Rajawat, Sonali Sonejita Nayak, Kanika Ghildiyal, Anurodh Sharma, Karan Jain, Chuzhao Lei, Bharat Bhushan, Bishnu Prasad Mishra, and Triveni Dutt. 2023. Landmarks in the history of selective sweeps. *Anim. Genet.* 54, 6 (Dec. 2023), 667–688. https://doi.org/10.1111/age.13355

[16] Isha Pathak and Bruno Bordoni. 2024. *Genetics, Chromosomes.* StatPearls Publishing, Treasure Island (FL). https://www.ncbi.nlm.nih.gov/books/NBK557784/ In: StatPearls [Internet]. 2024 Jan. PMID: 32491716.

[17] Pavlos Pavlidis and Nikolaos Alachiotis. 2017. A survey of methods and tools to detect recent and strong positive selection. *J. Biol. Res. (Thessalon.)* 24, 1 (Dec. 2017), 7. https://doi.org/10.1186/s40709-017-0064-0

[18] Pavlos Pavlidis, Daniel Živkovic, Alexandros Stamatakis, and Nikolaos Alachiotis. 2013. SweeD: likelihood-based detection of selective sweeps in thousands of genomes. *Mol. Biol. Evol.* 30, 9 (Sept. 2013), 2224–2234. https://doi.org/10.1093/molbev/mst112

[19] Sjoerd van den Belt, Hanqing Zhao, and Nikolaos Alachiotis. 2024. Scalable CNN-based classification of selective sweeps using derived allele frequencies. *Bioinformatics* (2024). https://doi.org/10.1093/bioinformatics/btae385 In press.

[20] Tanita Wein and Tal Dagan. 2019. The effect of population bottleneck size and selective regime on genetic diversity and evolvability in bacteria. *Genome Biol. Evol.* 11, 11 (Nov. 2019), 3283–3290. https://doi.org/10.1093/gbe/evz243

[21] Hanqing Zhao and Nikolaos Alachiotis. 2023. Effective data preprocessing techniques for CNN-based selective sweep detection. (Dec. 2023). https://doi.org/10.1109/bibm58861.2023.10385303

[22] Hanqing Zhao, Pavlos Pavlidis, and Nikolaos Alachiotis. 2023. SweepNet: A Lightweight CNN Architecture for the Classification of Adaptive Genomic Regions. In *Proceedings of the Platform for Advanced Scientific Computing Conference* (Davos, Switzerland) *(PASC '23).* Association for Computing Machinery, New York, NY, USA, Article 12, 10 pages. https://doi.org/10.1145/3592979.3593411

[23] Hanqing Zhao, Matthijs Souilljee, Pavlos Pavlidis, and Nikolaos Alachiotis. 2023. Genome-wide scans for selective sweeps using convolutional neural networks. *Bioinformatics* 39, 39 Suppl 1 (June 2023), i194–i203. https://doi.org/10.1093/bioinformatics/btad265

[24] Hanqing Zhao, Matthijs Souilljee, Pavlos Pavlidis, and Nikolaos Alachiotis. 2023. Genome-wide scans for selective sweeps using convolutional neural networks. *Bioinformatics* 39, 39 Suppl 1 (June 2023), i194–i203. https://doi.org/10.1093/

bioinformatics/btad265

[25] Qianqian Zhou, Nan Chen, and Siwei Lin. 2022. FASTNN: A deep learning approach for traffic flow prediction considering spatiotemporal features. *Sensors (Basel)* 22, 18 (Sept. 2022), 6921. https://doi.org/10.3390/s22186921

A APPENDIX A

During the preparation of this work, the author used ChatGPT in order to reverse engineer the pre-exiting SNP file loading function that was found in a Python file, belonging to PyTorch. The function generated by ChatGPT was tested to ensure that the files saved through the function could also be loaded by the pre-existing SNP file loading function. Aside from this, the text editor used was Overleaf and Grammarly was used to check any grammar mistakes in this paper. Lastly, bibtex.com's DOI to BibTeX converter was used to generate references in BibTeX format, which Overleaf then used to generate the reference list. After using these tools, the author reviewed and edited the content as needed and takes full responsibility for the content of the work.