Genetic algorithms with niche and parameters auto optimisation to find PNG interlaced image smaller than non-interlaced

Louis Villedieu
jod.li

Tuesday 26th February, 2019

Author Note

Louis Villedieu, jod.li .
This research was self-financed by the author.
Correspondence concerning this article should be addressed to louis@jod.li .

Abstract

Portable Network Graphics (PNG) image files usually have a larger size interlaced than not interlaced. This article explains how an online genetic algorithm has been used to find images for which the PNG interlaced version file is smaller in size than the PNG non-interlaced version file.

Keywords: Genetic algorithm, niche fitness sharing, parameters optimisation, distributed GA

Introduction

As explained in section “Interlaced PNG”, the purpose of the ADAM7 interlacing algorithm used for PNG images is to allow the display of images progressively even if all the data didn’t reach the image interpreter. This leads usually to files with a larger size for equivalent content. However, as explained in section “Finding Smaller Interlaced PNG”, a genetic algorithm made it possible to find images for which, the same image interlaced has a smaller size than its non-interlaced version. The details on this genetic algorithm are explained in section “Genetic Algorithm”. This genetic algorithm is distributed, it implements a niche mechanism based on Goldberg and Richardson (1987) as explained in section “Niche Fitness Sharing”, it implements some auto optimisation of the parameters as explained in section “Genetic Algorithm Parameters Auto Optimisation”.

Interlaced PNG

The ADAM7 interlacing algorithm, as explained in RFC2083, through a mechanism based on 7 passes, allows the progressive display of PNG images even if all the data was not received by the interpreter. This means that the PNG image interpreter can receive the PNG image data for example from a slow connection and display an approximation of the image without the remaining data. For example, if the data received contains only the 1st pass, the interpreter will at best display the image based only on the first pixel in the upper left corner of each 8x8 pixels square. The section “Size Difference Between Interlaced and Non-Interlaced PNG” explains the differences there could be in size between an interlaced PNG image file and a non-interlaced one.

Size Difference Between Interlaced and Non-Interlaced PNG

As explained in jod.li, based on RFC2083, the differences in data between two pixel-wise identical PNG images interlaced and non-interlaced are:

- In IHDR chunk: Interlace method (0 for non-interlaced, 1 (ADAM7) for interlaced). Which doesn’t lead to a file size difference by itself.
- In IDAT chunks (compressed and filtered pixels data): Compressed (deflated) scanlines starting with filter type containing the encoded image data. This usually changes the file size as explained below.

The uncompressed (inflated) data size can be calculated as follows for the cases when the pixel size is greater or equal to 8 bits.

Non-Interlaced Case

In the non-interlaced case, the uncompressed data length can be expressed as a function of the image height, width and bits per pixel (*bpp*).

$datalength = height * width * bpp/8 + height$

With:

$bpp = spp * bd$

Where:

- *spp* is the number of sample per pixel which depends on the colour type value as shown in table 1.
- *bd* is the bit depth of the image.

Note the ”+*height*” at the end of the *datalength* formula. This number is added because one filter encoded on a byte each is necessary for each pixel line of the image in the non-interlaced case.

Colour type	Colour type name	Sample per pixel (*spp*)
0	grayscale	1
2	RGB	3
3	palette based	1
4	grayscale with alpha	2
6	RGB with alpha	4

Table 1: Colour type to sample per pixel

Interlaced Case

In the interlaced case, the uncompressed data length can be expressed in a similar way as for the non-interlaced case above:

$datalength = height * width * bpp/8 + lines$

Where lines is the sum of the number of filters encoded in the file. There is one filter encoded for each line which depends on the interlacing pass.

$lines_{pass1} = ceil(heigh/8)$
$lines_{pass2} = (width > 4?CEIL(height/8) : 0)$
$lines_{pass3} = CEIL((height - 4)/8)$
$lines_{pass4} = (width > 2?CEIL(height/4) : 0)$
$lines_{pass5} = CEIL((height - 2)/4)$
$lines_{pass6} = (width > 1?CEIL(height/2) : 0)$
$lines_{pass7} = FLOOR(height/2)$

Size Difference

As shown in jod.li, the uncompressed data size of an interlaced image will almost always be greater than the one for the same image non-interlaced because of the number of filters to be encoded. There are special cases to this as it is possible for the number of filters to be the same for example in the case of a 1 pixel large image. We can easily show that the number of filters encoded for such an interlaced image with a pixel width will fulfill the following condition:

$filters_{interlaced-1pixel-width} = height$

Indeed, if such image has a height of for example 50, the number of filters will be:

$filters_{non-interlaced} = 50$
$filters_{interlaced} = 7+0+6+0+12+0+25 = 50$

So, a PNG interlaced will have the same size of uncompressed data as an equivalent non-interlaced PNG image in case they have a width of 1 pixel. As confirmed by the genetic algorithm findings in section “Finding Smaller Interlaced PNG”, it is the best we can get when looking for smaller interlaced compared to non-interlaced PNG images.

Other Reasons for Size Difference

As explained in stackoverflow, the size of pixel wise equivalent PNG images interlaced and non-interlaced can differ because of:

- Different non-pixel encoding related content embedded in the file such as palette (in the case of colour type =! 3) and non-critical chunks such as chromaticities, gamma, number of significant bits, default background colour, histogram, transparency, physical pixel dimensions, time, text, compressed text. Note that some of those non-pixel encoding related content can lead to different display of the image depending on the software used and the situation.
- Different compression related content such as better filtering choices, accidental lower data entropy/complexity due to interlacing.

The focus here is on the latter. The pixel uncompressed data can at best be of the same size between the interlaced and non-interlaced versions in the case of a one pixel wide PNG image. The way the interlaced image file can be smaller is by having data organised in a way more favourable for the compression after filtering.

Compare Images

As explained in stackoverflow, to know whether 2 PNG images describe the same pixels, one can rely on the ImageMagick program and use the following command:

```
diff <( convert noninterlaced.png rgba:- ) <( convert interlaced.png rgba:- )
```

This comparison has been used to validate that the interlaced and non-interlaced versions of the PNG image describe the same pixels.

Images Generation

The pixel data is written to a RGB file with bytes representing the red, green and blue values for each pixel. The same command is used to generate both images through ImageMagick with the exception of the interlace method as shown below:

Non-interlaced:

```
convert -depth 8 -size ${width}x${height} RGB:input.rgb -interlace None
PNG:noninterlaced.png
```

Interlaced:

```
convert -depth 8 -size ${width}x${height} RGB:input.rgb -interlace PNG
PNG:interlaced.png
```

Using those two commands ensures that the non-pixel related data is the same between the interlaced and non-interlaced versions of the PNG image.

Genetic Algorithm

In order to find the pixels and the image shape for which the interlaced PNG file has the smallest size compared to its non-interlaced PNG file equivalent, a genetic algorithm has been used.

The correspondence between the chromosome and the image is explained in section “Chromosome to Image”.

The fitness function is explained in section “Fitness Function”.

In order to minimise the time it takes to evolve the population, this genetic algorithm can be distributed as explained in section “Genetic Algorithm Distribution”.

In order to ensure and control the diversity of the population, the genetic algorithm uses a niche fitness sharing as explained in section “Niche Fitness Sharing”.

In order to find the most appropriate parameters for the population evolution, the genetic algorithm parameters can be auto optimised through a genetic algorithm as explained in section “Genetic Algorithm Parameters Auto Optimisation”.

The genetic algorithm and the functionalities described here are available as oga.jod.li .

Chromosome to Image

The chromosomes are encoded in bytes represented in hexadecimal strings. The first byte is the width in pixels of the image. The image height is deduced from the width and the chromosome length. The maximum chromosome length is 3000 bytes. The chunk size is of three bytes. The chunk allows to regroup bytes so that chromosome operations such as addition, deletion, crossover and swap are more meaningful as explained in section “Genetic Algorithm Features”.

Fitness Function

As explained in section “Images Generation” and section “Chromosome to Image”, the pixels and image shape are defined by the chromosomes. This allows the generation of two equivalent PNG image files, one interlaced and one non-interlaced. The fitness function is defined as follows:

$$fitness = 200 - size_{interlaced} + size_{non-interlaced}$$

The addition of 200 prevents negative fitness. A chromosome with a fitness higher than 200 thus has an interlaced PNG file size lower than a non-interlaced PNG file.

Genetic Algorithm Distribution

Genetic Algorithm Distribution Principle

Genetic algorithm can easily be distributed by sharing the chromosome fitness calculation amongst several clients. The goal pursued being a faster evolution. oga.jod.li is a way to distribute the evolution of genetic algorithm with the ability to spread the population through niche mechanisms to avoid local optima as explained in section “Niche Fitness Sharing” and the possibility to improve the speed of the evolution through auto optimisation of the genetic algorithm parameters (mutation, crossover, elitism, niche, chunks) through another genetic algorithm as explained in section “Genetic Algorithm Parameters Auto Optimisation”.

As shown in figure 1, the client device retrieves chromosomes to evaluate from the oga.jod.li server and sends back the fitness for the received chromosomes.

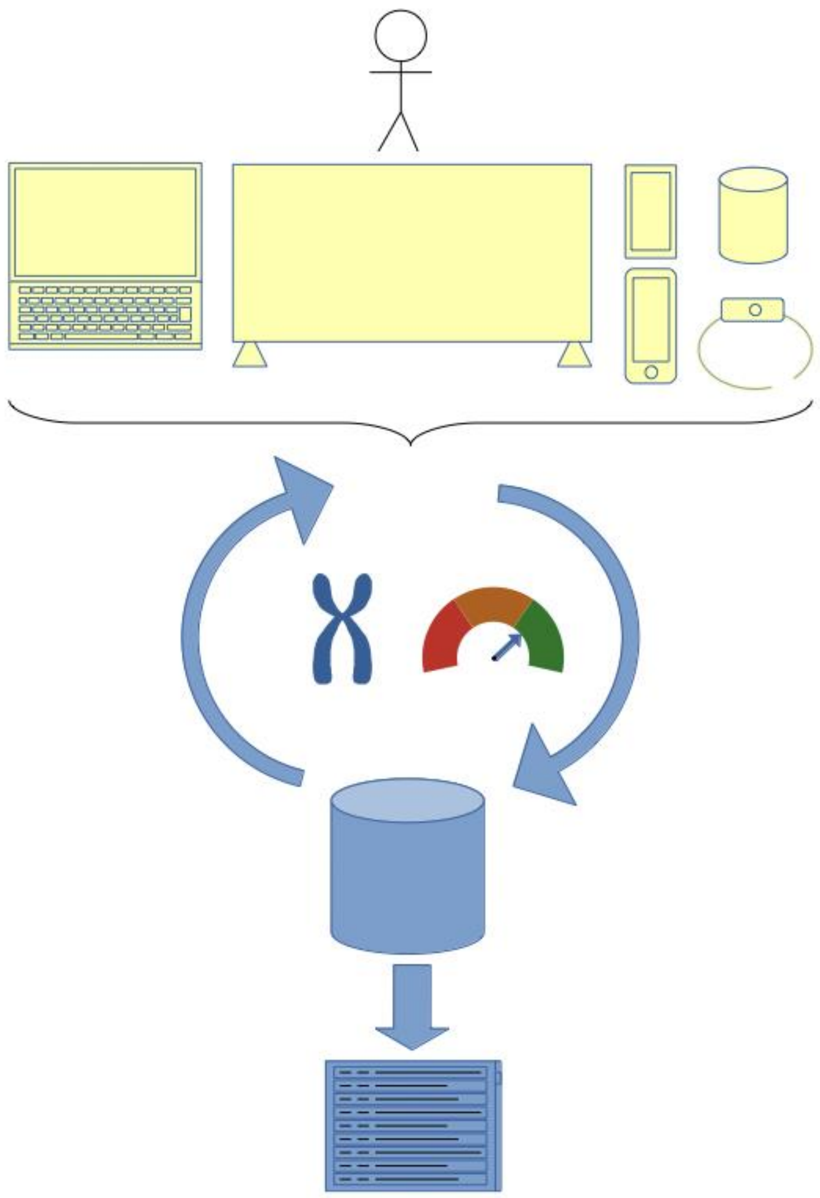
Figure 1: Distributed genetic algorithm evolution

The devices that can evaluate the chromosome fitness can be very diverse:

- Computer: general case. Can be used easily for example while looking for PNG files with lager size non-interlaced as it is the purpose of this paper.
- smart TV: for example to return performance of a chromosome if this is subject to the appreciation of someone using a TV. Could be for example part of a game where the chromosome would define the AI used for the game.
- Smartwatch: for example the chromosome could define the AI of a motivating app where the fitness would be measured depending on the effectiveness for the user.
- Smartphone: as for the smart TV, the chromosome could for example define the AI of a game.
- connected object: for example the chromosome could define the logic of a controller working in various unprecedented situations working with other connected objects.
- ...

The only requirement is the ability to call HTTPS urls to get chromosomes for evaluation from oga.jod.li server and to send the evaluated chromosome fitness back to the oga.jod.li server. The urls contain username and key parameters generated when creating a user either through OAuth2 login on Google, GitHub, stackoverflow, LinkedIn or by email validation.

Three parameters control the evolution distribution as explained below.

Chromosomes per batch parameter. Use a small number if you wish to use many devices at the same time. Use a large number if you have a poor connection.

Envelope parameter for the HTTPS responses which can be:

- JSON
- XML
- CSV

Encoding parameter for the chromosomes which can be:

- alnum [a-zA-Z0-9]
- alpha [a-zA-Z]
- lower [a-z]
- upper [A-Z]
- xdigit [A-F0-9] (hexadecimal)

Genetic Algorithm Features

As explained in section “Genetic Algorithm Parameters Auto Optimisation”, all the following genetic algorithm features correspond to parameters on https://en.oga.jod.li which can be auto optimised through a genetic algorithm. As those parameters require some randomness to be applied, it is not possible to predict exactly how much they affect the resulting chromosome. The illustrations represent the parent or parents that remain in the population as long as their fitness (shared or raw) allows, as well as the child generated through the explained operator.

Mutation

Mutation is controlled through two parameters.

The mutation rate parameter (0.01-0.50). The higher, the more the population will be randomly affected as illustrated in figure 2.

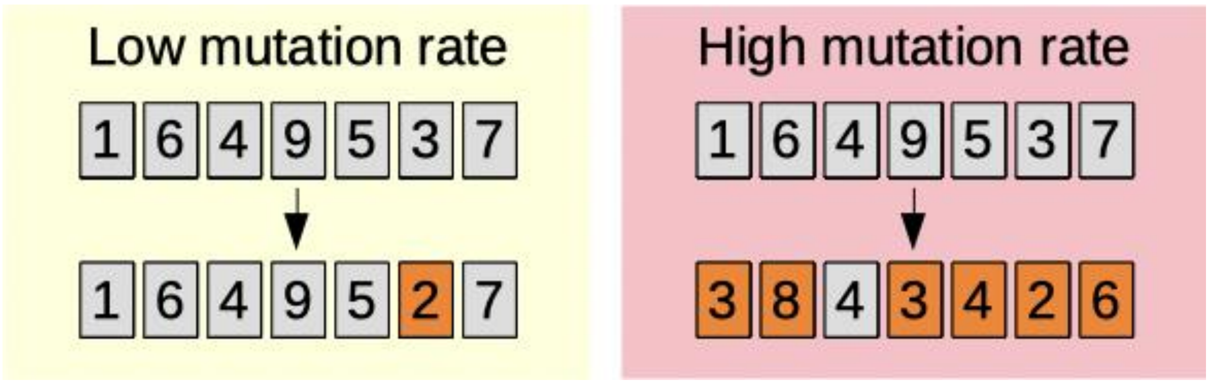

Figure 2: Mutation rate

The mutation proportion parameter (0.00-1.00). The higher, the more the mutated characters are changed as illustrated in figure 3.

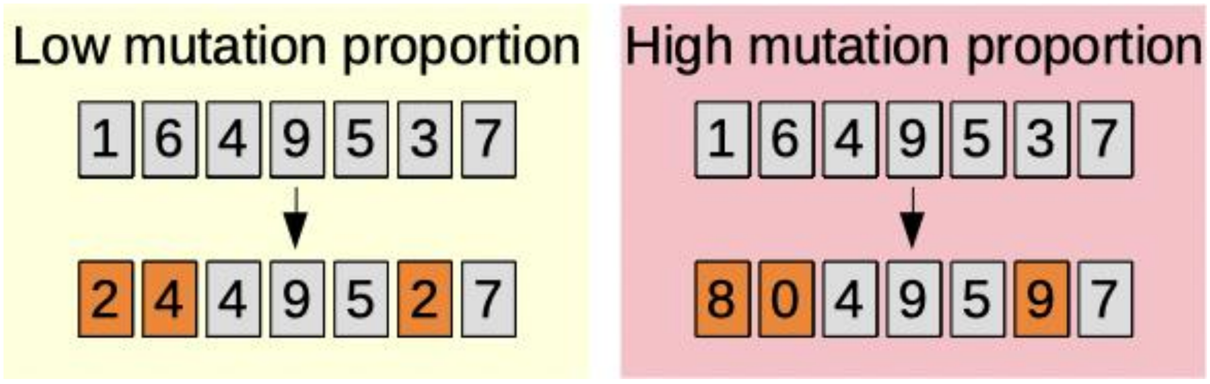

Figure 3: Mutation proportion

Crossover

Crossover is controlled through two parameters.

The crossover rate parameter (0.01-0.50). The higher, the more a chromosome is mixed with a second parent. More often and with more impact as illustrated in figure 4.

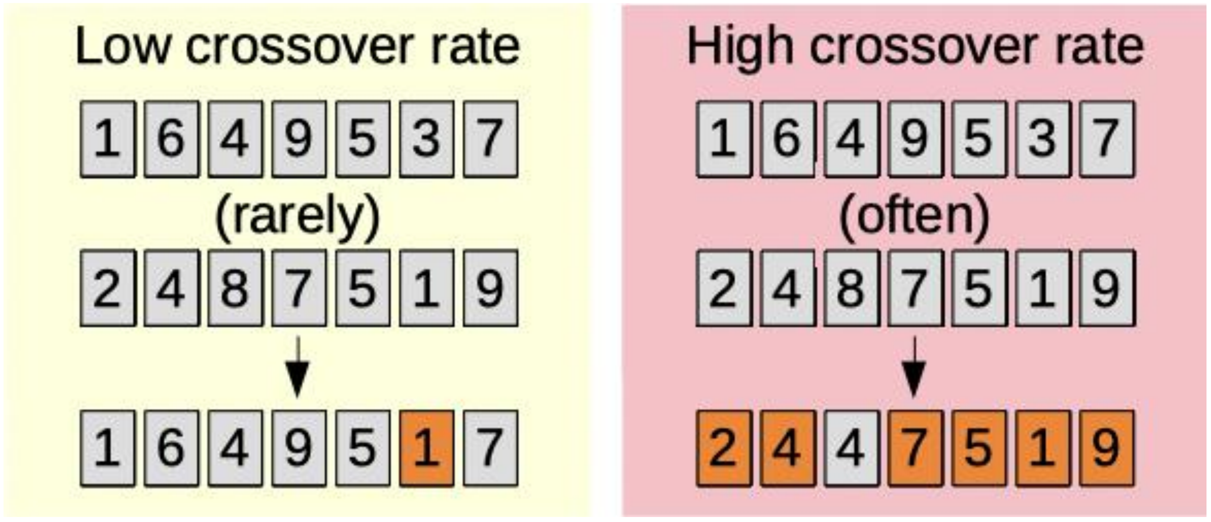

Figure 4: Crossover rate

The crossover chunk parameter (0.00-1.00). Sets the probability for a chunk to be replaced by another one at the same position from another chromosome as illustrated in figure 5.

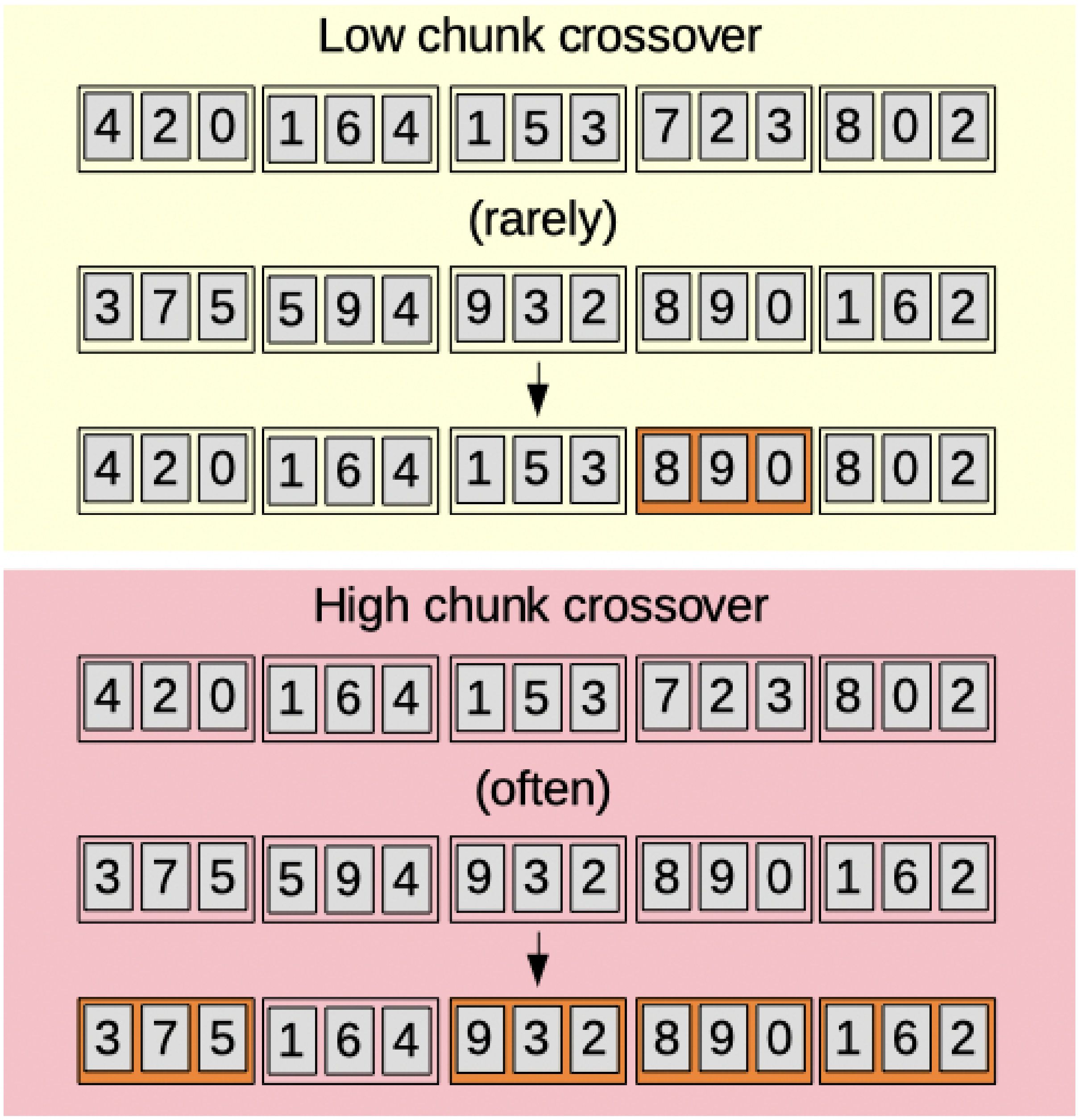

Figure 5: Chunk crossover

Elitism

The elitism rate parameter (0.50-1.00). The higher, the more likely the fittest chromosomes will be parents of the next ones as illustrated in figure 6.

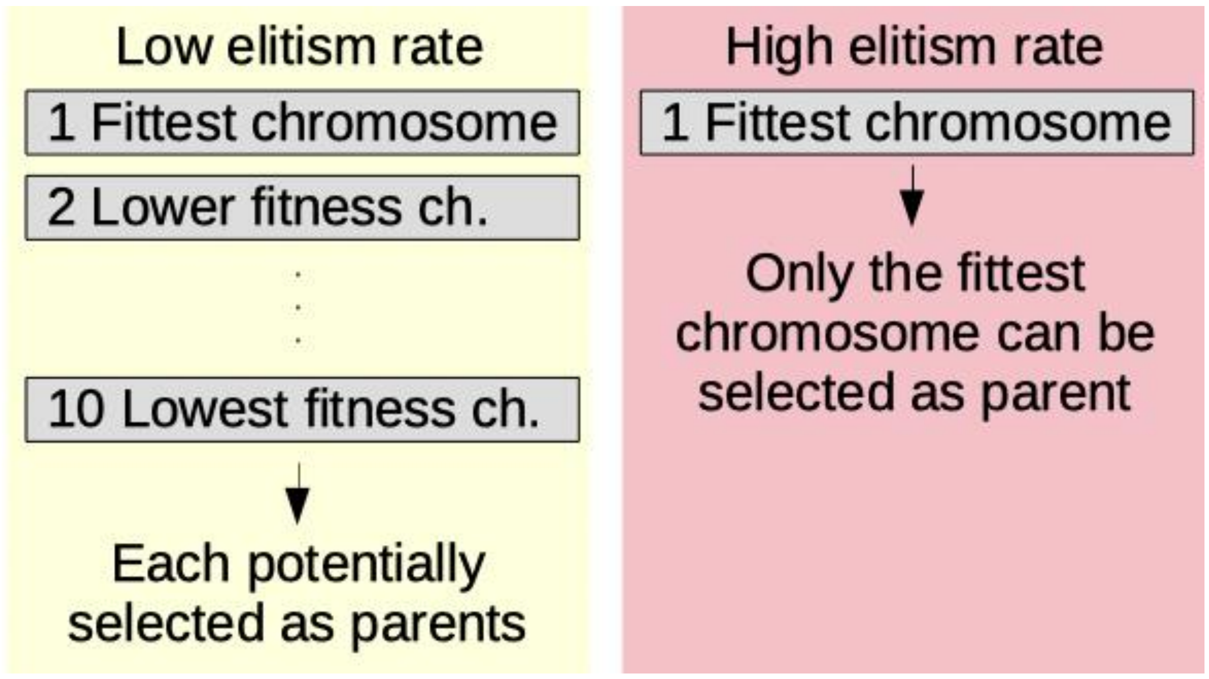

Figure 6: Elitism rate

Add Chunk

Add chunk parameter (0.00-1.00). Sets the probability for a chunk to be inserted between two other chunks as illustrated in figure 7.

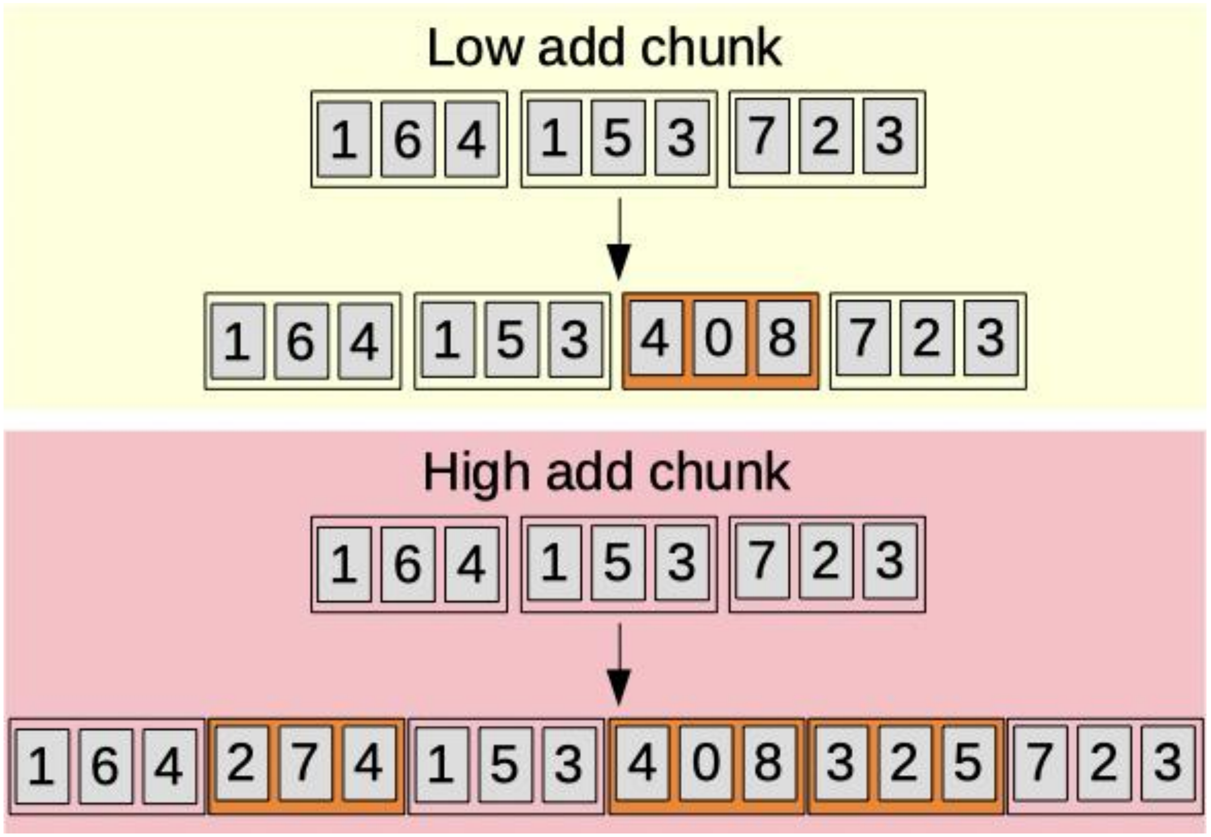

Figure 7: Add chromosome character chunks

Delete Chunk

Delete chunk parameter (0.00-1.00). Sets the probability for a chunk to be deleted as illustrated in figure 8.

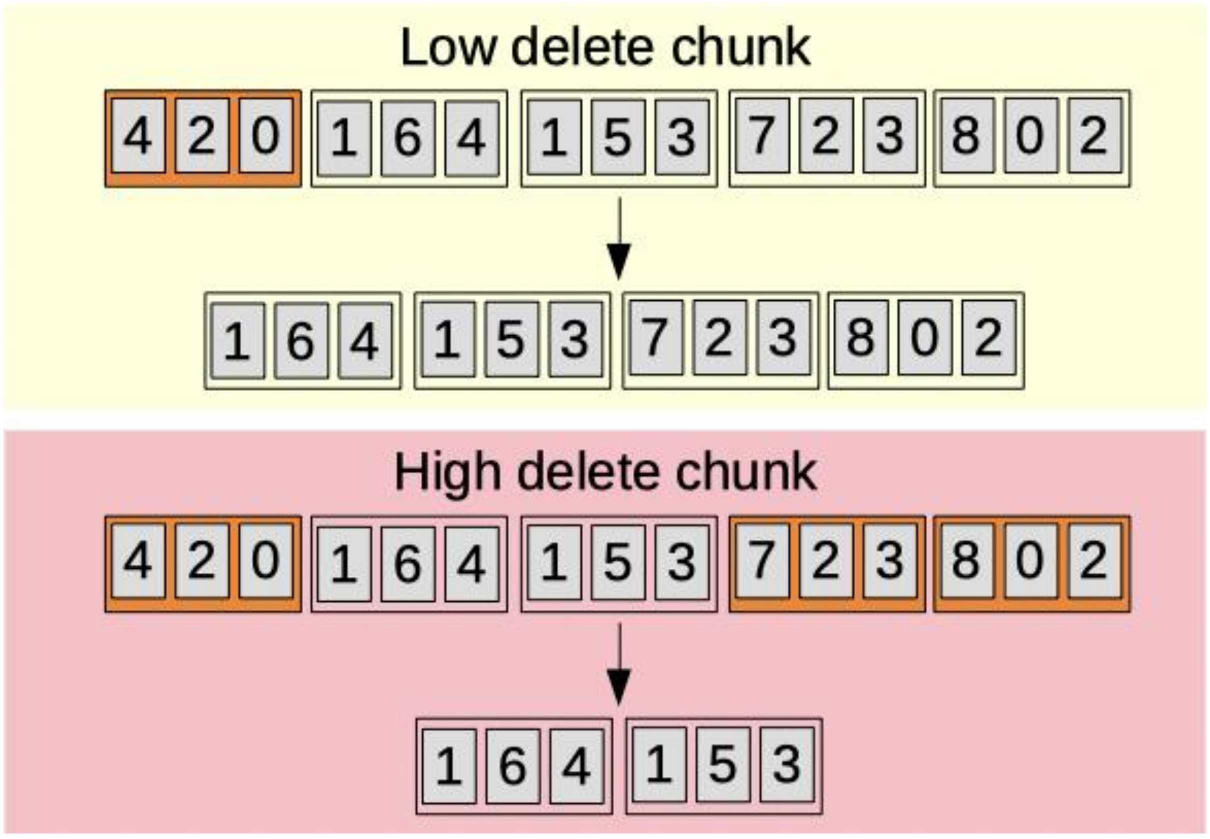

Figure 8: Delete chromosome character chunks

Swap Chunk

Swap chunk parameter (0.00-1.00). Sets the probability for a chunk to be exchanged with another one just before or after within the same chromosome as illustrated in figure 9.

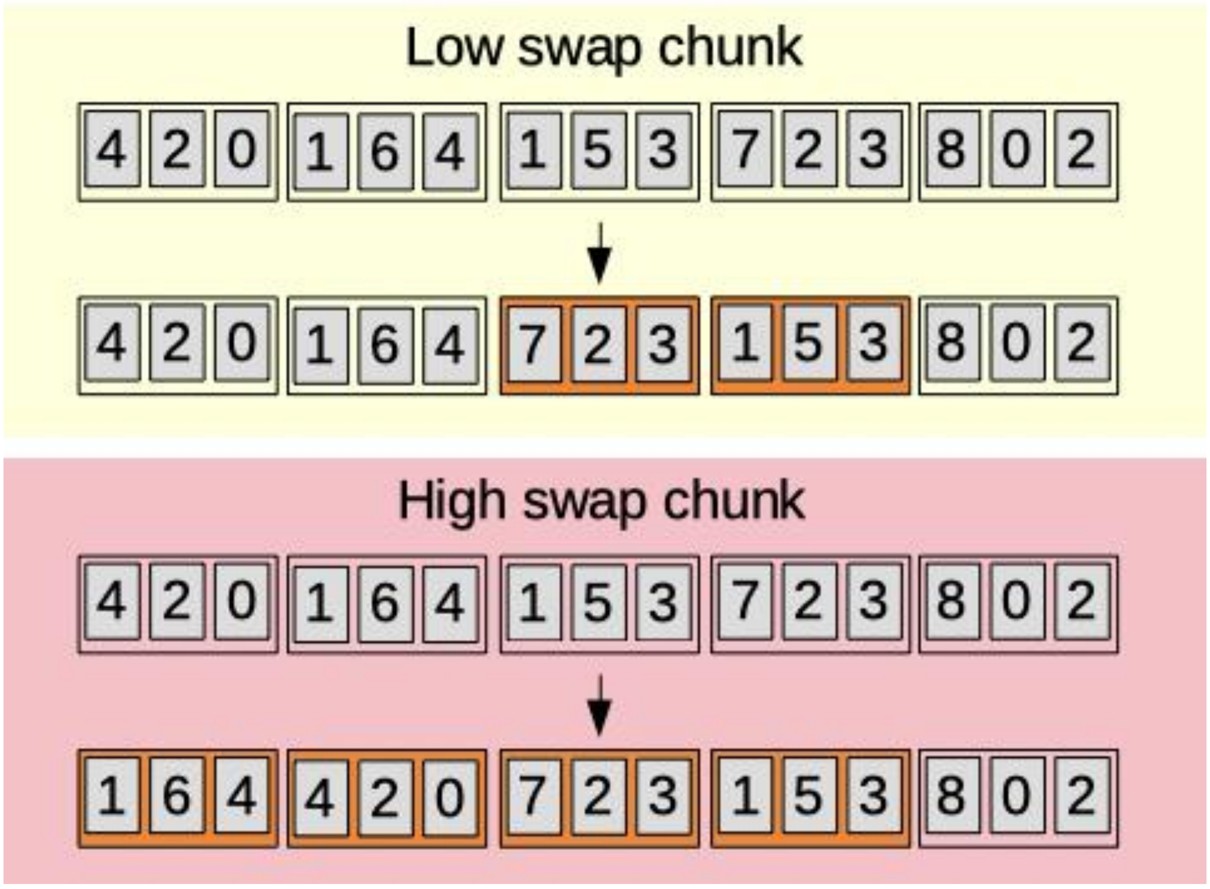

Figure 9: Swap chunk

Niche Fitness Sharing

The niche fitness sharing mechanism is inspired by Golberg and Richardson (1987). For more details on the mechanism, refer to section “Niche Fitness Sharing”. It is controlled by two parameters.

The niche distance ratio (ndr) parameter (0-100). Disable niche with 0. Otherwise, higher value means lower penalty on the fitness (shared fitness) for chromosomes being close to each other as illustrated in figure 10.

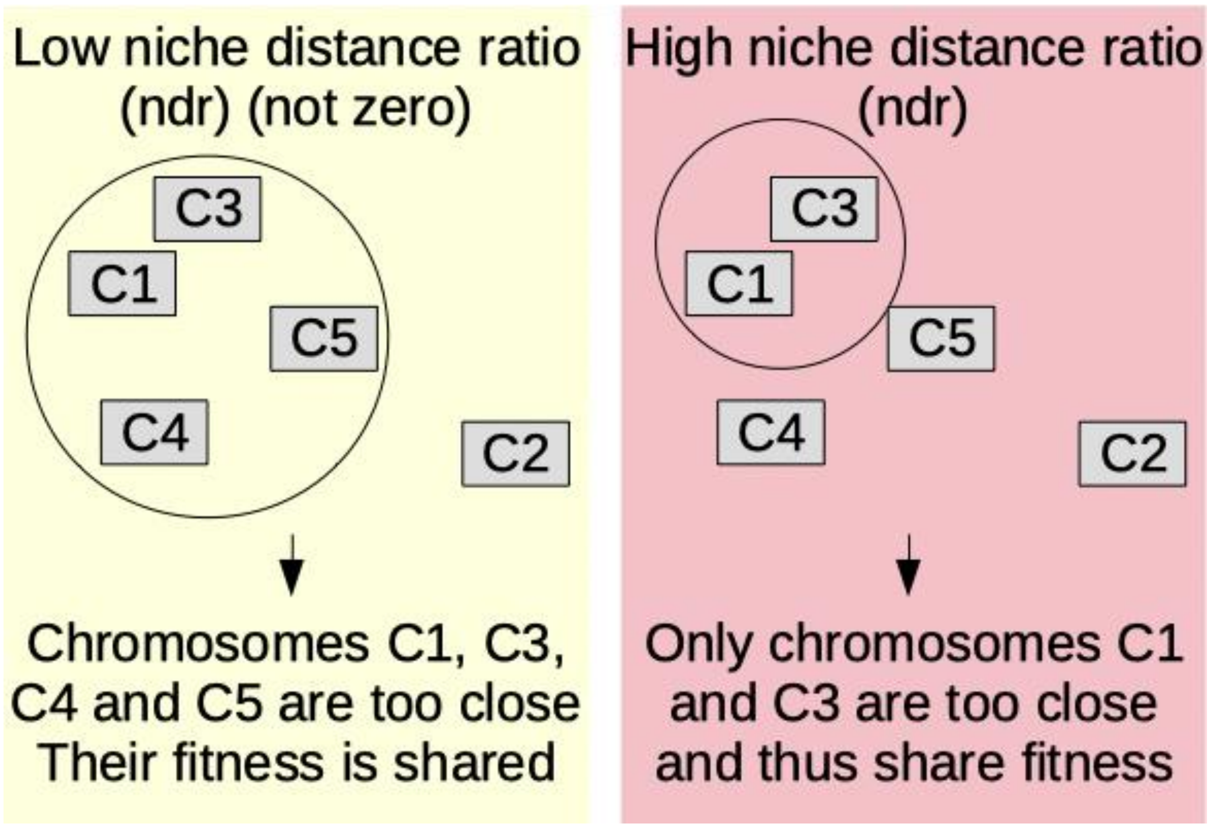
Low niche distance ratio (ndr) (not zero)
C3
C1
C5
C4
C2
Chromosomes C1, C3, C4 and C5 are too close Their fitness is shared
High niche distance ratio (ndr)
C3
C1
C5
C4
C2
Only chromosomes C1 and C3 are too close and thus share fitness

Figure 10: Niche distance ratio (ndr)

The niche power sigma (psi) parameter (1.00-10.00). Only makes sense if ndr != 0. Higher value means higher penalty on the fitness (shared fitness) for chromosomes being close to each other as illustrated in figure 11.

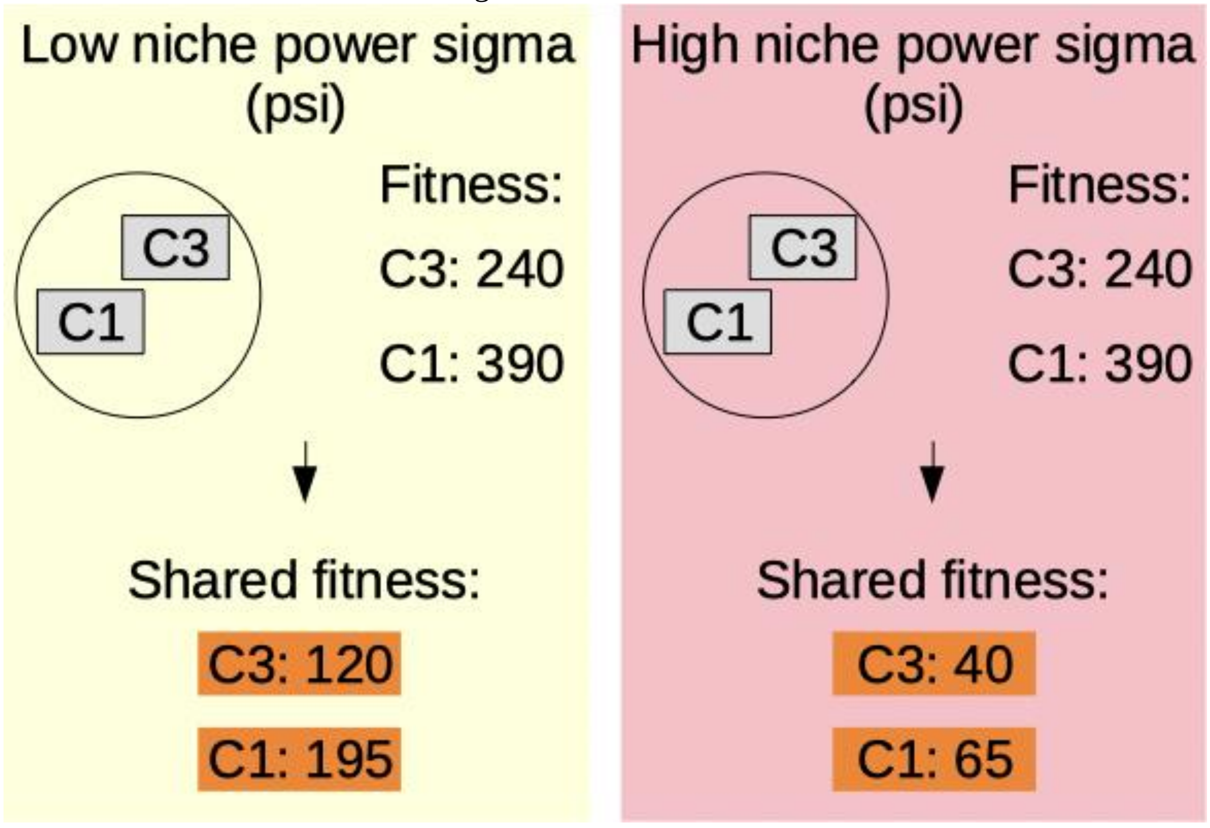

Figure 11: Niche power sigma (psi)

Niche Fitness Sharing

Niche Fitness Sharing Purpose

The purpose of the niche mechanism is to avoid a concentration of the chromosome in a narrow space and spread the search for fitter chromosomes avoiding local optima. The mechanism can be thought of as a penalty for consanguinity. This can avoid evaluating chromosomes that resemble too much existing ones.

Niche Implementation

If the niche mechanism is activated (ndr > 0), the parent chromosomes selection is not based on the raw fitness of the chromosomes but rather on the shared fitness calculated based on the concentration of the chromosomes. The sigma is the distance within which we consider two chromosomes to be close enough to share their fitness. The sigma is calculated from the ndr parameter and the maximum distance between any evaluated chromosome within the population. Then, the power sigma is applied to lower the influence of the distance on the concentration calculation.

This implementation is inspired by Goldberg and Richardson (1987) described page 20 of Michael Scott Brown (2010).

As the population is limited to a 100 evaluated chromosomes kept at a given time, a deletion takes place once this number is reached. The chromosomes deleted are those with the lowest shared fitness (while the non-niche behaviour deletes the lowest raw fitness) with the exception of the latest chromosome with the highest fitness which will never be deleted by the population limits. The 100 evaluated chromosomes limit does not apply to the number of chromosomes that can be evaluated but rather to the number of evaluated chromosomes that remain stored.

The impact of the niche parameters is described in section “Finding Smaller Interlaced PNG”.

Genetic Algorithm Parameters Auto Optimisation

Parameters Auto Optimisation Purpose

Genetic algorithms have parameters which can be optimised. At some point of the evolution, the optimal parameters may not be the same as earlier. It is possible to search for this optimisation manually. However, this is a complex task and may not be practical. The genetic algorithm auto optimisation on oga.jod.li lets the server take care of this optimisation.

Maximise Best Chromosome Fitness

Depending on the use we make of it, the ultimate goal of the genetic algorithm evolution can be to maximise the fitness value for the best chromosome. If we bring this to generations or population batches, this means that batches can be considered having a better performance if their maximum fitness is higher. Thus, the average and minimum fitness are not necessarily relevant to consider the overall performance of the batch. This is especially true if we consider the fact that to avoid local optima, we need to allow lower raw fitness chromosomes to be considered within the population as shown in section “Niche Fitness Sharing”.

Avoiding Old Parameters to Be Used for Too Long

The parameters must be adapted to the optimum search at the time the evolution is executed. Indeed, parameters that lead to a good performance at the beginning of the evolution may not be appropriate after hundreds of batch iterations. To avoid old parameter sets to remain the best ones for a long time while they may not be relevant anymore, each batch iteration leads to the decrease of the fitness of each parameter batch in a proportion (delta = 300) of the highest fitness. Thus, the oldest ones have a much higher penalty than the most recent ones.

Auto Optimisation Parameters

If auto optimisation is globally activated, the following parameters can be set to be auto optimised as mentioned in section “Genetic Algorithm Features”:

- Mutation
- Mutation proportion
- Crossover
- Elitism
- Niche distance ratio (ndr)
- Niche power sigma (psi)
- Add chunk probability
- Delete chunk probability
- Crossover chunk probability
- Swap chunk probability

Bored Kick

It can happen that the best fitness for several batches remains within narrow boundaries without improvement forming thus a plateau. The auto evolution of genetic algorithm parameters can take time to find the appropriate parameters to overcome this plateau.

The bored kick parameter is the number of batches to evaluate for a population to be considered boring and deserving a kick. The kick is given to some parameters provided they are set to be auto optimised. This generally consists in increasing the randomness of the population.

Increase of parameters:

- Mutation
- Mutation proportion
- Chunk addition
- Chunk deletion
- Chunk swap

Decrease of parameters:

- Elitism
- Niche distance ratio (depends on the spread of the population)

This applies only if the latest batch does not contain the highest fitness chromosome.

Bored Kick Outcomes

The bored kick mechanism didn’t have all the impact hoped for so far. Experiments with a bored kick parameter set to 10 batches lead to results illustrated in figure 12 and figure 13.

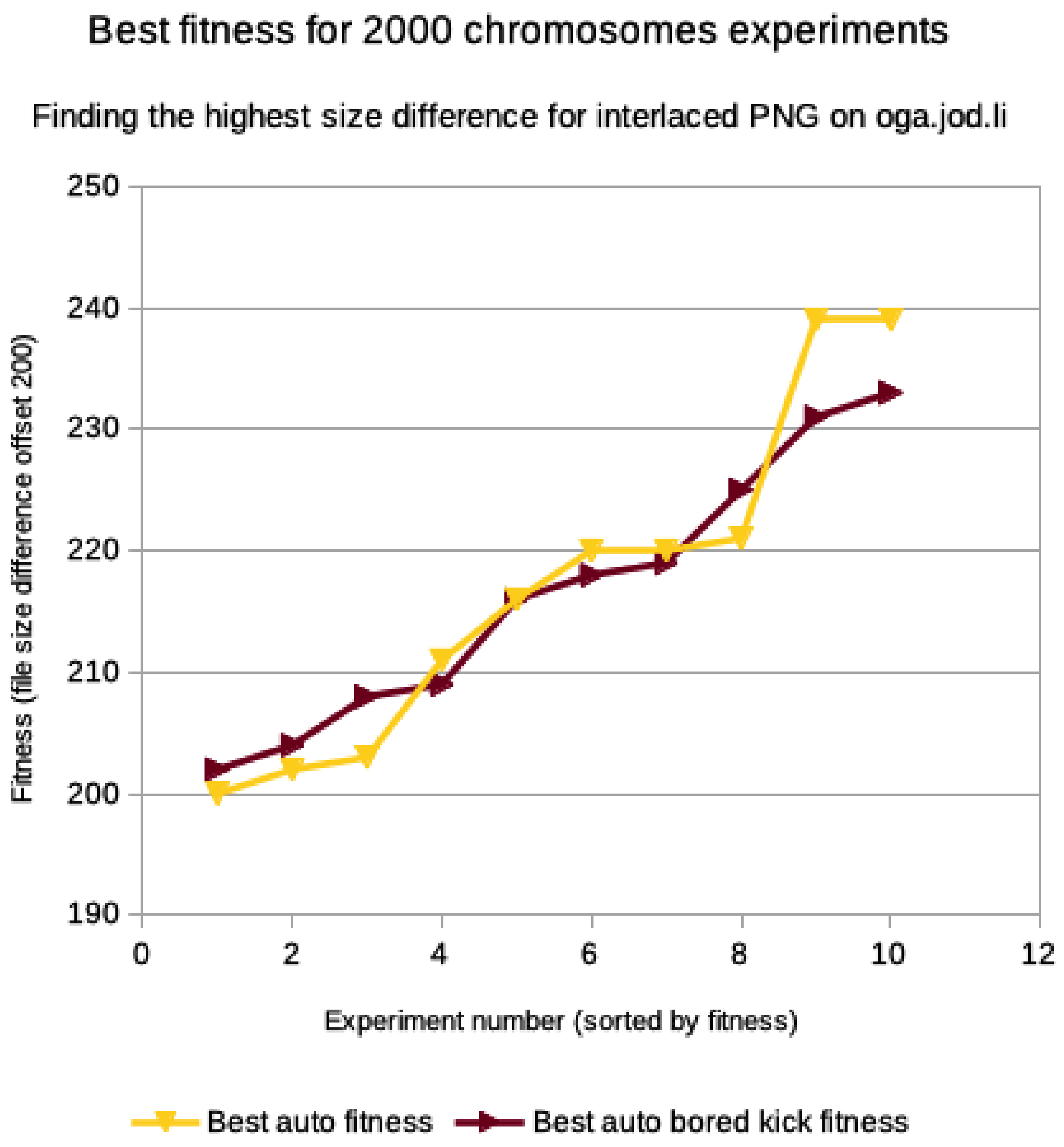

Figure 12: Best fitness for 2000 chromosomes experiments - bored kick of 10

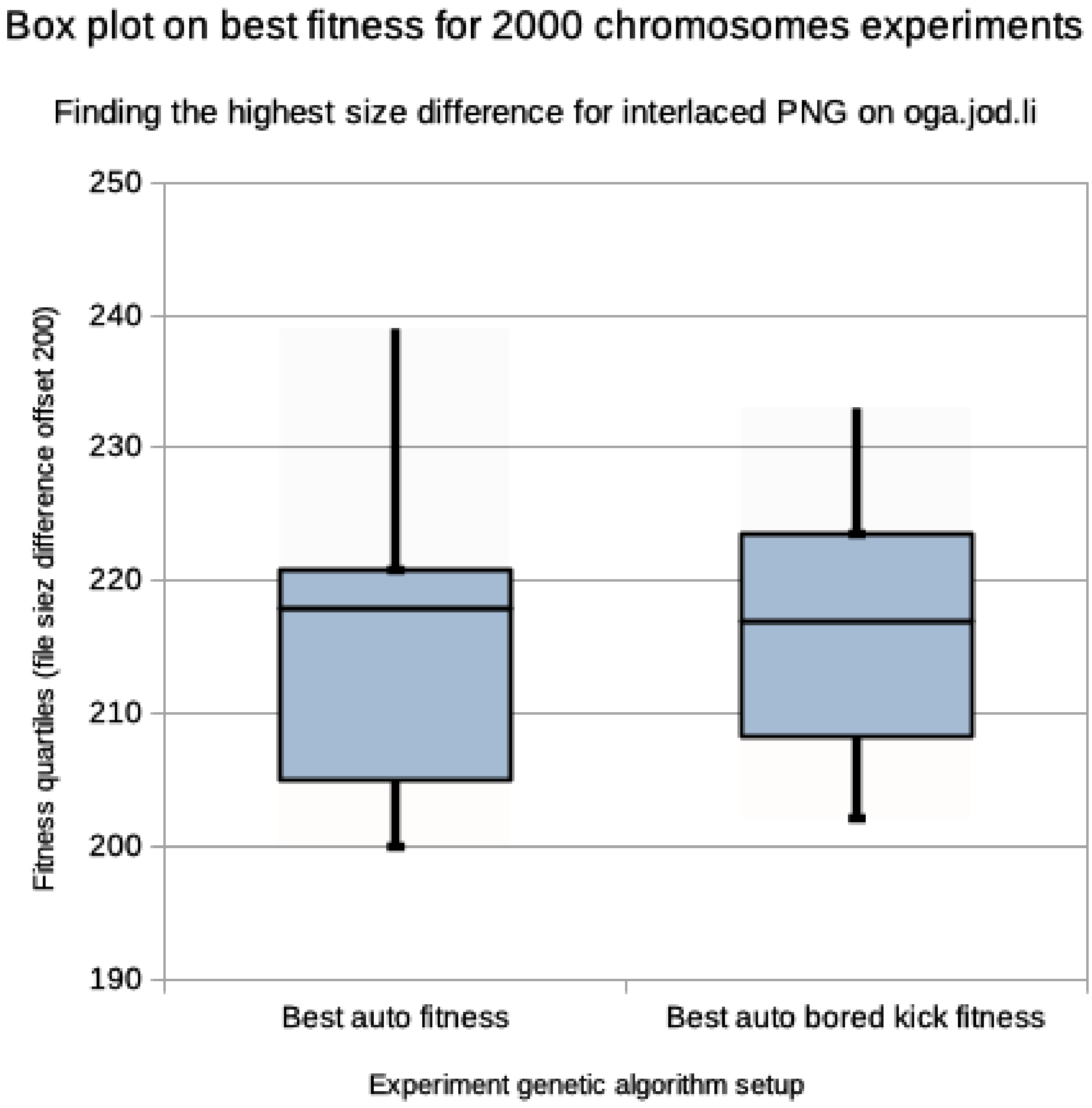

Figure 13: Box plot for best fitness for 2000 chromosomes experiments - bored kick of 10

The general outcome regarding the bored kick parameter through experiment results is that it leads to best fitnesses more concentrated. Indeed, the best fitnesses curve tend to flatten with the bored kick is set to 10 batches.

The bored kick mechanism can be used to increase the probability for a genetic algorithm to reach a reasonable best fitness but it does not necessarily lead to generally improved results. The probability for worst performance is slightly lower as well as the probability for best performance.

Finding Smaller Interlaced PNG

Using the presented solution, several experiments were conducted to find the image for which the interlaced PNG file had the largest positive difference compared to the non-interlaced equivalent version.

The outcome is that tall images with one pixel width have the best result as expected by explanations in section “Size Difference Between Interlaced and Non-Interlaced PNG”.

Each experiment was executed 10 times with 2000 chromosomes evaluated.

The following experiments were executed:

- Auto optimised parameters without niche.
- Auto optimised parameters with niche.

The best chromosome fitness of the above two experiments was found in batch 400 of one of the ten experiment runs without niche as shown in plot figure 14. The parameters used for the batch right before this best batch are displayed in figure 15 and were the ones used for the following experiments:

- Best fixed parameters without niche.
- Best fixed parameters with niche.

The results are presented in figure 16 and figure 17.

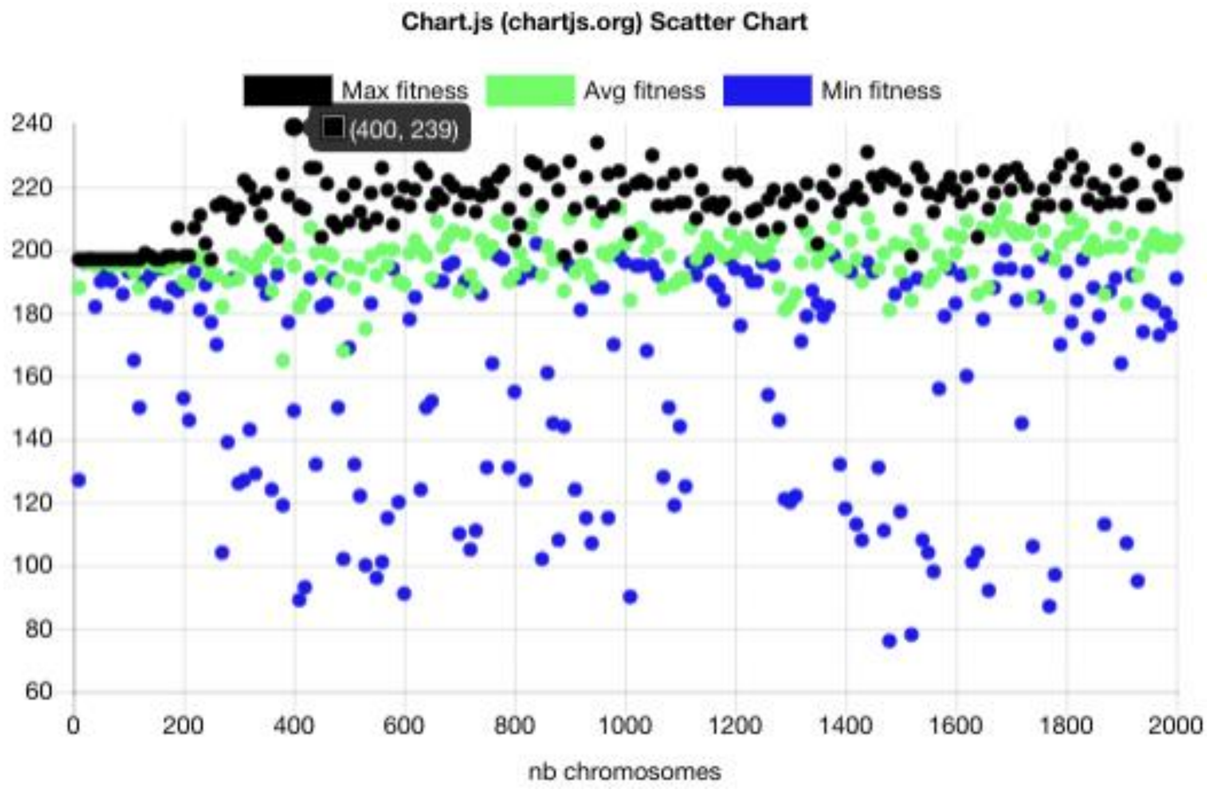

Figure 14: Best auto optimised batch fitness plot

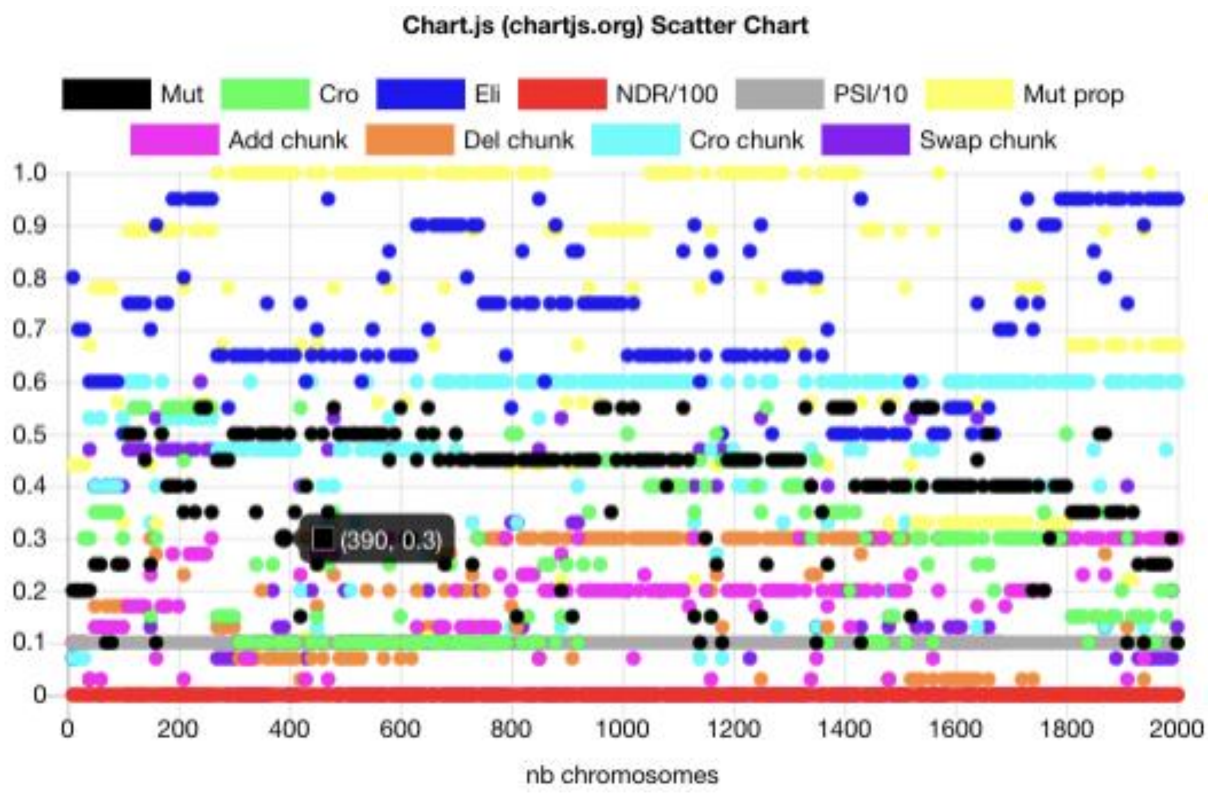

Figure 15: Best auto optimised batch fitness parameters plot

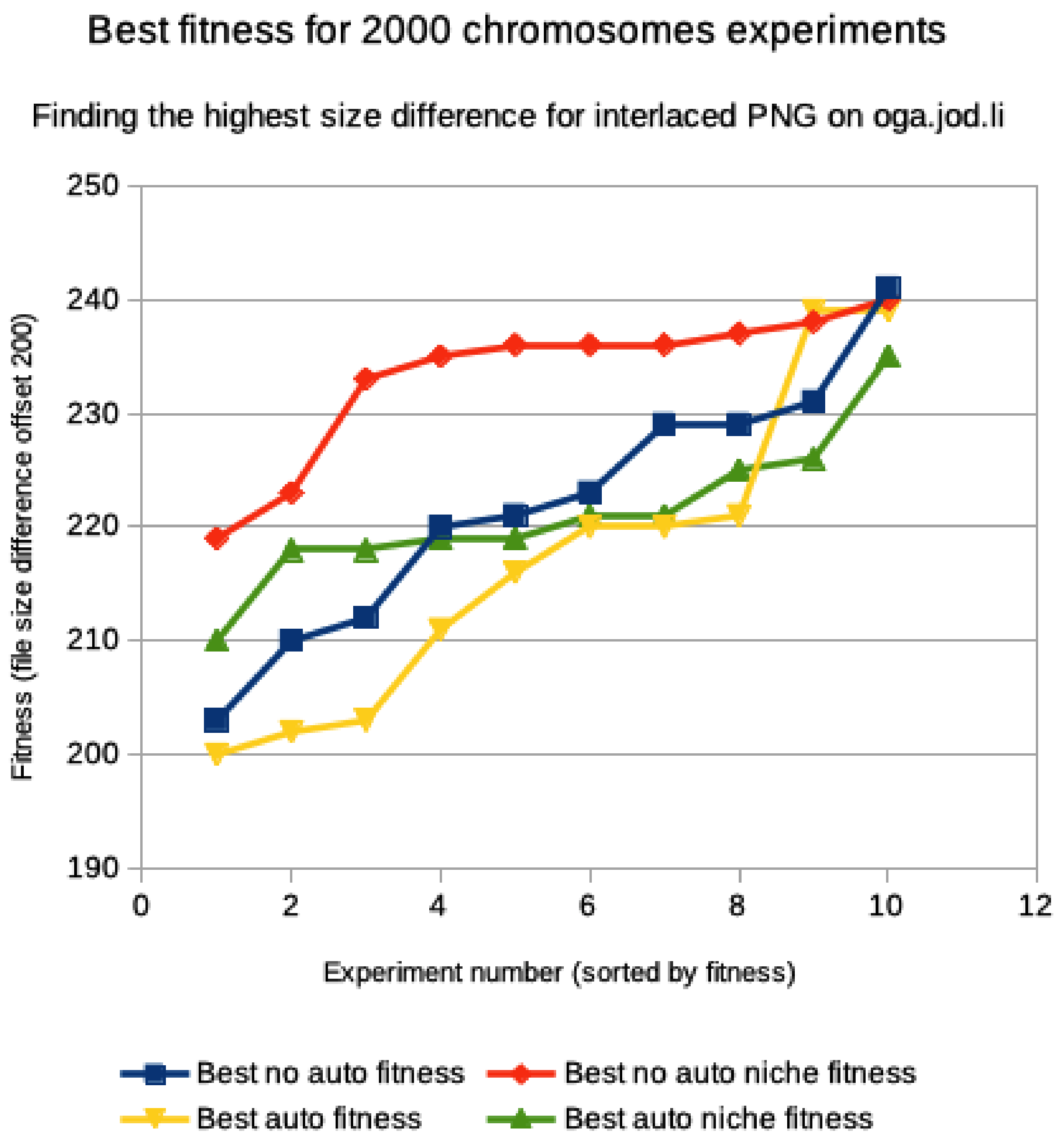

Figure 16: Best fitness for 2000 chromosomes experiments varying auto and niche

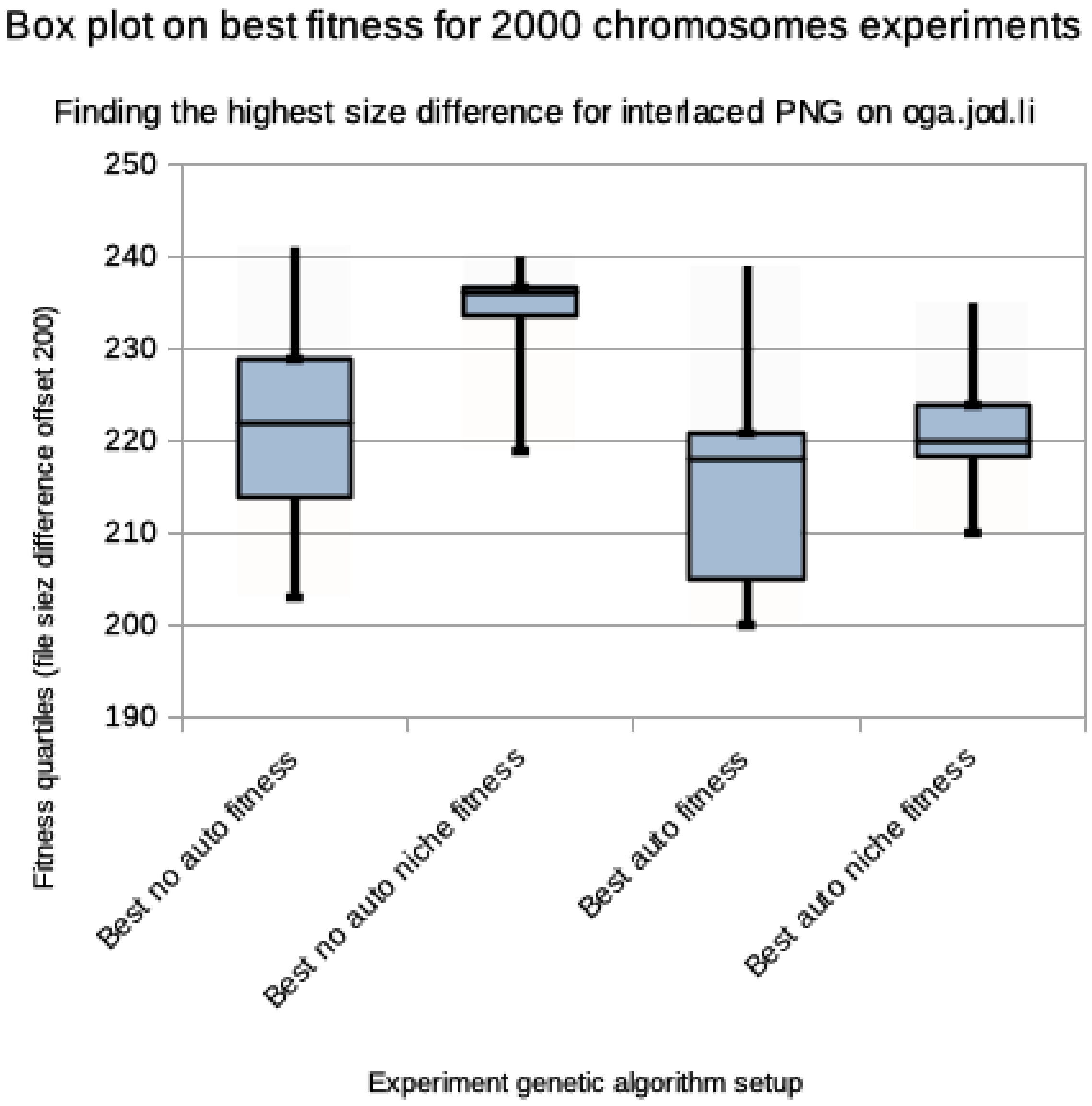

Figure 17: box plots on best fitness for 2000 chromosomes experiments varying auto and niche

The main outcomes are:

- Activating the niche mechanism generally improves the experiment results.
- The overall performance of the fixed experiments with the best parameters of the auto optimised experiments is generally better than the auto optimised experiments. This can be explained by the fact that the auto optimisation requires batches for improving the parameters.

The best chromosome is found with no niche on a non-auto optimised experiment. The resulting image was 41 bytes larger non-interlaced than interlaced. It is 1 pixel wide and 713 pixels high. The non-interlaced file size is 2581 bytes while the interlaced one is 2540 bytes. The image has been reproduced as figure 18 with a height of 24 pixels so that it can be displayed easily. As the rest of the division of 713 by 24 is 17, this many pixels of the end of the original image are truncated and not reproduced here. Finally, the image has been scaled up so that it fits the page. The original files can be found in the ”noauto8” folder of jod.li .

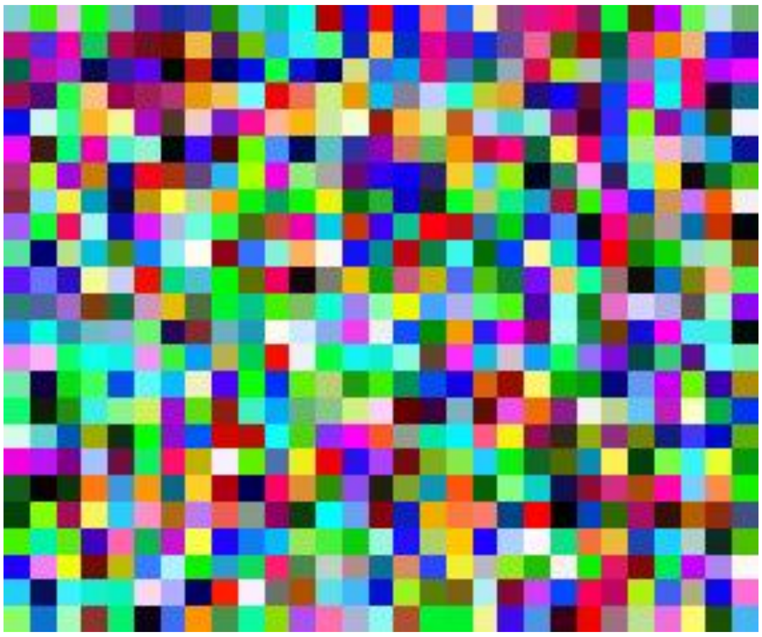
Figure 18: PNG with smallest interlaced size (pieces 24 pixels high cut and brought together to fit the page - original image is 1 pixel wide and 713 pixels high)

No particular pattern is readable from figure 18 that could have explained why this image is the best.

The population batch fitness (min, average, max) that lead to the best fitness plot from oga.jod.li is represented as figure 19. One can notice a plateau at the beginning of the evolution. This bootstrap duration usually refers to images that are not tall. Thus, the evolution needs to find out that images with small width and large height are more likely to lead to smaller size interlaced than non interlaced.

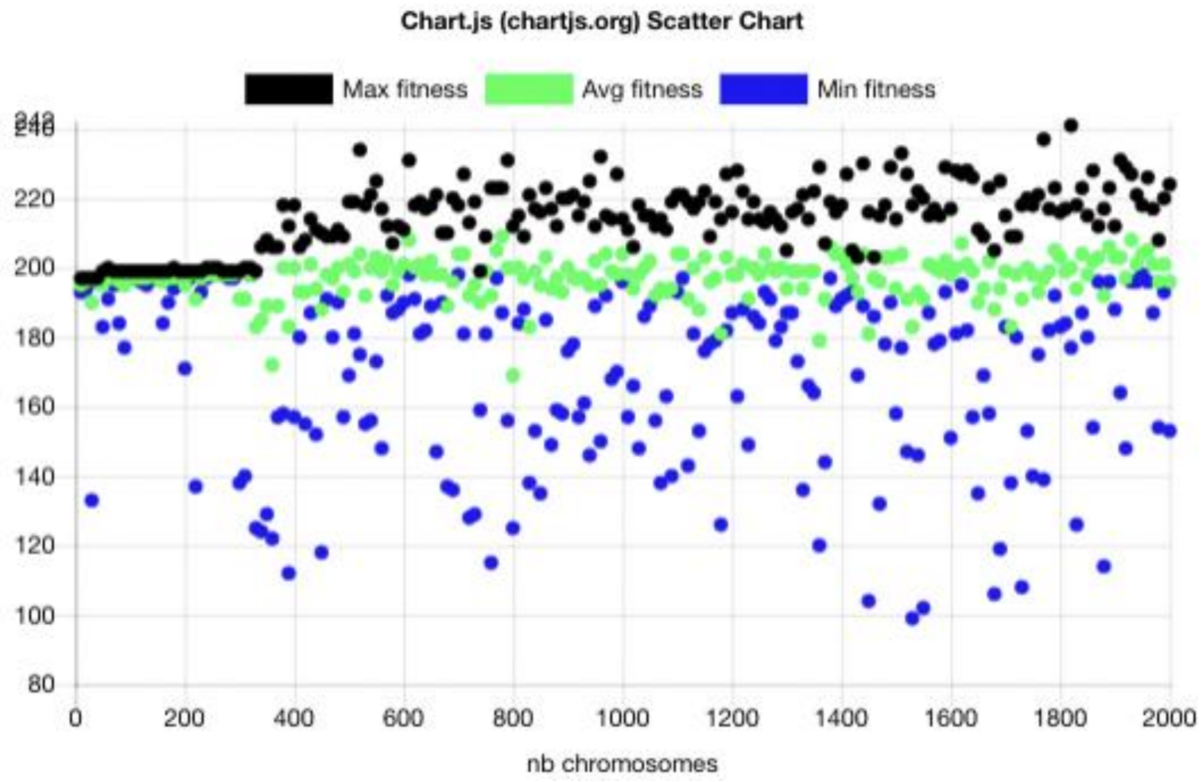

Figure 19: Best population batch fitness (min, average, max) oga.jod.li plot

Conclusions

Genetic algorithms can be used to find interlaced PNG images smaller in size than their non-interlaced equivalent. The resulting images don't seem to display any easily recognisable pattern except their shape which is 1 pixel wide and tall.

Auto optimisation of genetic algorithm parameters through a genetic algorithm allows to find efficient parameters and can improve the performance of the evolution. Though, this is not necessarily enough to find the globally best fitness. Analysis is necessary in the cases presented to find a more global optima. The auto optimisation can be used in many cases to find the best genetic algorithm parameters for a given problem and apply them in a manual way for better results. Other solutions such as for example artificial neural networks which could learn from the genetic algorithm based parameters auto optimisation can outperform the presented solution. The bored kick optimisation attempt doesn't lead to a noticeable improvement.

Niche fitness sharing applied to the genetic algorithm for the considered problem (find PNG files smaller interlaced than non-interlaced), leads to generally better results than without niche.

Future Work

The following future work would either help improving the results or the performance of the solution or increase the knowledge of the areas considered or improve the usability of the oga.jod.li genetic algorithm solution:

- Neural network based genetic algorithm parameters optimisation.
- Select different types of mutation operators.
- Define allowed character sets.
- Select different niche mechanisms as mentioned in Rasmus Ursem (2001).

- Upload population evolution package that can be downloaded, extracted and executed by client devices.
- Report on client performance for fitness evaluation.
- It would be interesting to fine-tune the boundaries and probabilities within which the parameters can be auto optimised. Pruning of the potentially irrelevant parameters would also help.
- Verify whether the auto optimisation is better performing than random choices of the parameters.
- Although it has been repeatedly noticed that the auto optimisation mechanism presented is relevant for the interlaced PNG problem, it would be necessary to demonstrate the relevance or not for other problems.
- Though the conclusion on the niche mechanism has been noticed repeatedly with the interlaced PNG case, it would be preferable to provide more evidence with more experiments that the niche can be useful when evolving populations with genetic algorithms.
- Run more experiments with different values of bored kick and more situations.

Acknowledgements

I would like to thank my wife for her constant and unconditional listening, support, advice, inspiration, laughs and love.

I thank my parents in law who made this work possible by their kindness, hospitality, caring and generosity.

References

Goldberg and Richardson. Fitness sharing algorithm. ICGA, 1987.

ImageMagick Free and open-source software suite for displaying, converting, and editing raster image and vector image files. https://www.imagemagick.org

jod.li oga.jod.li PNG interlaced file size experimentation files. https://jod.li/oga-exp/

jod.li Size of interlaced against non-interlaced PNG uncompressed data. https://jod.li/2018/10/01/size-of-interlaced-and-non-interlaced-png/

Michael Scott Brown. A Species-Conserving Genetic Algorithm for Multimodal Optimization. 2010.

oga.jod.li Optimised Online Genetic Algorithms. https://en.oga.jod.li

Rasmus Ursem. When Sharing Fails. 2001.

RFC2083 PNG (Portable Network Graphics) Specification Version 1.0. https://tools.ietf.org/html/rfc2083, 1997

stackoverflow How can an Interlaced .png file’s size be smaller than the original file?. https://stackoverflow.com/a/52587425/2590508