# ACCURATELY SCRAMBLING IMAGES USING EVOLUTIONARY COMPUTATION AS AN ALTERNATIVE MEAN OF CENSORSHIP

Connor Pillsworth

https://github.com/c-pill/Generating-ACIs

## ABSTRACT

Various methods exist to censor images with the purpose of protecting one's privacy. This paper introduces the idea of generating secure, satisfying images by scrambling pixels using Evolutionary Computational algorithms. These algorithms are compared to a conventional approach of scrambling images and the results are measured and tested using edge detection, an important aspect of computer vision.

## 1. INTRODUCTION

As technology advances and people spend an increasing amount of time online, privacy becomes more difficult to maintain. Among other things, images posted online may contain sensitive information or details that someone may want to keep private. By censoring images can be kept more private. Some familiar image censoring methods include blurring and blocking images (see Section 1.1). The advances in AI and computer vision in detection software furthers the importance of having several options of protecting one's privacy and personal information.

In this paper I test a new method of censoring images. The basis of this method is to create accurately censored images (ACIs) that resemble the original image. This is done by scrambling the pixels in images to a certain extent. For example, a censored image that you want to be 50% similar to the original will have 50% of its pixels in their original positions. By scrambling only a set percentage of pixels, these images will still resemble their original as well as obscure unwanted details. Because there can be millions of pixels in images it is difficult to accurately scramble pixels using conventional means. To accomplish this task, I created two Evolutionary Computational algorithms in an attempt to produce accurate results. I also created one Greedy algorithm to represent using a conventional method to this problem, as well as a comparison to determine if using Evolutionary Computation has merit when dealing with this problem. Most of the data and results from this paper can be found in the Appendix, but everything from this paper images and the code behind these algorithms can be accessed from GitHub using the link provided under the title of this paper.

## 1.1 RELATED WORK[1]

As mentioned in the previous Section, other methods of censorship include Blur and Box methods. The paper "Blur vs Block: Investigating the Effectiveness of Privacy-Enhancing Obfuscation for Images" (Yifang et al.) provides introductions and explanations of both methods in detail, as well as the reactions of participants to the results of these methods. While blurring provides more satisfaction and blocking provides a barrier against human recognition. The purpose of ACIs is to have an image that brings protection and satisfaction to viewers. The paper "Scrambling-Based Tool for Secure Protection of JPEG Images (Korshunov & Ebrahimi) implements a scrambling tool that allows users to scramble sections of images and unscramble these images using a generated key. There are two differences between the techniques used in this scrambling tool and the Evolutionary Computational algorithms presented in this paper. The first is that there are too many random decisions and evaluations in these algorithms to be able to be undone by a key, making them more secure and final. Secondly, these algorithms provide users the option to set how much of the image they want censored. With further research and development, these algorithms could be used in a similar tool to provide customizable and more secure—and more satisfying—results.

# 2. ALGORITHMS

In the efforts of accurately scrambling images, I created three algorithms: two involving Evolutionary Computation and the third being a Greedy solution used to compare to its evolutionary counterparts. Each algorithm starts by storing the original image to be scrambled and the percentage of similarity wanted in the censored image (the Evolutionary Computational algorithms also require the population size *N* of generations). The original image is then converted to RGB format to allow any PNG[2] image to be censored despite its initial pixel format. Afterwards, the image's proportions, number of pixels, and pixel information are recorded. Once the algorithms are complete the best solution is stored in a designated folder with a title in the format of:

$$\{goal\ \%\} \pm \{\%\ off\}\%$$

To evaluate solutions generated by these algorithms the following fitness function is used:

$$fitness = |\%\ pixels\ similar - goal\ \%|$$

The goal of the algorithms is to have a solution with a fitness as close to zero as possible.

## 2.1 EVOLUTIONARY COMPUTATION

In theory, algorithms utilizing Evolutionary Computation should create censored images that are significantly more difficult to unscramble due to the substantial number of random operations involved in creating and modifying solutions. These algorithms are

[1] All related and referenced work can be accessed from References

[2] Each image must be a PNG. This is based on python's PIL library, not the algorithms themselves.

also able to produce an extremely large number of accurate solutions, so no algorithm can try to undo these censorships based on a solution's arrangement of pixels. The first of these algorithms is EPGACI, using an Evolutionary Programming approach, and the second is GAGACI, using a Genetic Algorithm approach. Both algorithms are very greedy in selections because individual diversity is not a priority in this problem. Because the only evaluation is how many pixels match the original, there are no local optimums to avoid. The only goal of these algorithms is to find an accurate solution.

### 2.1.1 METHODS USED IN BOTH EPGACI AND GAGACI

As a result of images having an exceptionally large number of pixels, it is unfeasible to begin with completely scrambled individuals. Completely scrambled images have an almost 0% similarity to the original, so the algorithms would more or less have to reconstruct the original image using the pixels given. This would be a difficult and expensive endeavor. Instead, the initial generation is created by creating copies of the original image and mutating each copy using Mass Swap Mutate. Mass Swap Mutate is a mutation that swaps a random number of pixels up to the number of pixels in the image, which can be represented by the following range:

$$0 \leq swaps \leq \#\ pixels$$

resulting in an individual that is a random percent similar to the original image rather than every individual beginning with a similarly low percentage of similarity. While this mutation is used in both EPGACI and GAGACI to generate the initial population, EPGACI uses it as a mutation function.

Smart Swap Mutate is a mutation that swaps up to double the number of pixels needed to be changed for a fitness of 0 (global optimum). If the fitness is very close to the global optimum—close enough to be considered 0 when dividing—the mutation can only swap up to 1/3 of the number of pixels in the image. This can be represented by the following range:

$$0 \leq swaps \leq \begin{cases} \frac{\#\ pixels}{3} & if\ fitness \approx 0 \\ \frac{\#\ pixels}{fitness} * 2 & else \end{cases}$$

This mutation is used in both EPGACI and GAGACI to have a better chance of improving individuals compared to Mass Swap Mutation. This is because too many swaps often result in worse individuals when fitnesses are closer 0%.

Both algorithms use a Round Robin Approach and an elitist selection to determine survivors. The only difference between the approaches is that EPGACI gives worse individuals a 20% chance of winning while GAGACI is greedier[3] and gives them a 0% chance. The survivor selection takes the top *N* individuals (based on the wins from Round Robin) to survive into the next generation.

### 2.1.2 EPGACI

EPGACI (Evolutionary Programming for Generating Accurately Censored Images) is the first algorithm utilizing Evolutionary Computation to generate ACIs. For this

[3] The reason GAGACI is greedier than EPGACI will be explained in its dedicated section.

paper, the population size used was 15 individuals and the termination criteria was running 200 iterations. Each iteration involves the generation of children using mutation and survivor selection between the parents and the children. See Section 2 and Section 2.1.1 for more information on how fitness is calculated and how the individual generation is initialized.

To generate children one of two mutations were used based on a chance[4] determined by how many iterations are left to be performed, Mass Swap Mutate and Smart Swap Mutate (see Section 2.1.1). If the number of iterations performed is less than 3/5 of the total iterations to be performed, Mass Swap Mutate has an 80% chance of being used to generate the child and Smart Swap Mutate has the remaining 20% chance to be used. When 3/5 of the total number of iterations has been completed the chances of mutation are reversed: Smart Swap Mutate has an 80% chance of being used and Mass Swap Mutate has the remaining 20% chance of being used.

The idea behind these odds is that during the first portion of EPGACI the fitnesses are large not close to 0, and therefore need a large number of pixels to be moved around. Mass Swap Mutate swapping a large number of pixels will do this and find better individuals every generation. If a decent solution is found early on, giving a 20% chance to use Smart Swap Mutate has a chance of finding a better child from the decent parent. Later when fitnesses are closer to the global optimum Smart Swap Mutate will be better to further improve solutions, while the smaller chance of Mass Swap Mutate can allow for larger leaps in swaps that may create fitter children.

Once the children are generated, EPGACI uses a Round Robin approach paired with an elitist survivor selection to find individuals to survive (see Section 2.1.1).

### 2.1.3 GAGACI

GAGACI (Genetic Algorithm for Generating Accurately Censored Images) is the second algorithm utilizing Evolutionary Computation to generate ACIs. The main difference between GAGACI and EPGACI is that GAGACI uses crossovers and only uses Smart Swap Mutate for mutation. For this paper, the population size used was 15 individuals and the termination criteria was running 10 iterations. Each iteration involves parent selection, crossover, mutation, and survivor selection. using mutation and survivor selection between the parents and the children. See Section 2 and Section 2.1.1 for more information on how fitness is calculated and how the individual generation is initialized.

GAGACI uses Tournament Selection with 3 opponents to select parents for crossover. For crossover GAGACI uses a modified version of PMX crossover that I created called Smart PMX crossover. Smart PMX is the same as PMX crossover but has a limit to how large the operation can be (similar to the logic behind Smart Swap Mutate). The maximum range of the crossover is up to double the number of pixels needed to reach the goal percentage from the average of the parent's fitnesses. If the average fitness is very close to the global

[4] The chance of a specific mutation being used is rerolled for each child.

optimum, Order crossover is used instead. This is represented by the following:

$$d: maximum\ distance\ of\ PMX\ cross$$

$$a: average\ fitness\ of\ parents$$

$$d = \frac{\#\ pixels}{a} * 2$$

$$crossover \begin{cases} Order & if\ d \approx 0 \\ PMX\ with\ limit\ d & else \end{cases}$$

The limiting of PMX and Order crossovers to create Smart PMX is solely to make GAGACI faster while still creating accurate solutions. Because of the thousands and even millions of pixels in images, any type of crossover used on images is computationally expensive. This is especially true when considering the number of computations used in PMX crossover, so to be able to use PMX in this problem there must be a limit on how large of an operation it can be. Using only PMX crossover would produce accurate results, but due to time limitations I could not reasonably assess the impact of performing only PMX crossovers compared to using Smart PMX. The amount of time crossovers taking is also why GAGACI uses only 10 iterations compared to EPGACI's 200 iterations. Due to time limitations, I could only test GAGACI running using 20 iterations. This did not show any remarkable improvement in its solutions compared to using only 10 iterations, but with more time this could be evaluated further. In Section 3.3, I will discuss why further testing in this regard may not be necessary. Smart PMX produces one child and is performed *N* number of times each iteration, resulting in *N* children.

After all crossovers are complete, each child is mutated using Smart Swap Mutate. Smart Swap Mutate is the only method of mutation used in GAGACI because crossover is used to 'explore' solutions so Mass Swap Mutate is not necessary[5].

Once the children are finished mutating, GAGACI uses a Round Robin approach paired with an elitist survivor selection to find individuals to survive (see Section 2.1.1). While EPGACI's Round Robin gives worse individuals a 20% chance of winning, GAGACI does not. This is because GAGACI uses less iterations than EPGACI, so it does not have as many opportunities to explore worse solutions. In various problems, not being able to explore diverse solutions is a weakness, but since diversity is not a priority in this problem exploration is not crucial in creating good individuals. It is also important to note that if GAGACI was not extremely greedy and elitist, the lack of iterations used would be less likely to find good solutions and sacrifice fitness for diversity and exploration. If GAGACI was given more iterations and only used PMX crossover it would be able to be less greedy and be able to explore more. Due to time limitations, I could not test whether or not this version of GAGACI would outperform the version of GAGACI used and described in this paper.

## 2.2 GSGACI

GSGACI (Greedy Solution for Generating Accurately Censored Images) is a Greedy algorithm used to generate ACIs. The secondary purpose of GSGACI is to

[5] Compared to EPGACI using Mass Swap Mutate to 'explore.'

compare itself with EPGACI and GAGACI as a way of determining whether or not Evolutionary Computation feasibly generates good solutions based on several factors. These factors will be introduced and analyzed in Section 3.

GSGACI begins by creating a copy of the original image. From left to right and top to bottom, GSGACI goes to each pixel in the gen position and determines whether or not the pixel's position should match its position in the original image. The chance of the pixel matching is the goal %. For example, if you want a solution that is 25% similar to the original each pixel will have a 25% chance of staying in that position. If GSGACI decides that the pixel should not be in its original position, it will be replaced by another pixel from a random, undecided position.

Pixels replaced using this method are lost, but upon testing other approaches where the pixel replaced is stored elsewhere for later use, losing pixels is much faster and leads to equally accurate results. There is likely a better greedy solution where it is fast, accurate, and does not lose any pixels, but due to time limitations I was unable to find it. GSGACI is used to compare different approaches— Greedy vs Evolutionary Computation—to create an accurately censored image, so this pixel loss is acceptable for GSGACI[6].

## 3. RESULTS OF ALGORITHMS

The images used in these algorithms are Me.png, Face.png, and Background.png with, respectively, 416,034 pixels, 200,970 pixels, and 594,864 pixels. This is important when considering the effect on how long these algorithms take to generate results.

To test EPGACI, GAGACI, and EPGACI each algorithm was run with 3 different goal percentages: 25%, 50%, and 75%. For each goal percent, each algorithm was run 5 times. Every time an algorithm generated a solution the speed (time it took in seconds) and the fitness of the solution were recorded. By using 5 runs of every algorithm and goal percentage the following were calculated: Average speed, standard deviation of speed, average fitness, and standard deviation of fitness. The best fitness for each algorithm was recorded for every goal percentage[7].

### 3.1 COMPARING RESULTS OF DIFFERENT IMAGES

As mentioned at the beginning of Section 3, the images used in this paper are all varied sizes and therefore a different number of pixels. By using these images, the robustness of the algorithms can be tested. In other words, one can see if the algorithms will perform differently when processing differently sized images. In theory the algorithms should not have any trouble handling different sizes. This can be tested by comparing the speed (time it takes for

[6] Pixel loss is absolutely unacceptable in EPGACI and GAGACI due to the number of computations involved. If too many pixels are lost, the resulting image will lose resemblance, and the purpose of these algorithms would be moot.

[7] Due to Two-Column format, some figures are not displayed but are mentioned. All figures and images are accessible from the Appendix. The tables of data used to create figures are also available in the Appendix.

algorithm to complete) and the accuracy (how close the solution is to the global optimum of 0%) of each algorithm for each goal percentage. Figures 1, 3, and 5 display the total time it took for each algorithm to run each goal percentage 5 times as well as the average time taken and the standard deviation of time of the respective image. Figures 2, 4, and 6 display the average fitness, standard deviation of fitnesses, and the best fitness found for each algorithm after five runs of each goal percentage. Figure 7 displays a summary for when each algorithm should be used.

### 3.1.1 SPEED OF ALGORITHMS

Based on Figures 1, 3, and 5, the following patterns can be seen regarding the speed of each algorithm processing each image for every goal percentage. The time it takes for EPGACI and GSGACI to perform five runs is consistent regardless of the goal percentage. GAGACI, however, is inconsistent with the total time spent, average time spent, and the standard deviation of time spent. GAGACI at 25% takes the longest amount of time, then 50%, then 75% takes the least amount of time. The reason for GAGACI's inconsistency is most likely Smart PMX and the other adjustments made (see Section 2.1.3) in an effort to minimize the amount of time it takes for the algorithm to develop a solution.

These figures also show the dramatic difference in speed the algorithms possess. GAGACI takes an enormous amount of time to generate solutions[8], sometimes taking approximately one hour to complete runs that EPGACI completes in a matter of minutes[9]. Despite EPGACI's outperforming GAGACI, it is far slower than GSGACI where solutions are generated in milliseconds.

### 3.1.2 ACCURACY OF ALGORITHMS

Based on Figures 2, 4, and 6, the following patterns can be seen regarding the accuracy of each algorithm processing each image for every goal percentage. EPGACI is shown to, for the majority of the time, produce the most accurate results and has the best average results. GAGACI is shown to have an irregular pattern of results, sometimes being able to compete with EPGACI, but often performing significantly worse than its counterpart. EPGACI shows the majority of its consistency in producing the most accurate results. In terms of standard deviation GSGACI typically outperforms both EPGACI's and GAGACI's inconsistent results. Despite its consistency, GSGACI's results are much less accurate on average and even its best results are significantly less accurate than the Evolutionary Computational algorithms.

[8] These results highlight the necessity of taking strides to reduce the run time of GAGACI, with GAGACI running for an incredibly long time even after the adjustments highlighted in Section 2.1.3.
[9] The longest amount of time GAGACI took to generate an image is seen in Table 5, where Background.png is used with a goal of 50%, taking one hour and 42 minutes.

## 3.2 SUMMARY OF RESULTS

| Best for … | EPGACI | GAGACI | GSGACI |
|---|---|---|---|
| Accurate Results | 8 | 1 | 0 |
| Accurate Average of Results | 9 | 0 | 0 |
| Consistent Results | 1 | 0 | 8 |
| Fastest Results | 0 | 0 | 9 |
| Consistently Fast Results | 0 | 0 | 9 |

*Table 7: Summary of Results*

Based on the data collected and summarized in Tables 8-12, Table 7 displays where each algorithm's solutions are the best. Because each of the three algorithms were used with three different goal percentages, each algorithm has nine opportunities to have generated the best solution for a category. An example of this process is shown using a portion of Table 8, which displays which algorithms generated the best, or most accurate, fitnesses. This data was counted and recorded in Table 7 under 'Best for Accurate Results'.

| Best Fitness | Me.png | Face.png | Background.png |
|---|---|---|---|
| 25% Algorithm | EPGACI 25% | EPGACI 25% | GAGACI 25% |
| 50% Algorithm | EPGACI 50% | EPGACI 50% | EPGACI 50% |
| 75% Algorithm | EPGACI 75% | EPGACI 75% | EPGACI 75% |

*Table 8: Best Fitness*

Because EPGACI appears 8 times, it is given 8 points in Table 7. The 'Fastest' category stems from the average time taken to complete, and the 'Consistent' categories stem from the standard deviation data. All related tables and figures can be found in the Appendix.

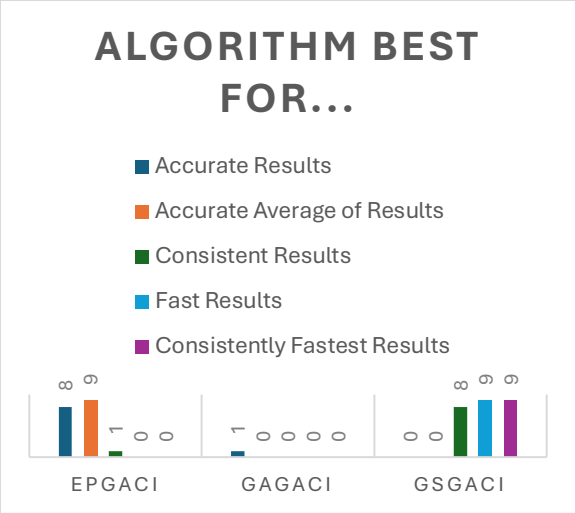



*Figure 7: Displays data from Table 7*

Based on these results EPGACI and GSGACI are the two dominant algorithms out of the three. EPGACI seems to be the best for producing the most accurate results and the best results on average, getting 9/9 and 8/9 in these respective categories. GSGACI seems to be the best for producing the most consistent results in terms of accuracy and speed, as well as the fastest results. GSGACI got 8/9, 9/9, and 9/9 in these respective categories. GAGACI got a total of one point from beating EPGACI once in the most accurate result.

## 3.3 DISCUSSION OF RESULTS AND FURTHER DEVLOPMENT

The data presented throughout Section 3 and its subsections give a clear understanding of the performance of these algorithms. EPGACI is able to generate fully accurate results in minutes, GAGACI is able to generate results almost as accurate (and occasionally better) results in tens or hundreds of minutes, and GSGACI is able to generate results— that are much less

accurate than EPGACI and GAGACI— in milliseconds. If accuracy is the priority choose EPGACI. If time is your priority and you do not mind losing pixels (see Section 2.2) choose GSGACI. GAGACI is outperformed by both algorithms in its current state and is not viable for generating ACIs. As previously mentioned in Sections 2.1.3 and 3.1.1, GAGACI's adjustments to save time likely hinders its ability to produce accurate results. If unrestrained it could feasibly outperform EPGACI, but at the cost of larger amounts of time. Further development of GAGACI would likely produce lackluster results when considering the amount of time it would take to adjust and wait for results to be generated. Because GSGACI will always be the fastest solution, as it is a Greedy solution, further development of EPGACI should focus on improving the accuracy of the results with less focus on improving speed. GSGACI on the other hand could be further developed to fix the issue of losing pixels and reevaluated against EPGACI to determine if GSGACI could generate more competitive results.

# 4. ACIs AS A METHOD OF CENSORING IMAGES

While the emphasis of this paper is on comparing the feasibility of utilizing Evolutionary Computational algorithms (see Section 2) to censor images, it is equally—if not more important—to determine whether or not the method of censorship used by these algorithms have any merit.

## 4.1 TESTING ACIs AND THE RESULTS

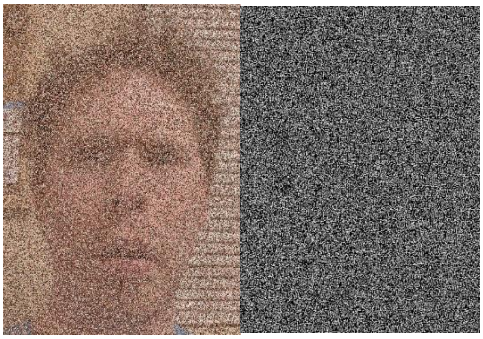
Canny edge detection[10] will be used to test how ACIs perform as a method of censorship. As the name suggests, canny edge detection is used to find edges or boundaries in images when using computer vision. The result of canny edge detection is shown on Face.png. Because of the importance of edge detection in image processing, if ACIs are able hinder canny edge detection then they have effectively censored the image against computer vision and therefore is a method of censorship[11].

*Face.png before and after using canny edge detection.*

The ACIs processed by canny edge detection are the best ACIs generated by EPGACI because this algorithm, on average, produces the most accurate results (see Section 3.2). After processing the original images and the best ACIs from EPGACI-25%, EPGACI-50%, and EPGACI-75% of each image, it is evident that ACIs are able to keep canny edge detection from working properly.

[10] Find "CANNY EDGE DETECTION: A COMPREHENSIVE REVIEW" and "Recent advances on image edge detection: A comprehensive review" in References for information and explanations of the importance of canny edge detection in image processing.

[11] While this is a bold claim it has merit. Further testing should be considered to add more justification (see Section 4.3.



*Face.png 25 +- 0.0072% before and after using canny edge detection*

Canny edge detection consistently produces images similar to the example above for every ACI processed[12].

## 4.2 DISCUSSION OF RESULTS AND FURTHER DEVLOPMENT

As displayed in the previous Section, ACIs' ability to prevent canny edge detection highlights its ability as a method of censorship. Further testing was considered for this paper, but time limitations prevented any additional tests of how well ACIs are able to censor images. Despite this lack of testing, I believe that hindering canny edge detection shows enough merit to call this method of censorship successful. Further testing should include other edge detection algorithms as well as using AI to see if ACIs can prevent object recognition to any degree. Testing with AI can also be used to compare how well ACIs perform versus other methods of censorship. Another route of exploration can be combining ACIs with other methods of censorship to see if there is any merit in doing so.

# 5 CONCLUSION

This paper presents two algorithms, EPGACI and GAGACI, utilizing Evolutionary Computation to generate accurately scrambled images (ACIs) as well as one greedy algorithm, GSGACI, as a means to compare the quality and timeliness of results. EPGACI is both timely and, in the vast majority of tests, produces the most accurate results. GAGACI, on the other hand, takes an exceedingly long time to generate results and, in the vast majority of tests, produces results less accurate than its counterpart. Both algorithms outperform GSGACI in terms of results but are much slower with EPGACI generating results in minutes, GAGACI in tens or hundreds of minutes, and GSGACI in milliseconds. Based on these results EPGACI is the best choice for generating ACIs due to its timely, accurate results without the pixel loss seen in GSGACI.

Additionally, this paper touches on the viability of ACIs as methods of censorship by testing ACIs generated using EPGACI. By processing these images using canny edge detection, it is clear that computer vision will struggle to detect objects in ACIs, hindering the ability of algorithms and AI to recognize objects in these images. This displays the potential of ACIs as an effective method of censorship, but additional testing should be pursued in order to determine how well this method censors images.

[12] All images and processed/generated images are accessible from the GitHub link provided at the beginning of this paper.

**REFERENCES**


Agrawal, Himangi, and Krish Desai. “CANNY EDGE DETECTION: A COMPREHENSIVE REVIEW.” *International Journal of Technical Research & Science*, vol. 9, no. Spl, 25 June 2024, pp. 27–35, www.researchgate.net/publication/383184441_CANNY_EDGE_DETECTION_A_COMPREHENSIVE_REVIEW, https://doi.org/10.30780/specialissue-iset-2024/023.

Jing, Junfeng, et al. “Recent Advances on Image Edge Detection: A Comprehensive Review.” *Neurocomputing*, vol. 503, Sept. 2022, pp. 259–271, www.sciencedirect.com/science/article/abs/pii/S0925231222008141, https://doi.org/10.1016/j.neucom.2022.06.083.

Li, Yifang, et al. “Blur vs. Block: Investigating the Effectiveness of Privacy-Enhancing Obfuscation for Images”. 1 July 2017, ieeexplore.ieee.org/document/8014910, https://doi.org/10.1109/cvprw.2017.176.

Pavel Korshunov, and Touradj Ebrahimi. “Scrambling-Based Tool for Secure Protection of JPEG Images.” *2022 IEEE International Conference on Image Processing (ICIP)*, 1 Oct. 2014, pp. 3423–3425, www.researchgate.net/publication/264860353_Scrambling-based_tool_for_secure_protection_of_JPEG_images, https://doi.org/10.1109/icip.2014.7025694.

# APPENDIX

## *Figures*

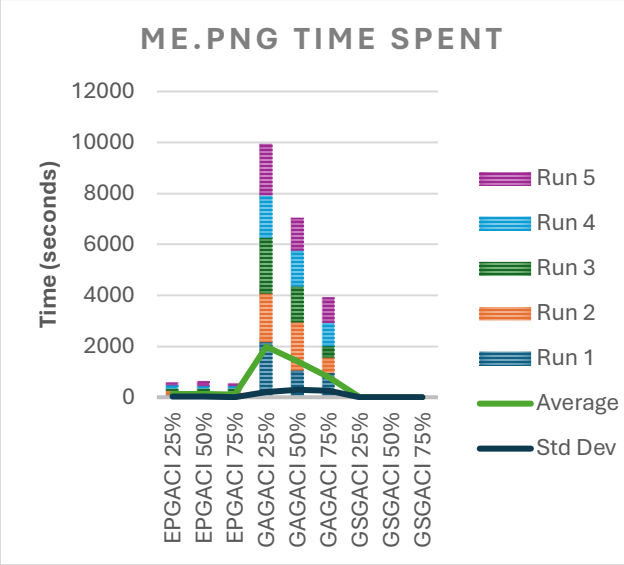



*Figure 1: Me.png Time Spent*

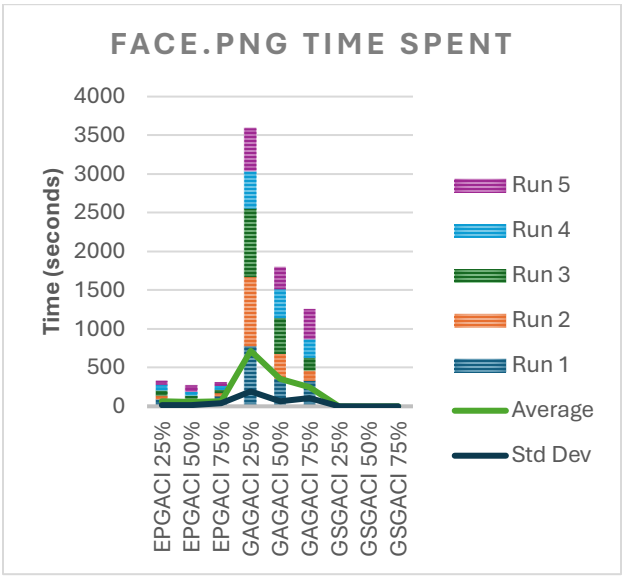



*Figure 3: Face.png Time Spent*

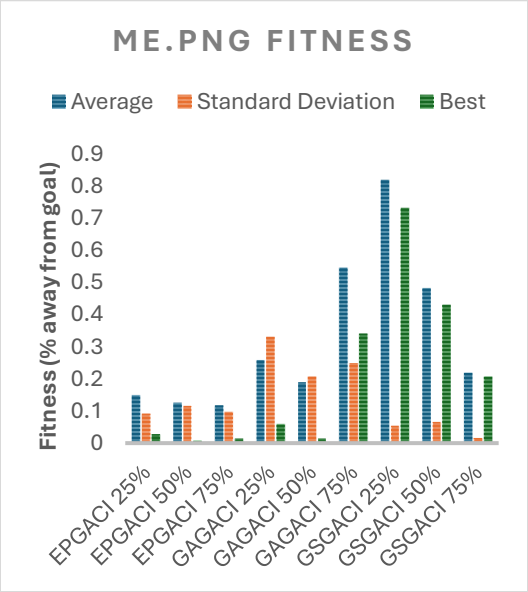



*Figure 2: Me.png Fitness*

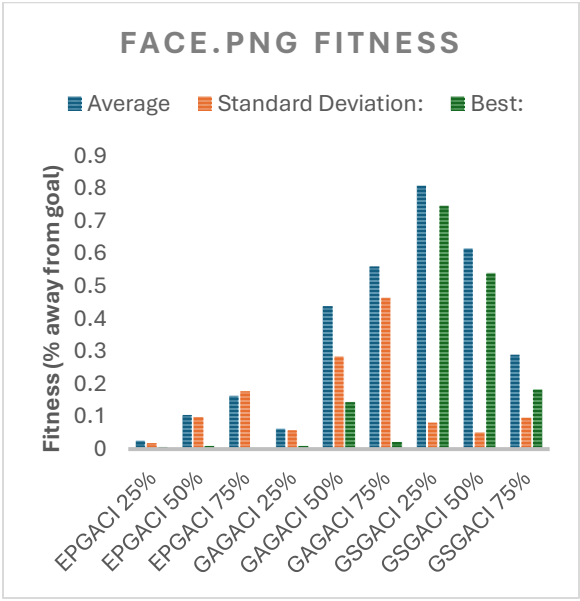



*Figure 4: Face.png Fitness*

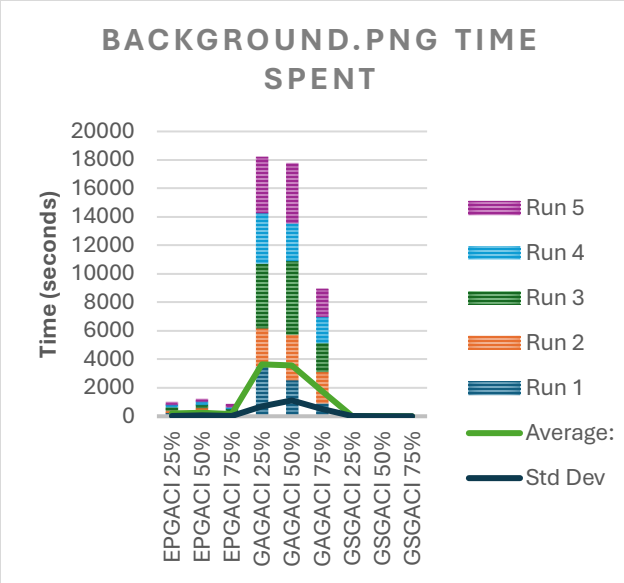



*Figure 5: Background.png Time Spent*

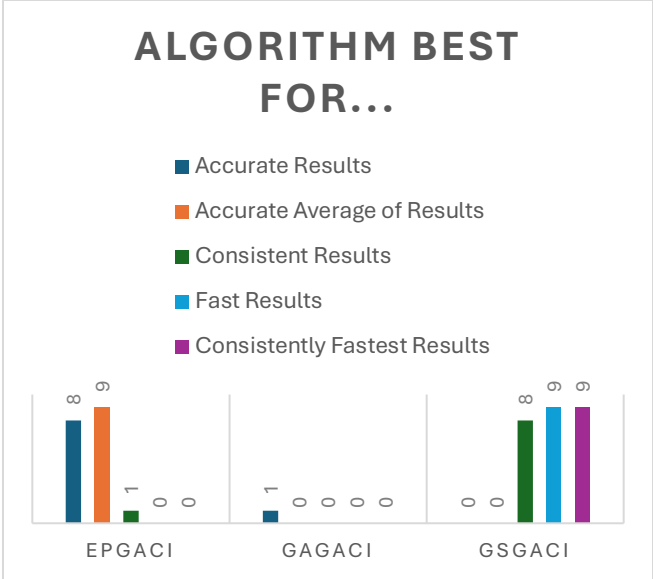



*Figure 7: Summary of Results*

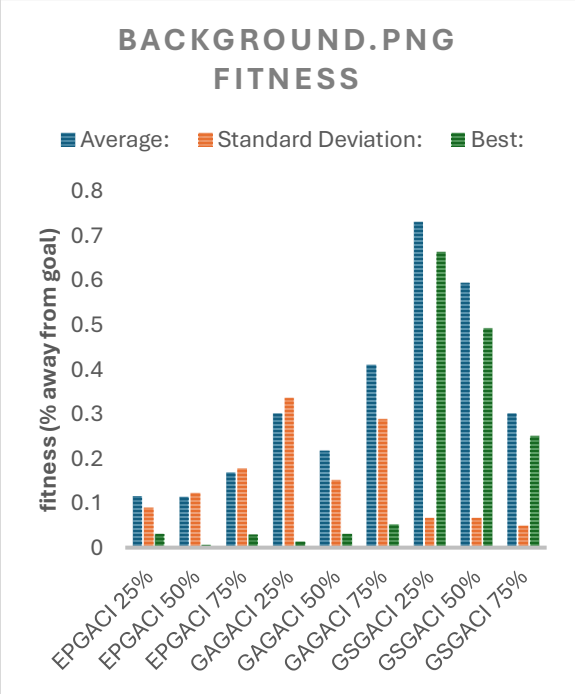



*Figure 6: Background.png Fitness*

## *Tables*

| Speed (in seconds) | EPGACI 25% | EPGACI 50% | EPGACI 75% | GAGACI 25% | GAGACI 50% | GAGACI 75% | GSGACI 25% | GSGACI 50% | GSGACI 75% |
|---|---|---|---|---|---|---|---|---|---|
| Run 1 | 106.535 | 107.7 | 119.023 | 2161.59 | 1069.689 | 966.068 | 0.088 | 0.087 | 0.07 |
| Run 2 | 146.149 | 133.7 | 114.249 | 1923.204 | 1863.056 | 591.674 | 0.086 | 0.066 | 0.067 |
| Run 3 | 105.927 | 106.8 | 115.271 | 2170.024 | 1428.357 | 451.743 | 0.082 | 0.076 | 0.065 |
| Run 4 | 109.529 | 100.9 | 107.905 | 1655.804 | 1397.87 | 902.819 | 0.073 | 0.077 | 0.059 |
| Run 5 | 120.307 | 177.0 | 106.192 | 2015.668 | 1273.518 | 1014.563 | 0.086 | 0.067 | 0.058 |
| Average | 117.689 | 125.25 | 112.528 | 1985.258 | 1406.498 | 785.3734 | 0.083 | 0.0746 | 0.0638 |
| Standard Deviation | 16.9275 | 31.5735 | 5.34293 | 211.2740 | 291.5996 | 248.8978 | 0.006 | 0.008561 | 0.005167 |

*Table 1: Man.png Speed*

| Fitness | EPGACI 25% | EPGACI 50% | EPGACI 75% | GAGACI 25% | GAGACI 50% | GAGACI 75% | GSGACI 25% | GSGACI 50% | GSGACI 75% |
|---|---|---|---|---|---|---|---|---|---|
| Run 1 | 0.16356 | 0.27305 | 0.01406 | 0.059492 | 0.013221 | 0.385429 | 0.72986 | 0.43914 | 0.23567 |
| Run 2 | 0.02704 | 0.04350 | 0.08039 | 0.180635 | 0.140129 | 0.368354 | 0.81880 | 0.55379 | 0.23543 |
| Run 3 | 0.24721 | 0.22425 | 0.05444 | 0.068624 | 0.530967 | 0.860145 | 0.81736 | 0.55115 | 0.20755 |
| Run 4 | 0.22053 | 0.08004 | 0.18881 | 0.144819 | 0.043746 | 0.770011 | 0.86182 | 0.43025 | 0.20659 |
| Run 5 | 0.08665 | 0.00745 | 0.24938 | 0.841396 | 0.218254 | 0.340957 | 0.86327 | 0.43145 | 0.20779 |
| Average | 0.14900 | 0.12566 | 0.11741 | 0.258993 | 0.189263 | 0.544979 | 0.81822 | 0.48116 | 0.21861 |
| Standard Deviation | 0.09183 | 0.11645 | 0.09814 | 0.329548 | 0.207474 | 0.249121 | 0.05417 | 0.06519 | 0.01547 |
| Best | 0.02704 | 0.00745 | 0.01406 | 0.059492 | 0.013221 | 0.340957 | 0.729867 | 0.43025 | 0.20659 |

*Table 2: Man.png Fitness*

| Speed (in seconds) | EPGACI 25% | EPGACI 50% | EPGACI 75% | GAGACI 25% | GAGACI 50% | GAGACI 75% | GSGACI 25% | GSGACI 50% | GSGACI 75% |
|---|---|---|---|---|---|---|---|---|---|
| Run 1 | 88.58 | 43.368 | 132.692 | 777.275 | 352.678 | 326.736 | 0.047 | 0.075 | 0.046 |
| Run 2 | 54.482 | 51.556 | 40.524 | 894.56 | 326.607 | 136.452 | 0.041 | 0.036 | 0.038 |
| Run 3 | 60.33 | 48.052 | 39.039 | 884.99 | 458.69 | 162.256 | 0.041 | 0.036 | 0.032 |
| Run 4 | 77.364 | 48.514 | 53.169 | 479.319 | 372.401 | 241.714 | 0.042 | 0.032 | 0.032 |
| Run 5 | 52.604 | 76.211 | 46.281 | 546.766 | 280.453 | 384.394 | 0.042 | 0.039 | 0.042 |
| Average | 66.672 | 53.5402 | 62.341 | 716.582 | 358.1658 | 250.3104 | 0.0426 | 0.0436 | 0.038 |
| Standard Deviation | 15.6604 | 13.0072 | 39.7186 | 192.9068 | 65.90354 | 105.6059 | 0.002509 | 0.017728 | 0.006164 |

*Table 3: Face.png Speed*

| Fitness | EPGACI 25% | EPGACI 50% | EPGACI 75% | GAGACI 25% | GAGACI 50% | GAGACI 75% | GSGACI 25% | GSGACI 50% | GSGACI 75% |
|---|---|---|---|---|---|---|---|---|---|
| Run 1 | 0.00970 | 0.25028 | 0.00173 | 0.05498 | 0.42543 | 0.98596 | 0.74563 | 0.53888 | 0.18485 |
| Run 2 | 0.02463 | 0.16221 | 0.13410 | 0.15947 | 0.58764 | 0.84863 | 0.74911 | 0.65034 | 0.18236 |
| Run 3 | 0.04652 | 0.04975 | 0.44657 | 0.06841 | 0.83246 | 0.84714 | 0.75160 | 0.65134 | 0.36149 |
| Run 4 | 0.00721 | 0.05523 | 0.03458 | 0.02562 | 0.14430 | 0.02164 | 0.89839 | 0.65283 | 0.35900 |
| Run 5 | 0.04254 | 0.00995 | 0.20276 | 0.01069 | 0.20102 | 0.09429 | 0.89739 | 0.58068 | 0.36000 |
| Average | 0.02612 | 0.10548 | 0.16395 | 0.06383 | 0.43817 | 0.55953 | 0.80842 | 0.61481 | 0.28954 |
| Standard Deviation | 0.01813 | 0.09866 | 0.17700 | 0.05815 | 0.28309 | 0.46203 | 0.08169 | 0.05237 | 0.09671 |
| Best | 0.00721 | 0.00995 | 0.00173 | 0.01069 | 0.14430 | 0.02164 | 0.74563 | 0.53888 | 0.18236 |

*Table 4: Face.png Fitness*

| Speed (in seconds) | EPGACI 25% | EPGACI 50% | EPGACI 75% | GAGACI 25% | GAGACI 50% | GAGACI 75% | GSGACI 25% | GSGACI 50% | GSGACI 75% |
|---|---|---|---|---|---|---|---|---|---|
| Run 1 | 230.449 | 190.368 | 161.499 | 3465.894 | 2516.595 | 911.332 | 0.188 | 0.114 | 0.092 |
| Run 2 | 186.148 | 392.631 | 154.992 | 2690.681 | 3229.123 | 2210.987 | 0.108 | 0.118 | 0.085 |
| Run 3 | 226.165 | 229.18 | 183.286 | 4546.037 | 5138.881 | 2002.126 | 0.117 | 0.12 | 0.086 |
| Run 4 | 169.75 | 219.631 | 157.66 | 3566.152 | 2650.526 | 1866.046 | 0.113 | 0.102 | 0.079 |
| Run 5 | 180.694 | 219.617 | 198.484 | 3964.788 | 4214.101 | 1960.355 | 0.111 | 0.103 | 0.1 |
| Average | 198.641 | 250.285 | 171.184 | 3646.710 | 3549.845 | 1790.169 | 0.1274 | 0.1114 | 0.0884 |
| Standard Deviation | 27.7586 | 80.8981 | 18.9118 | 682.4840 | 1111.925 | 507.2200 | 0.034033 | 0.008414 | 0.007956 |

*Table 5: Background.png Speed*

| Fitness | EPGACI 25% | EPGACI 50% | EPGACI 75% | GAGACI 25% | GAGACI 50% | GAGACI 75% | GSGACI 25% | GSGACI 50% | GSGACI 75% |
|---|---|---|---|---|---|---|---|---|---|
| Run 1 | 0.03009 | 0.04118 | 0.02908 | 0.51137 | 0.20508 | 0.63124 | 0.66334 | 0.65443 | 0.25047 |
| Run 2 | 0.14188 | 0.00554 | 0.33369 | 0.16255 | 0.36563 | 0.47270 | 0.66452 | 0.65376 | 0.25283 |
| Run 3 | 0.03109 | 0.02975 | 0.06320 | 0.78152 | 0.11380 | 0.71949 | 0.79665 | 0.58685 | 0.32041 |
| Run 4 | 0.24442 | 0.22206 | 0.39017 | 0.01361 | 0.37017 | 0.17869 | 0.79648 | 0.58668 | 0.32074 |
| Run 5 | 0.12826 | 0.27014 | 0.02908 | 0.03294 | 0.03093 | 0.05178 | 0.73193 | 0.49187 | 0.36277 |
| Average | 0.11515 | 0.11374 | 0.16904 | 0.30040 | 0.21712 | 0.41078 | 0.73058 | 0.59472 | 0.30144 |
| Standard Deviation | 0.08929 | 0.12269 | 0.17775 | 0.33494 | 0.15080 | 0.28743 | 0.06632 | 0.06662 | 0.04861 |
| Best | 0.03009 | 0.00554 | 0.02908 | 0.01361 | 0.03093 | 0.05178 | 0.66334 | 0.49187 | 0.25047 |

*Table 6: Background.png Fitness*

| Best for ... | EPGACI | GAGACI | GSGACI |
|---|---|---|---|
| Accurate Results | 8 | 1 | 0 |
| Accurate Average of Results | 9 | 0 | 0 |
| Consistent Results | 1 | 0 | 8 |
| Fast Results | 0 | 0 | 9 |
| Consistently Fastest Results | 0 | 0 | 9 |

*Table 7: Summary of Results*

| Best Fitness | Me.png | Face.png | Background.png |
|---|---|---|---|
| 25% | 0.027040 | 0.007215 | 0.013616562 |
| 50% | 0.007450 | 0.009952 | 0.00554657 |
| 75% | 0.014060 | 0.001739 | 0.029083252 |
| | | | |
| 25% Algorithm | EPGACI 25% | EPGACI 25% | GAGACI 25% |
| 50% Algorithm | EPGACI 50% | EPGACI 50% | EPGACI 50% |
| 75% Algorithm | EPGACI 75% | EPGACI 75% | EPGACI 75% |

*Table 8: Best Fitness*

| Best Average Fitness | Me.png | Face.png | Background.png |
|---|---|---|---|
| 25% | 0.14900 | 0.02612 | 0.11515274 |
| 50% | 0.12566 | 0.10548 | 0.11374054 |
| 75% | 0.11741 | 0.16395 | 0.16904754 |
| | | | |
| 25% Algorithm | EPGACI 25% | EPGACI 25% | EPGACI 25% |
| 50% Algorithm | EPGACI 50% | EPGACI 50% | EPGACI 50% |
| 75% Algorithm | EPGACI 75% | EPGACI 75% | EPGACI 75% |

*Table 9: Best Average Fitness*

| Best Std Dev of Fitness | Me.png | Face.png | Background.png |
|---|---|---|---|
| 25% | 0.05417 | 0.01813 | 0.06632363 |
| 50% | 0.06519 | 0.05237 | 0.06662455 |
| 75% | 0.01547 | 0.09671 | 0.04861736 |
| | | | |
| 25% Algorithm | GSGACI 25% | EPGACI 25% | GSGACI 25% |
| 50% Algorithm | GSGACI 50% | GSGACI 50% | GSGACI 50% |
| 75% Algorithm | GSGACI 75% | GSGACI 75% | GSGACI 75% |

*Table 10: Best Standard Deviation of Fitness*

| Average Speeds | EPGACI 25% | EPGACI 50% | EPGACI 75% | GAGACI 25% | GAGACI 50% | GAGACI 75% | GSGACI 25% | GSGACI 50% | GSGACI 75% |
|---|---|---|---|---|---|---|---|---|---|
| Me.png | 117.6894 | 125.2512 | 112.528 | 1985.258 | 1406.498 | 785.3734 | 0.083 | 0.0746 | 0.0638 |
| Face.png | 66.672 | 53.5402 | 62.341 | 716.582 | 358.1658 | 250.3104 | 0.0426 | 0.0436 | 0.038 |
| Background.png | 198.6412 | 250.2854 | 171.1842 | 3646.7104 | 3549.8452 | 1790.1692 | 0.1274 | 0.1114 | 0.0884 |

*Table 11: Average Speeds*

| Std Dev of Speeds | EPGACI 25% | EPGACI 50% | EPGACI 75% | GAGACI 25% | GAGACI 50% | GAGACI 75% | GSGACI 25% | GSGACI 50% | GSGACI 75% |
|---|---|---|---|---|---|---|---|---|---|
| Me.png | 16.92754769 | 31.57354088 | 5.342937862 | 211.2740247 | 291.5996607 | 248.8978614 | 0.006 | 0.008561542 | 0.005167204 |
| Face.png | 15.6604998 | 13.00729508 | 39.71862623 | 192.906887 | 65.90354466 | 105.6059833 | 0.00250998 | 0.017728508 | 0.006164414 |
| Background.png | 27.75868863 | 80.89815444 | 18.91181554 | 682.4840901 | 1111.925612 | 507.2200275 | 0.034033807 | 0.008414274 | 0.00795613 |

*Table 12: Standard Deviation of Speeds*