Improving Image Filter Efficiency: A Multi-objective Genetic Algorithm Approach to Optimize Computing Efficiency

Julien Biau[1], Sylvain Cussat-Blanc[2](✉), and Hervé Luga[2]

[1] INIA SAS, Toulouse, France
julien.biau@gmail.com
[2] University of Toulouse, Toulouse, France
sylvain.cussat-blanc@ut-capitole.fr, herve.luga@irit.fr

Abstract. For real-time applications in embedded systems, an efficient image filter is not defined solely by its accuracy but by the delicate balance it strikes between precision and computational cost. While one approach to manage an algorithm's computing demands involves evaluating its complexity, an alternative strategy employs a multi-objective algorithm to optimize both precision and computational cost.

In this paper, we introduce a multi-objective adaptation of Cartesian Genetic Programming aimed at enhancing image filter performance. We refine the existing Cartesian Genetic Programming framework for image processing by integrating the elite Non-dominated Sorting Genetic Algorithm into the evolutionary process, thus enabling the generation of a set of Pareto front solutions that cater to multiple objectives.

To assess the effectiveness of our framework, we conduct a study using a Urban Traffic dataset and compare our results with those obtained using the standard framework employing a mono-objective evolutionary strategy. Our findings reveal two key advantages of this adaptation. Firstly, it generates individuals with nearly identical precision in one objective while achieving a substantial enhancement in the other objective. Secondly, the use of the Pareto front during the evolution process expands the research space, yielding individuals with improved fitness.

Keywords: Genetic Programming · Cartesian Genetic Programming · Multi-Objective · Genetic Improvement · Image processing · Real Time Applications · Embedded Systems

1 Introduction

When employing image filters on embedded systems with limited computing power, the challenge extends beyond precision; one must also consider the constraints of computational capacity. In the realm of real-time applications on embedded systems, an efficient image filter is one that strikes an optimal balance between fitness and computational time.

While assessing an algorithm's complexity is a common practice, involving measurements of time and estimations of worst-case scenarios, controlling the equilibrium between precision and computation time remains a nuanced challenge. An alternative approach involves the use of multi-objective algorithms to concurrently optimize precision and computation.

In this work, we introduce a multi-objective evolutionary strategy within the framework of Cartesian Genetic Programming for Image Processing for Genetic Improvement (CGP-IP-GI). Our approach accounts for both filter precision and execution time, yielding a range of high-performing solutions. The choice of solution depends on the desired fitness level or the available computing power. Additionally, our multi-objective adaptation results in solutions with improved fitness when compared to previous mono-objective studies.

This paper is organized as follows: Sect. 2 presents the state-of-the-art in multi-objective genetic algorithms, while Sect. 3 outlines the adaptation of CGP-IP-GI, incorporating the elite non-dominated sorting genetic algorithm to achieve multi-objective optimization. Section 4 details the experiments underpinning this research, and Sect. 5 offers a comprehensive presentation of the results. Finally, in Sect. 6, we present our preliminary conclusions.

2 Related Works

This section discusses prior research in the realm of multiple objective problems, with a specific focus on the field of genetic algorithms. It also introduces Cartesian Genetic Programming for Image Processing (CGP-IP) and its recent advancement for genetic enhancement (CGP-IP-GI).

2.1 Multi-objective Evolution Algorithms

In contrast to a single-objective problem, which seeks to find an optimal solution, a multi-objective problem entails the simultaneous optimization of multiple objective functions. Enhancing one aspect in a multi-objective problem may have detrimental effects on the outcomes of other objectives. As a result, multi-objective genetic algorithms aim to generate a collection of efficient solutions that belong to the Pareto-optimal front.

Mathematically, the concept of Pareto optimality can be formally defined as follows (Eq. 1). Assuming, without loss of generality, a maximization problem, and given two decision vectors a and b belonging to the decision space X, vector a is said to *dominate* vector b if and only if:

$$\begin{cases} \forall i \in [1,2,...,n] : f_i(a) >= f_i(b) \\ \exists j \in [1,2,...,n] : f_j(a) > f_j(b) \end{cases} \tag{1}$$

All decision vectors which are not dominated by any other decision vector of a given set are called *nondominated* regarding this set. If it is clear from the context which set is meant, we simply leave it out. The decision vectors that are nondominated within the entire search space are denoted as Pareto optimal and constitute the *Pareto-optimal set* or *Pareto-optimal front*.

Non-dominated Sorting Genetic Algorithm

The Non-dominated Sorting Genetic Algorithm (NSGA), originally developed by Srivas and Deb [28], follows a methodology that starts by randomly sorting the obtained solutions based on their dominance. Within each subpopulation, the algorithm computes the proximity between solutions, and selection is carried out using the roulette method. This method gives a higher probability to solutions from the first non-dominated subpopulation. New solutions are generated through a combination of crossover and mutation applied to the selected solutions, and the algorithm continues its search for solutions until a predefined stop criterion is met. NSGA ranks solutions based on their dominance and assigns precision values.

NSGA has also been employed in a study focused on optimizing Stirling thermal engines, with the goal of achieving maximum power, thermal efficiency, and minimum pressure drop [1].

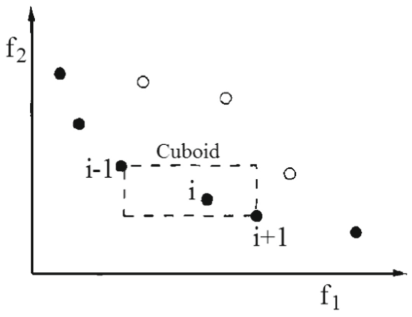


Fig. 1. The crowding distance is computed, with points identified by filled circles representing solutions belonging to the same nondominated front. (Source [6])

Elitist Non-dominated Sorting Genetic Algorithm

The elite non-dominated sorting genetic algorithm (NSGA-II), introduced by Deb and Goel [7,8], bears resemblance to NSGA. However, NSGA-II employs the crowding distance to select the most isolated results (as depicted in Figs. 1 and 2), eliminating the need to calculate the precision parameter (σ_{share}). Consequently, the algorithm ensures that the optimal Pareto solution discovered up to the current step is retained. The solution selection mechanism is employed to control population size, although this approach may diminish proximity to the optimal solution. As long as the number of solutions with the first non-dominated solutions does not exceed the number of primary populations, all solutions in this set are selected.

2.2 Cartesian Genetic Programming

Cartesian Genetic Programming (CGP) is a form of Genetic Programming (GP) in which programs are represented as directed, often acyclic graphs indexed by

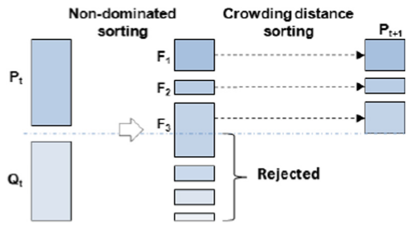


Fig. 2. Basics of NSGA-II procedure. (Source [14])

Cartesian coordinates (Fig. 3). CGP was invented by Miller and Thomson [20], [22] for use in evolving digital circuits, although it has been applied in a number of domains [23]. CGP is used in [16] to evolve neural networks, in [5,10] for object detection in image processing, and in [15] for image noise reduction. Its benefits include node neutrality, being the encoded parts of the genome that don't contribute to the interpreted program, node reuse, and a fixed representation that reduces program bloat [21]. A recent review of CGP is given in [24].

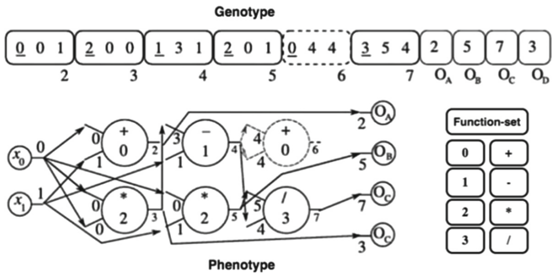


Fig. 3. A genotype in the form of a CGP (Cartesian Genetic Programming) and its associated schematic phenotype are presented for a group of four mathematical equations. The genes highlighted in the genotype dictate the function of individual nodes. The addresses for each program input and node in both the genotype and phenotype are displayed below. Inactive regions of both the genotype and phenotype are depicted as grey dashes, specifically in the case of node 6. (Source [25]). (Color figure online)

In CGP, functional nodes, defined as a set of evolved genes, connect to program inputs and to other functional nodes via their Cartesian coordinates. The outputs of the program are taken from any internal node or program input based on evolved output coordinates. The CGP nodes are organized in a rectangular grid with R rows and C columns. Nodes have the flexibility to establish connections with any node from preceding columns, and this connectivity is governed by a parameter L, representing the number of columns to which a node can connect. In this investigation, consistent with previous studies such as [24], the value of R is set to 1, indicating that all nodes reside in a single row.

The CGP genotype consists of a list of node genes; each node in the genome encodes the node function, the coordinates of the function inputs (here referred to as Connection 0 and Connection 1), and optionally, the parameters for the node function. Finally, the end of the genome encodes the nodes that give the final program output. By tracing back from these output nodes, a single function can be derived for each program output by offering a concise and legible program representation.

The genes in CGP are optimized through using the 1+λ algorithm. A population of λ individuals are randomly generated and evaluated on a test problem. The evaluation is performed by decoding the program graph from the individual genotype and applying the program to a specific problem such as image masking, as in this study. The best individual based on this evaluation is retained for the next generation. A mutation operator is applied to this individual to create λ new individuals; in CGP, the mutation operator randomly samples a subset of new genes from a uniform distribution. This new population is evaluated, and the best individual is retained for the next generation; this iterative process continues until a configured stopping criterion.

2.3 Cartesian Genetic Programming for Image Processing

An important choice to make when using CGP is the set of possible node functions. In the original circuit design, the node functions are logic gates such as AND and NOR. Applications of CGP in game playing and data analysis use a standard set of mathematical functions such as $x+y$, $x*y$, and $cos(x)$ for a node with inputs x and y. Function sets must be defined such that outputs of any node will be valid for another node; in mathematical functions, this is often guaranteed by limiting the domain and range of the functions between –1 and 1.

CGP-IP is an adaption of CGP that uses image processing functions and applies programs directly to images [10]. The inputs and outputs of the evolved functions are images that allows for consistency between node functions; each node function is defined to input an image of a fixed size and output an image of the same size. CGP-IP has previously used a set of 60 functions [10] from OpenCV, a standard open-source image processing library.

In a previous work [10], CGP-IP has used an island population distribution algorithm. In this method, multiple populations compete inside “islands" that are independent $1+\lambda$ evolutionary algorithms. A migration interval parameter dictates the frequency of expert sharing between islands, allowing for the synchronization of the best individual across islands. Island models have been shown to be good alternatives to the genetic algorithm, as they help preserve genetic diversity [30]. Their use in CGP-IP has demonstrated an improvement compared to the $1+\lambda$ algorithm.

CGP-IP individuals are evaluated by applying the evolved filter to a set of images, comparing them to target images, and computing a difference metric between the output image from the evolved filter and the target such as the mean error or Matthews Correlation Coefficient (MCC) [19]. In this paper, we use MCC, which measures the quality of binary classification and has been showed particularly adapted to classification tasks using CGP [9]. Calculations are based on the confusion matrix, which is the count of the true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN).

$$mcc = \frac{TPxTN - FP * FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \quad (2)$$

A MCC with a score of 1 corresponds to a perfect classification, 0 to a random classifier, and –1 to a fully inverted classification. Therefore, our fitness function for evolution is defined as follows:

$$fitness = 1 - mcc \quad (3)$$

The closer our fitness value is to 0, the more accurate our phenotype is.

2.4 Genetic Improvement

Genetic Improvement (GI) is a relatively recent field of software engineering research that uses search to improve existing software [26]. Using handwritten code as a starting point, GI searches the space of program variants created by applying mutation operators. The richness of this space depends on the power and expressivity of the mutation operators, which can modify existing code by changing functions or parameters, add new code, and, in some cases, remove parts of a program. Over the past decade, the GI field has greatly expanded, and current research on GI has demonstrated its many potential applications. GI has been used to fix software bugs [2,18], drastically speed up software systems [18,29], port a software system between different platforms [17], transplant code features between multiple versions of a system [27], grow new functionalities [12], and more recently to improve memory [31] and energy usage [4].

The majority of GI work uses GP to improve the programs under optimization [2,17,18,27,29]. In most methods, applying GI on a existing program is achieved by encoding the program within a GP tree and then computing the corresponding genome. GP mutation operators are applied to the encoded program to generate adjacent programs. To this end, both the program encoding and the operators must be defined to suit the initial program that is to be enhanced with additional functions to improve the functional graph. The fitness used during the evolutionary optimization of the program can be based on various metrics such as program length, efficiency, or relevance to given test cases [2,17,29].

2.5 Genetic Improvement in Cartesian Genetic Programming for Image Processing

In this section, we describe the node insertion and deletion operators designed for GI with CGP-IP [3]. In general, CGP can create a graph through the random mutation of node gene parameters such as connections or functions. However, this can be destructive when improving a given genome, because the modifications to active nodes can remove parts of the function graph. Previous research has proposed self-modifying genomes [11] which use functions which can add or remove nodes upon function execution. CGP-IP-GI propose node insertion and deletion operators during the mutation process instead of during the execution of the program. These operators are designed to maintain the active subgraph of a program, that is, they are not destructive.

The mutation operator comprises three possible operators: node insertion, node deletion, and standard parameter modification using a uniform distribution. The node operators have configurable mutation rates: r_{ins} and r_{del}. If one of these structural operators is chosen, it will be the only mutation performed; otherwise, standard mutation occurs. In this study, we use $r_{ins} = 0.1$ and $r_{del} = 0.1$ for all experiments.

3 Multi-objective Implementation in Cartesian Genetic Programming for Image Processing

To accomplish multi-objective optimization, we will employ the NSGA-II algorithm. In order to do so, it is imperative to adapt CGP-IP-GI by making adjustments to its evolutionary algorithm and integrating a Pareto front as the evaluation function.

3.1 Evolutionary Algorithm

In the typical scenario, CGP-IP-GI employs an evolutionary algorithm with a $\mu + \lambda$ strategy, where $\mu = 1$ parent generates λ children. However, to ensure that the Pareto front is adequately representative, it becomes essential to consider $\mu > 1$. Accordingly, the evolution phase is adjusted to enable the generation of λ children from μ parents.

3.2 Adapting NSGA-II for CGP-IP-GI

In order to maintain multiple candidates following each selection phase, it becomes necessary to replace the evaluation function that previously considered only one objective with the NSGA-II algorithm, which takes into account two objectives and enables the selection and retention of a set of solutions.

The NSGA-II algorithm can be divided into two phases: the first phase involves the elimination of dominated solutions, while the second phase focuses on preserving the most isolated solutions by sorting them.

Selection of Non-dominated Solutions
Solutions not belonging to the Pareto front or dominated solutions are deleted. Only non-dominated solutions are retained (Algorithm 1). A solution x $\in$ E dominates if x' $\in$ E if:

$$\forall i, f_i(x) \leq f_i(x') \text{ and } \exists i, f_i(x) < f_i(x') \tag{4}$$

with $f_i(x)$ the function of objective **i**

Sorting by Isolation Distance
If the number of non-dominated solutions is greater than the number of parents μ, it is necessary to sort and retain only μ solutions. The extreme points must be kept in each generation and, therefore, be assigned an infinite crowding distance. For each solution between the extreme points, the crowding distance between the previous solution and the next solution is calculated. Solutions are sorted by crowding distance and the μ solutions with the greatest distance are kept (Algorithm 2).

Algorithm 1: Removal of dominated solutions

```
Data: solutions is an array containing all solutions
Result: frontpareto contains the solutions on the Pareto front
solutions.sort([fitness,duration]);
for $i \leftarrow 0$ to $solutions.length - 1$ do
    front = True;
    for $j \leftarrow i + 1$ to $solutions.length$ do
        if $solutions[i].duration \geq solutions[j].duration$ then
            front = False;
        end
    end
    if front then
        frontpareto.push(solutions[i])
    end
end
frontpareto.push(solutions[solutions.length-1]);
```

Algorithm 2: Sorting solutions by crowding distance and selecting μ solutions

```
Data: solutions is an array containing all solutions
Result: parents contains $\mu$ solutions to use as parents
for $i \leftarrow 1$ to $solutions.length - 1$ do
    solutions[i].crowding = abs(solutions[i-1].fitness - solutions[i+1].fitness) +
    abs(solutions[i-1].duration - solutions[i+1].duration)
end
parents.push(solutions[0]);
parents.push(solutions[-1]);
solutions.sort(crowding);
for $i \leftarrow 0$ to $\mu - 2$ do
    parents.push(solutions[i]);
end
```

3.3 Synchronization of Islands

CGP-IP-GI uses a distribution of individuals who evolve on different islands. The islands are synchronized at a fixed interval. Synchronization involves taking the best of individuals from all the islands and implanting them as a parent individual on all the islands. The synchronization process is adapted to use the NSGA-II algorithm. During synchronization, all the individuals of each island are grouped together, and only the individuals belonging to the Pareto front are kept. If the number of individuals retained on the Pareto font is greater than μ (number of parents), a sorting by isolation distance is applied to keep only the most isolated individuals. The remaining μ individuals are implanted in the place of the previous parents on each island.

3.4 Objectives Functions

In the study presented in this paper, we will be optimizing two objectives simultaneously. The first objective pertains to result precision, and the objective function for precision is defined as: $1 - MCC$. Details on how to calculate this objective function can be found in Sect. 2.3 above. A lower value of the objective function implies better precision, and thus, the goal is to minimize it.

The second objective focuses on the execution time of the computation. To measure computation time, we employ the Python function **process_time**[1]. This function corresponds to the time required for executing the precision objective function. A smaller result of this objective function indicates a shorter computation time, and the objective here is also to minimize it.

4 Experiments

To assess this multi-objective adaptation in the context of Genetic Improvement (GI), we will apply the same experimental conditions as those outlined in a prior publication on image filter optimization using CGP-IP-GI [3]. This approach enables us to evaluate and, subsequently, compare the outcomes of the multi-objective adaptation with the results obtained in the single-objective version.

4.1 Urban Traffic Dataset

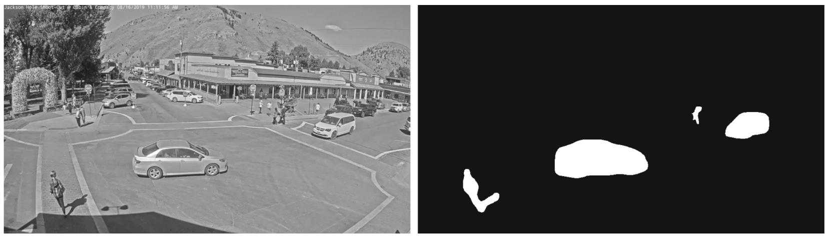


Fig. 4. Example from the urban traffic image dataset

In this study, we employed CGP-IP for the purpose of object identification within a cityscape. The dataset is sourced from urban traffic livestream cameras[2], and we generated output masks using Mask-RCNN [13]. These masks are used to isolate and retain only the relevant objects within the scene. The primary objective of this dataset is to create a filter capable of extracting and tracking specific objects in the video stream. To achieve this, the filter must discern the moving objects from one frame to the next

[1] https://docs.python.org/3/library/time.html.
[2] https://camstreamer.com/live/streams/14-traffic.

in a sequence of five-minute videos. The videos are captured in 16-bit RGB color with a resolution of 1024×576 pixels. Prior to input, we convert the images into grayscale (as depicted in Fig. 4.A). The target classification (as shown in Fig. 4.B) is aimed at identifying significant objects such as pedestrians and vehicles.

The expert filter used in this dataset was designed by engineers. It functions by subtracting the previous image from the current image and applying erode and dilate function to remove the noise. For this dataset, evolution was run for 5000 generations over forty independent trials.

4.2 CGP-IP Parameters

In this work, we have used the following parameters for CGP-IP:

- R: number of rows in CGP is 1
- C: number of columns in CGP is set to 50 but can change with the node addition and deletion operators
- r_{mut}: mutation rate for each gene is 0.25
- r_{ins}: node insertion mutation operator rate is 0.1
- r_{del}: node deletion mutation operator rate is 0.1
- number of islands: number of parallel $1 + \lambda$ evolutions is 4
- μ: number of parents on each island is 4
- λ: population size on each island is 8
- Synchronisation interval between islands: number of generations before islands that synchronize with the chromosome with the best fitness is 20
- number of generations is 5000

Each node within the graph is encoded with eight parameters. The first parameter signifies an index within the list of image processing functions. The second parameter, "Connection 0", represents a connection with a preceding node, where the output of the preceding node serves as the input for the function. The third parameter, "Connection 1", is also a connection with a previous node, using its output as input for the function (although not all functions make use of "Connection 1"). The fourth, fifth, and sixth parameters, labeled "Parameters 0", "Parameters 1", and "Parameters 2", are real numbers corresponding to the first, second, and third parameters of the function. These parameters are not necessarily used because not all functions require three parameters. For instance, Gabor Filter parameters are only utilized in conjunction with Gabor filter functions. Throughout the evolution process, mutations can occur on the function index, connections, or parameters.

4.3 Image Processing Functions

The function set employed in this study is primarily derived from the CGP-IP function set [10]. However, it's important to note that new functions have been introduced into the OpenCV library since the publication of the previous work. In addition to the pre-existing list of image processing functions [10] integrated into CGP-IP, we have incorporated two additional OpenCV functions: "watershed" and "distance transform".

Table 1. Significant values of the Pareto front at generation 5000 over 40 runs

Fitness	1.0	0.70	0.59	0.55	0.52	0.50	0.49	0.49	**0.39**	0.38	0.36	**0.35**
Duration	0.09	0.12	0.18	0.33	0.66	1.05	1.79	2.58	**2.63**	3.13	7.65	**11.7**

5 Results

In this section, we present the results obtained from our dataset over the course of 5000 generations and across 40 independent runs. The application of a multi-objective algorithm enabled us to procure a diverse range of efficient solutions catering to both objectives.

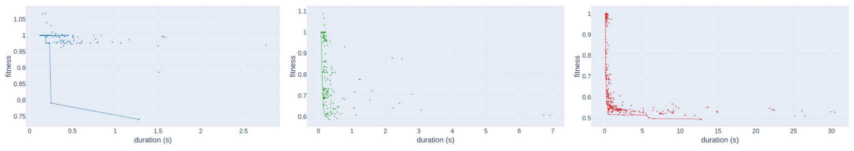


Fig. 5. All individuals over 40 runs for generation 1 (A), 100 (B) and 1000 (C). The line corresponds to Pareto front.

Figure 5 depicts the Pareto front at generation 1 in blue, generation 100 in green, and generation 1000 in red over the 40 runs. During this evolutionary progression, we observe a significant enhancement in precision from generation 1 to 1000, coinciding with a stabilization in execution time. Beyond generation 1000 and up to 5000, only four solutions exhibit noticeable precision improvements.

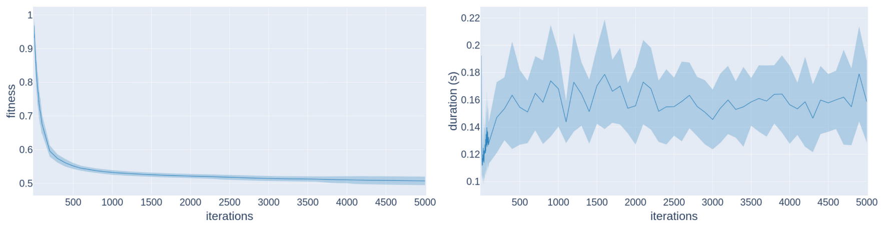
Figure 6.A showcases the evolution of the mean and standard deviation of the best precision achieved in each of the 40 runs throughout 5000 generations. This graph highlights a pronounced surge in precision up to generation 500, followed by a gradual but continuous improvement, with the standard deviation progressively increasing from generation 3500 and doubling by generation 5000.

Figure 6.B illustrates the evolution of the mean and standard deviation of the shortest execution duration in each of the 40 runs over 5000 generations. Here, we observe an increase in both mean and standard deviation up to generation 250, after which they remain at consistent levels until generation 5000.

A close examination of Figs. 6.A and 6.B uncovers the intertwined nature of precision and execution time, which stabilizes after generation 300.

Figure 7 showcases the solutions comprising the Pareto front at generation 5000, encompassing a wide array of solutions to accommodate diverse objectives. Table 1 provides a summary of representative values from this Pareto front.

Figure 8.A presents the evolution of the mean and standard deviation of the number of active nodes for the best precision achieved in each of the 40 runs throughout 5000



A: Better precision B: Shortest duration

Fig. 6. Average and standard deviation of the best precision and shortest duration over 40 runs

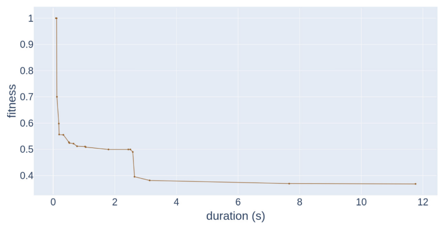


Fig. 7. Pareto front at generation 5000 over 40 runs

generations. Notably, there is a consistent rise in both the mean and standard deviation of active nodes during the entire 5000-generation span.

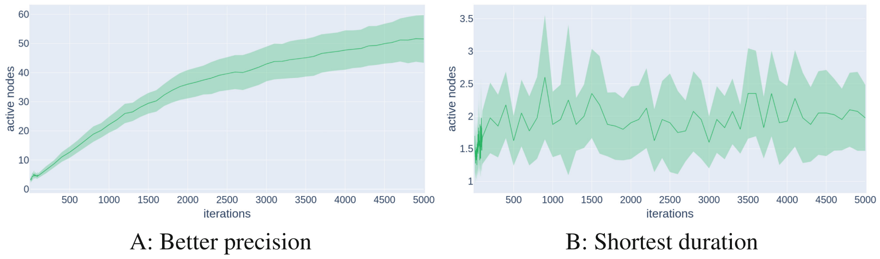
Figure 8.B mirrors this trend, capturing the evolution of the mean and standard deviation of the number of active nodes for the shortest execution duration in each of the 40 runs over 5000 generations. We observe an increase in both mean and standard deviation up to generation 200, followed by a stable trajectory up to generation 5000. The comparative analysis of Figs. 6 and 8 underscores the connection between improved objectives and an increased number of active nodes.

The Pareto front established after 5000 generations provides a broad spectrum of solutions, with precision ranging from 0.36 to 1.0 and execution times spanning from 90 milliseconds to 11.7 s (see Table 1). This diversity enables the selection of solutions based on the interplay of these two objectives. For instance, even when the highest precision achieved is **0.35**, there exists a solution offering slightly less precision (**0.39**, i.e., a 10% reduction) but with a significantly shorter execution time (**2.63** s instead of **11.7** s, representing a computational cost reduction of 78%).

5.1 Comparing Multi-objective to Mono-objective Results

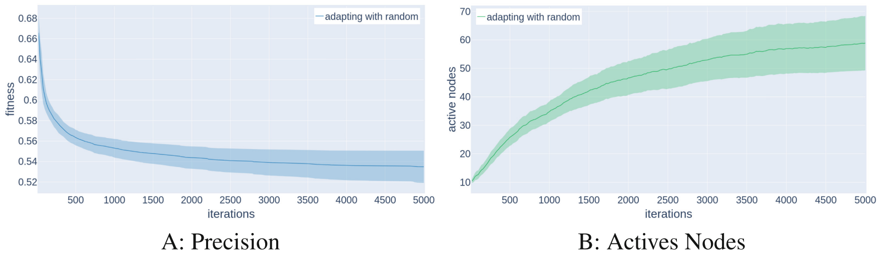
The CGP-IP-GI framework has previously been examined in a single-objective setup using the same dataset and CGP-IP parameters [3], and Fig. 9 is extracted from this prior publication.

In Fig. 9.A, we observe the evolution of both the mean and standard deviation of precision over 40 runs spanning 5000 generations. When compared to Fig. 6.A, both graphs exhibit a consistent and swift enhancement up to generation 500, followed by a



A: Better precision B: Shortest duration

Fig. 8. Average and standard deviation of the number of actives nodes of the best precision and the shortest duration over 40 runs



A: Precision B: Actives Nodes

Fig. 9. Average and standard deviation of the precision and the number of actives nodes over 40 runs from [3]

more gradual improvement. Notably, the standard deviations in both figures are quite similar. The multi-objective adaptation yields a slightly improved precision (2%) at the 5000th generation.

Moving to Fig. 9.B, it illustrates the progression of the mean and standard deviation of the number of active nodes over 40 runs during 5000 generations. In comparison with Fig. 8.A, both graphs display a parallel evolution characterized by a continuous increase in the number of active nodes and a comparable standard deviation. At the 5000th generation in Fig. 9.A, the mean value stands at 0.53, with a minimum of 0.44, a maximum of 0.59, and a standard deviation of 0.03 (as outlined in Table 2). While the mean precision in Figs. 9.A and 6.A displays a slight difference at the 5000th generation, comparing their best fitness demonstrates that the multi-objective adaptation enables the attainment of individuals with superior fitness. Significantly, the T-test p-value between Figs. 6.A and 9.A falls below $1e^{-5}$.

The results from this experiment, conducted over 40 runs and spanning 5000 generations, reveal a consistent improvement in accuracy, as well as an ongoing increase in the number of active nodes. Moreover, when compared to the outcomes of previous studies [3], our findings demonstrate that the use of a multi-objective algorithm does not compromise precision over the course of 5000 generations. On the contrary, maintaining a Pareto front during the evolution process, as opposed to a single individual, results in an expanded search space and leads to more optimal solutions.

Table 2. Comparing of multi-objective and mono-objective [3] results

Evolutionary algorithm	Mean Fitness	Min Fitness	Standard deviation
Mono-objective	0.53	0.44	0.03
Multi-objective	**0.5**	**0.35**	**0.02**

6 Conclusion

In this study, our primary objective was to investigate the feasibility of effectively managing the computational demands inherent in the genetic evolution of image filters. We pursued a multi-objective approach by enhancing the adaptability of CGP-IP-GI to accommodate multiple objectives.

The results obtained from our experiments involving this multi-objective adaptation have provided valuable insights. They offer a range of solutions that belong to a Pareto front, where a slight reduction in precision leads to a substantial reduction in computation time. Maintaining a Pareto front throughout the evolution process, rather than relying on a single individual, significantly expands the search space, thus facilitating the discovery of efficient solutions.

This study, in conjunction with comparisons to prior publications, reinforces the notion that it is indeed possible to reduce the computational time required for image filters while preserving precision. This outcome holds significant promise for the utilization of GP-based algorithms in embedded applications, freeing up computational resources for other essential tasks. Additionally, our findings underscore the efficiency of adapting the NSGA-II algorithm within genetic algorithms for genetic improvement.

In practical terms, the evolution of the CGP-IP-GI framework now enables the execution of multi-objective genetic improvement, offering a wide array of effective solutions for various objectives. The selection of the most suitable solution remains a human-driven process, aligning with the specific constraints and requirements of the application. For systems with limited computing power, the dynamic adaptation to the available computational resources from this pool of solutions holds substantial potential.

One noteworthy avenue that merits exploration is the use of multi-objective algorithms to expand the search space of single-objective algorithms. While our study primarily aimed to reduce computational time as the second objective, an intriguing prospect exists in controlling the number of active nodes. This approach could involve either minimizing or maximizing them. The selection of the most suitable secondary objective to maximize the search space warrants further investigation.

Furthermore, our findings open the door to the exploration of divergent search approaches, offering the possibility of combining directed search, computational cost optimization, and divergent search. This avenue holds promise for future research to unlock new levels of efficiency and effectiveness in the field of genetic improvement.

References

1. Ahmadi, M.H., Hosseinzade, H., Sayyaadi, H., Mohammadi, A.H., Kimiaghalam, F.: Application of the multi-objective optimization method for designing a powered stirling heat engine: Design with maximized power, thermal efficiency and minimized pressure loss. Renewable Energy **60**, 313–322 (2013) . https://doi.org/10.1016/j.renene.2013.05.005,https://www.sciencedirect.com/science/article/pii/S0960148113002504
2. Arcuri, A., Yao, X.: A novel co-evolutionary approach to automatic software bug fixing. In: 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), pp. 162–168 (2008). https://doi.org/10.1109/CEC.2008.4630793
3. Biau, J., Wilson, D., Cussat-Blanc, S., Luga, H.: Improving image filters with cartesian genetic programming. In: Proceedings of the 13th International Joint Conference on Computational Intelligence (IJCCI 2021), ECTA, vol. 1, pp. 17–27. INSTICC, SciTePress (2021). https://doi.org/10.5220/0010640000003063
4. Bruce, B.R., Petke, J., Harman, M.: Reducing energy consumption using genetic improvement. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, GECCO 2015, pp. 1327–1334. Association for Computing Machinery, New York (2015). https://doi.org/10.1145/2739480.2754752,https://doi.org/10.1145/2739480.2754752
5. Cortacero, K., et al.: Evolutionary design of explainable algorithms for biomedical image segmentation. Nat. Commun. (2023)
6. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Trans. Evol. Comput. **6**, 182–197 (2002)
7. Deb, K., Goel, T.: Controlled elitist non-dominated sorting genetic algorithms for better convergence. In: Eckart, Z., Lothar, T., Kalyanmoy, D., Artemio, C.C., David, C. (eds.) Evolutionary Multi-Criterion Optimization, pp. 67–81. Springer, Berlin (2001). https://doi.org/10.1007/3-540-44719-9_5
8. Deb, K., Goel, T.: A hybrid multi-objective evolutionary approach to engineering shape design. In: Eckart, Z., Lothar, T., Kalyanmoy, D., Artemio, C.C., David, C. (eds.) Evolutionary Multi-Criterion Optimization, pp. 385–399. Springer, Berlin (2001). https://doi.org/10.1007/3-540-44719-9_27
9. Harding, S., Graziano, V., Leitner, J., Schmidhuber, J.: Mt-cgp: Mixed type cartesian genetic programming. In: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation, GECCO 2012, pp. 751–758. Association for Computing Machinery, New York (2012). https://doi.org/10.1145/2330163.2330268,https://doi.org/10.1145/2330163.2330268
10. Harding, S., Leitner, J., Schmidhuber, J.: Cartesian Genetic Programming for Image Processing, pp. 31–44. Springer, New York (2013). https://doi.org/10.1007/978-1-4614-6846-2_3
11. Harding, S.L., Miller, J.F., Banzhaf, W.: Self-Modifying Cartesian Genetic Programming, pp. 101–124. Springer, Berlin (2011). https://doi.org/10.1007/978-3-642-17310-3_4
12. Harman, M., Jia, Y., Langdon, W.B.: Babel pidgin: Sbse can grow and graft entirely new functionality into a real world system. In: Le Goues, C., Yoo, S. (eds.) Search-Based Software Engineering, pp. 247–252. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-09940-8_20
13. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn (2018)
14. Jafarian, F., Amirabadi, H., Sadri, J.: Application of multi-objective optimization algorithm and artificial neural networks at machining process (March 2013)
15. Kalkreuth, R., Rudolph, G., Krone, J.: More efficient evolution of small genetic programs in Cartesian Genetic Programming by using genotypie age. In: 2016 IEEE Congress on Evolutionary Computation (CEC), pp. 5052–5059. IEEE (Jul 2016). https://doi.org/10.1109/CEC.2016.7748330,https://ieeexplore.ieee.org/document/7748330/

16. Khan, G.M., Miller, J.F., Halliday, D.M.: Evolution of cartesian genetic programs for development of learning neural architecture. Evol. Comput. **19**(3), 469–523 (2011) https://doi.org/10.1162/EVCO_00043
17. Langdon, W.B., Harman, M.: Evolving a cuda kernel from an nvidia template. In: IEEE Congress on Evolutionary Computation, pp. 1–8 (2010). https://doi.org/10.1109/CEC.2010.5585922
18. Langdon, W.B., Harman, M.: Optimizing existing software with genetic programming. IEEE Trans. Evol. Comput. **19**(1), 118–135 (2015). https://doi.org/10.1109/TEVC.2013.2281544
19. Matthews, B.: Comparison of the predicted and observed secondary structure of t4 phage lysozyme. Biochimica et Biophysica Acta (BBA) - Protein Structure **405**(2), 442–451 (1975). https://doi.org/10.1016/0005-2795(75)90109-9,https://www.sciencedirect.com/science/article/pii/0005279575901099
20. Miller, J.F.: An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In: Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation, GECCO 1999, vol. 2, pp. 1135–1142. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1999)
21. Miller, J.F.: What bloat? cartesian genetic programming on boolean problems. In: 2001 Genetic and Evolutionary Computation Conference Late Breaking Papers, pp. 295–302 (2001). https://www.elec.york.ac.uk/intsys/users/jfm7/gecco2001Late.pdf
22. Miller, J.F., Thomson, P.: Cartesian genetic programming. In: Genetic Programming. pp. 121–132. Springer, Berlin Heidelberg (2000). doi: https://doi.org/10.1007/978-3-642-17310-3_2
23. Miller, J.F.: Cartesian genetic programming. Springer (2011)
24. Miller, J.F.: Cartesian genetic programming: its status and future. Genetic Program. Evolvable Mach. 1–40 (2019)
25. Miragaia, R., Fernández, F., Reis, G., Inácio, T.: Evolving a multi-classifier system for multi-pitch estimation of piano music and beyond: an application of cartesian genetic programming. Appl. Sci. **11**(7), 2902 (2021)
26. Petke, J., Haraldsson, S.O., Harman, M., Langdon, W.B., White, D.R., Woodward, J.R.: Genetic improvement of software: a comprehensive survey. IEEE Trans. Evol. Comput. **22**(3), 415–432 (2018). https://doi.org/10.1109/TEVC.2017.2693219
27. Petke, J., Harman, M., Langdon, W.B., Weimer, W.: Using genetic improvement and code transplants to specialise a c++ program to a problem class. In: Nicolau, M., et al. (eds.) Genetic Programming, pp. 137–149. Springer, Berlin Heidelberg, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44303-3_12
28. Srinivas, N., Deb, K.: Muiltiobjective optimization using nondominated sorting in genetic algorithms. Evol. Comput. **2**(3), 221–248 (1994). https://doi.org/10.1162/evco.1994.2.3.221
29. White, D.R., Arcuri, A., Clark, J.A.: Evolutionary improvement of programs. IEEE Trans. Evol. Comput. **15**(4), 515–538 (2011). https://doi.org/10.1109/TEVC.2010.2083669
30. Whitley, D., Rana, S., Heckendorn, R.: The island model genetic algorithm: On separability, population size and convergence. J. Comput. Inform. Technol. **7** (1998)
31. Wu, F., Weimer, W., Harman, M., Jia, Y., Krinke, J.: Deep parameter optimisation. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, GECCO 2015, pp. 1375–1382. Association for Computing Machinery, New York (2015). https://doi.org/10.1145/2739480.2754648