Genetic Programming: From design to improved implementation

Victor R. López-López, Leonardo Trujillo, Pierrick Legrand, Gustavo Olague

Genetic Programming: From Design to Improved Implementation

Víctor R. López-López, Leonardo Trujillo
Instituto Tecnológico de Tijuana
Tijuana, BC, Mexico
vlopez, leonardo.trujillo @tectijuana.edu.mx

Pierrick Legrand
Université de Bordeaux
IMB, UMR CNRS
INRIA, France
pierrick.legrand@u-bordeaux.fr

Gustavo Olague
CICESE
Ensenada, BC, Mexico
olague@cicese.mx

ABSTRACT

Genetic programming (GP) is an evolutionary-based search paradigm that is well suited to automatically solve difficult design problems. The general principles of GP have been used to evolve mathematical functions, models, image operators, programs, and even antennas and lenses. Since GP evolves the syntax and structure of a solution, the evolutionary process can be carried out in one environment and the solution can then be ported to another. However, given the nature of GP it is common that the evolved designs are unorthodox compared to traditional approaches used in the problem domain. Therefore, efficiently porting, improving or optimizing an evolved design might not be a trivial task. In this work we argue that the same GP principles used to evolve the solution can then be used to optimize a particular new implementation of the design, following the Genetic Improvement approach. In particular, this paper presents a case study where evolved image operators are ported from Matlab to OpenCV, and then the source code is optimized an improved using Genetic Improvement of Software for Multiple Objectives (GISMOE). In the example we show that functional behavior is maintained (output image) while improving non-functional properties (computation time). Despite the fact that this first example is a simple case, it clearly illustrates the possibilities of using GP principles in two distinct stages of the software development process, from design to improved implementation.

Keywords

Genetic programming; genetic improvement; computer vision

1. INTRODUCTION

Over the past twenty years or so, Genetic Programming (GP) has established itself as one of the most successful evolutionary computation paradigms [5, 10]. In one sense, GP is similar to other supervised machine learning algorithms, such as neural networks, support vector machines or decision trees, where given a dataset of known inputs and expected outputs these algorithms derive models that best fit, or help explain, the data. The similarities are evident when we consider that the most common application domains for all of these algorithms, including GP, are quite similar, namely regression and classification, probably the most common supervised learning tasks in machine learning. Indeed, the success of standard GP[1] has lead researchers to develop even more powerful GP-based approaches for these type of tasks, including the Fast Function Extraction algorithm (FFX) [11] and Geometric Semantic Genetic Programming (GSGP) [24]. These algorithms offer powerful modeling techniques that are competitive with, and many times outperform, traditional machine learning algorithms [1].

However, the GP approach is much more general than the picture painted above. In a broader sense, we can say that GP is an evolutionary approach to solve design problems, where the goal of the design can be a mathematical (e.g., a real valued mapping for symbolic regression), or a much higher-level construct such as a computer vision algorithm [13, 20], an antenna [4] or even a quantum computer algorithm [19]. Indeed, in many design problems GP has often outperformed man made designs, evolving what are now commonly referred to as human-competitive results [6]. In this perspective, GP can be seen as an automated evolutionary-based process for computer aided design [13], where the search returns promising new solutions to design problems, particularly useful for problems where human experts have reached a plateau of sorts and progress has stagnated. The designs produced by a GP search can then be used by human experts based on what they require. For instance, we can use GP as a completely automated process where the result of evolution is basically the best solution (or a set of the best solutions) found by GP, which is returned to the user and applied to the problem *as is*.

[1]We will use the terms GP and standard GP to refer to the most common and original variant of genetic programming, as proposed by John Koza, using a tree representation and syntactic search operators such as subtree mutation and crossover.

On the other hand, a more nuanced approach is also possible, where the result of evolution is seen as an initial design, that can then be analyzed and possibly modified and improved by the final user. Moreover, the solution can then be ported to a new system or scenario that might be different than the one in which it was originally evolved. Suppose, for example, that a GP algorithm is used to evolve a controller for a simulated robot, where both the GP algorithm and the simulated robot are implemented using Java and compatible libraries. Lets imagine that the evolved controller achieves *state-of-the-art* performance (not uncommon in GP literature [6]), outperforming all other known strategies in this particular domain. The performance is so strong, that someone might want to use the evolved design in another simulation environment, with very different software or hardware requirements (think, for instance, a system that needs to support CUDA or FPGA hardware). In essence, this should be entirely possible, since GP returns the syntactic structure of the final design, which can be modified and implemented as the user requires.

What is interesting in this perspective, is that the product of GP evolution is a design that a human expert (scientist or engineer) can then implement in a variety of ways. In particular, as the example above indicates, we are interested in a scenario where we want to port a GP solution that was evolved in one environment to another. This is of course another programming problem, where the solution is known and only the implementation is missing. It is normally assumed that there is no longer a need for an evolutionary search in this step. However, while this assumption might be true in symbolic regression or classification, it might not hold when the product of evolution is a more complex or higher-level process, possibly a new algorithm. Given that what GP produces may be quite different to traditional solutions for a particular problem domain, it may not be entirely evident how this newly evolved design should be ported to a new programming language or system. In particular, it may not be straightforward to determine what is the most efficient implementation for this new solution, such that the design retains its high-fitness behavior while also exploiting the specific features and characteristics of the new computing environment where it is required. Luckily, GP has been recently expanded to address this kind of task, in what is known as Genetic Improvement (GI) [9, 2, 8, 15, 25], allowing us to use the same basic principles of GP in this new task. In GI the idea is to start with a complete and functional implementation of a specific algorithm or software system, which is written in a particular programming language and which is expected to run on a specific computing device. It is assumed that this implementation is not optimal in some way, so the goal is to use an evolutionary search process to modify the code in such a way that the main functional properties of the software are maintained but that one or several non functional properties are improved. Previous works in GI have shown great promise, greatly improving non functional properties such as computational cost or energy consumption while at the same time maintaining the code completely functional and able to solve the original task for which it was designed.

1.1 Proposal

In this work, we intend to put forth a software design cycle that integrates GP in both of the stages discussed above, namely: (1) evolving the solution to a design problem and (2) optimizing the implementation of the solution in a new system, platform or environment.

First, GP is used to solve a difficult design problem, using a specific formulation and implementation. Second, a human programmer ports the evolved solution to a new software system. In this step, given that the evolved solution may be *unorthodox*, it is reasonable to assume that the human programmer might not necessarily develop the most efficient or effective implementation. Therefore, we envision a third step in the process where GP is used to optimize and/or enhance the previously evolved solution for this new environment, following the general GI approach.

In particular, this work considers as a first case study a computer vision (CV) problem, where GP has been able to produce competitive operators for interest point detection in digital images [21, 13, 14]. The general overview of our approach is summarized in Figure 1.

The remainder of this paper proceeds as follows. Section 2 presents the local feature extraction problem in CV. Then, in Section 3 we discuss how this problem has been addressed using GP. Section 4 describes the problem statement of the case study studied in this work, while also providing a brief description of GI. Section 5 describes our experimental work and our initial results. Finally, Section 6 provides a summary of this work, draws conclusions and outlines future work.

2. COMPUTER VISION PROBLEM: LOCAL FEATURE DETECTION

CV seeks to develop artificial systems that can emulate some of the abilities exhibited by the human visual system, which can analyze and interpret visual information automatically. CV systems usually focuses on high-level tasks that include object recognition, object detection, images classification and 3D reconstruction. One way to solve theses tasks is by using local feature extraction methods, that are used to identify the most interesting or promising regions in an image before attempting to interpret all of the content from the captured scene. This section provides a short review of standard approaches towards local feature extraction in CV systems. Comprehensive surveys on this topic can be found in [12, 23]. Here, we briefly describe some of the more popular methods, particularly those that are used in the experimental work.

2.1 Interest Point Detectors

Interest points are simple point features within an image; i.e., interest points are image pixels that are salient or unique when compared with neighboring pixels. The algorithms used to detect interest points analyze the intensity patterns within local image regions and only make weak assumptions regarding the underlying structure of the observed scene. Interest points are quantitatively and qualitatively different from other points, and they usually represent only a small fraction of the total number of pixels in an image.

A measure of how salient or interesting each pixel $\mathbf{x}$ is can be obtained using a mapping of the form $K(\mathbf{x}) : \mathbb{R}^+ \rightarrow \mathbb{R}$ called an interest point operator. Applying the mapping K to an image I produces what can be called an *interest image* I^*. Afterwards, most detectors follow the same basic process: non-maxima suppression that eliminates pixels that

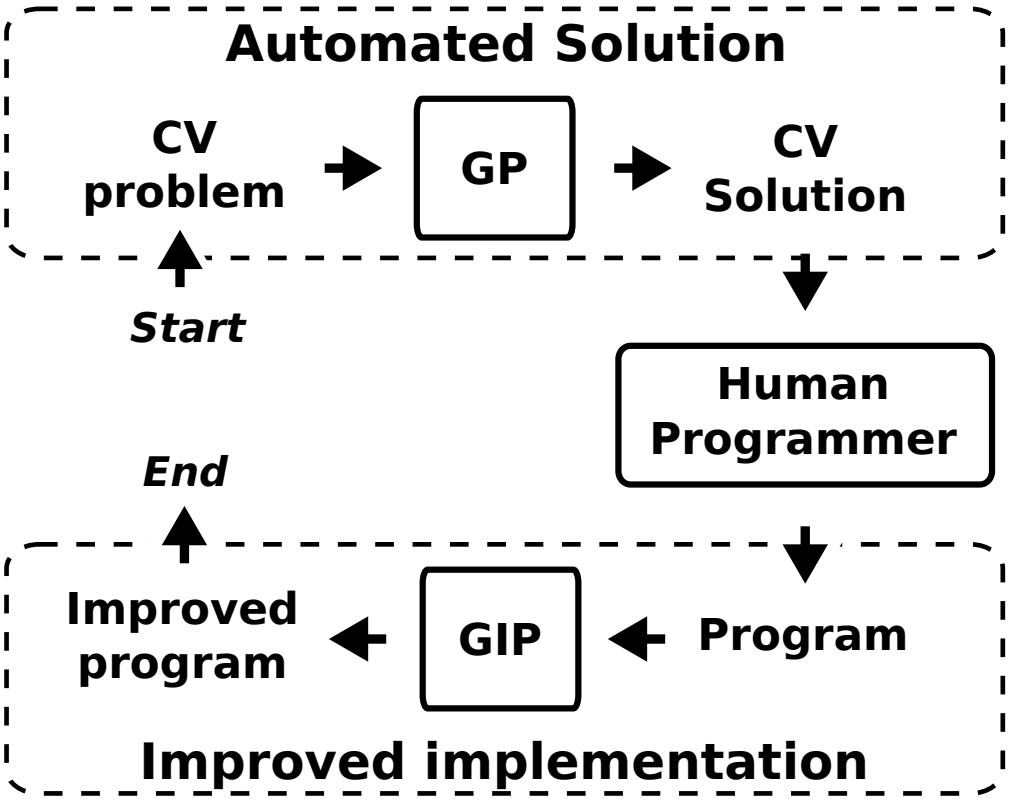


Figure 1: Conceptual diagram of the proposed approach, where GP principles are used in two distinct stages of the software development process, from evolutionary design with standard GP to an improved implementation with GI.

are not local maxima, and a thresholding step that obtains the final set of points. Therefore, a pixel $\mathbf{x}$ is tagged as an interest point if the following conditions hold,

$$K(\mathbf{x}) > max\left\{K(\mathbf{x_W})|\forall \mathbf{x_W} \in \mathbf{W}, \mathbf{x_W} \neq \mathbf{x}\right\} \wedge K(\mathbf{x}) > h \ , \quad (1)$$

where $\mathbf{W}$ is a square neighborhood of size $n \times n$ around $\mathbf{x}$, and h is an empirically defined threshold. The first condition in Equation 1 accounts for non-maximum suppression and the second is the thresholding step; Figure 2) provides a graphical illustration of the interest point detection process.

The problem of detecting interest points has been well-studied and a large variety of proposals exist in current literature. For instance, the most widely used methods employ image operators that are based on the local second-moment matrix, defined as

$$A(\mathbf{x}, \sigma_I, \sigma_D) = \sigma_D^2 \cdot G_{\sigma_I} * \begin{bmatrix} L_x^2 & L_x L_y \\ L_x L_y & L_y^2 \end{bmatrix} ,$$

where σ_D and σ_I are the differentiation and integration scales respectively, and $L_u = L_u(x, \sigma_D)$ is the Gaussian derivative in direction u of image I at pixel $\mathbf{x}$. For instance, the interest point operator used by the Harris detector [3] is

$$K_H(\mathbf{x}) = Det(A) - k \cdot Tr(A)^2, \quad (2)$$

while the Shi-Tomasi operator [17] is given by

$$K_{S-T}(\mathbf{x}) = \min\left\{\lambda_1, \lambda_2\right\} \ , \quad (3)$$

where λ_1, λ_2 are the two eigenvalues of A. Methods such as these are commonly provided as default interest point detectors in popular CV libraries such as OpenCV.

3. AUTOMATIC DESIGN OF LOCAL FEATURE DETECTORS WITH GP

In this section, we review alternative approaches towards local feature extraction, based on an automatic design methodology with GP.

It is instructive to consider that most published feature detectors are normally accompanied with experimental evidence from a particular domain that illustrates the superiority of the method under some conditions when compared with other techniques. However, such results can rarely provide assurance that a particular method will be well suited for a new or unique scenario. Therefore, researchers have performed extensive experimental evaluations and comparisons of these methods, using domain independent criteria that capture the underlying characteristics that such methods are expected to have [12, 22, 20].

Based on those works, recent contributions have followed the opposite approach, using these experimental criteria as the basis for objective functions, and then posed a search and optimization problem to automatically synthesize high-performance feature detectors [22, 21, 13, 14, 20]. The general goal is to exploit meta-heuristic and hyper-heuristic search methods to help researchers during the design process of specialized operators. However, we do not suggest that such an approach is in some way superior to a traditional

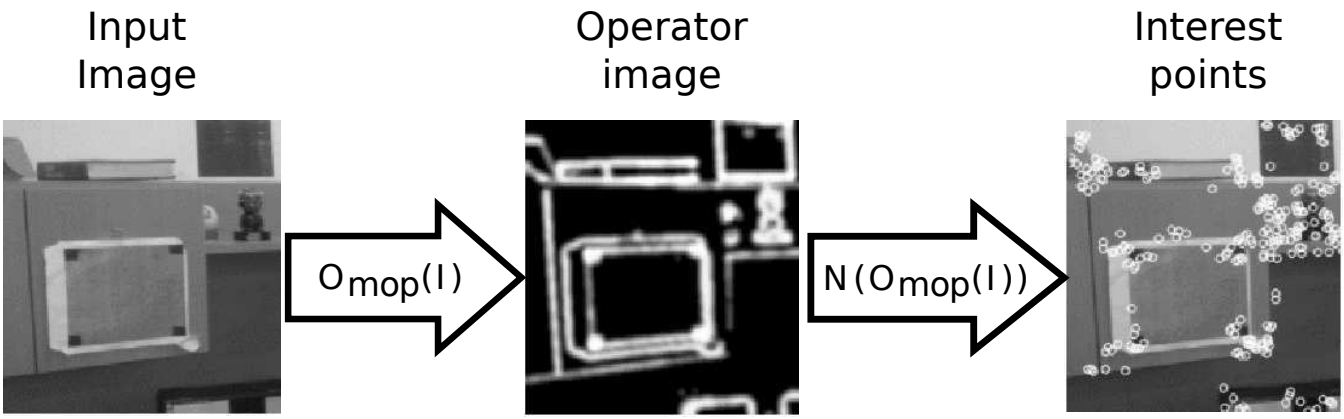


Figure 2: Example of the interest point detection process, where $O_{MOP}(I)$ is the MOP operator and the $N(\cdot)$ represents the non-maxima suppression and thresholding function.

design process. Instead, we agree with [13], hyper-heuristic searchers such as GP can produce novel designs that can assist in the development of high-performance and possibly unconventional solutions to difficult problems. In particular, this paper studies the interest point detector generated with a multi-objective GP [13, 21, 14], which is described in the following sections. But first a brief introduction to GP is provided, which is the core algorithm used to derive the operator.

3.1 Genetic Programming

Evolutionary algorithms (EA) are population-based search methods, where candidate solutions are stochastically selected and varied to produce new solutions for a specified problem. This process is carried out iteratively until a predefined termination criterion is met. In general, to apply an EA the following aspects must be defined based on domain knowledge. First, an encoding scheme to represent and manipulate candidate solutions for a given problem. Second, an evaluation or fitness function f that measures the quality of each solution based on the high-level given in the problem statement. Third, an EA applies variation operators that take one or more solutions from the population as input and produce one or more solutions as output. Fourth, solutions are chosen by the variation operators based on their fitness using a selection mechanism that prefers highly fit solutions but also encourages diversity. Finally, a survival strategy decides which individuals within the population will appear in the following iteration.

Among EAs, GP is arguably one of the most advanced techniques, since it can be used for automatic program induction and automatic design [10]. In standard GP each individual is represented by a syntax-tree, because such structures can efficiently express simple computer programs, functions, or mathematical operators. Tree nodes contain a single element from a specified finite set of primitives $P = T \bigcup F$. Leaf nodes contain elements from the set of terminals T, which normally correspond to inputs, while internal nodes contain elements from the set of functions F, which are the basic operations used to build more complex expressions. In essence, P defines the underlying search space for the evolutionary search, that contains all of the programs that can be constructed with that primitive set. An important feature of the search space is that even when a maximum depth or size limit for individual trees is enforced, normally the search space is very large but finite.

3.2 Multi-Objective Interest Point Detector

In [13, 14], a multi-objective GP was used to design the multi-objective interest point (MOP) detector, optimized based on two competing objectives, point dispersion and repeatability rate. In fact, the MOP detector was constructed by carefully analyzing the Pareto front and the Pareto-optimal set of solutions generated by the GP search. The symbolic expression of the MOP operator O_{MOP} is

$$O_{MOP}(I) = G_2 * \left| G_1 * log(G_1 * I^2) + h \cdot G_2 * |G_1 * I - I| + \frac{G_1 * I}{I} \right|^2 , \quad (4)$$

where I is the input image, G_σ denotes Gaussian smoothing filters with scale σ, $h = 0.05$ is a weight factor that controls the disparity of the detected points, and $*$ denotes convolution operation. After applying the MOP operator, non-maxima suppression and thresholding are used to select the interest points; this process is illustrated in Figure 2.

4. PROBLEM STATEMENT: IMPROVING A HUMAN WRITTEN PROGRAM WITH GP

The interest point operator described in the preceding section was evolved using Matlab and the GPLab toolbox [18], a popular open source GP library for Matlab. Afterward, the operator was ported to C++ based on the OpenCV computer vision library, probably the most widely used vision library. The C++ source code is show in Figure 3.

This operator and implementation is used as the initial case study of the proposal put forth in this paper: optimizing the new implementation of a previously evolved design produced by a GP search, using the principles of GI.

As a brief summary, GI [9, 2, 8, 15, 25] is the process of automatically improving the functional and non-functional properties of a system using GP. Langdon presents a very complete overview of GI in [7]. In the present work we have chosen to use the Genetic Improvement of Software for Multiple Objective Exploration (GISMOE) approach and the

```
GaussianBlur(im, dst, Size(9,9),1.0,1.0);
absdiff(dst,im,dst);
GaussianBlur(dst, dst, Size(9,9),2.0,2.0);
im2=dst*W;
pow(im,2,dst);
GaussianBlur(dst, dst, Size(9,9),1.0,1.0);
log(dst,dst);
GaussianBlur(dst, dst, Size(9,9),1.0,1.0);
im2=dst+im2;
GaussianBlur(im, dst, Size(9,9),1.0,1.0);
divide(dst,im,dst);
dst=dst+im2;
dst=abs(dst);
pow(dst,2,im2);
GaussianBlur(im2, im2, Size(9,9),2.0,2.0);
```

Figure 3: The MOP operator in C++ using the OpenCV library and written by a human programmer

public open-source libraries [9, 16] which can be downloaded at http://www0.cs.ucl.ac.uk/staff/ucacbbl/gismo/ in the Free Code section. As stated before, here we will use GISMOE to refine an implementation of a previously evolved image operator.

5. EXPERIMENTS AND RESULT

The experiment consists of applying the GISMOE approach to the MOP program to improve the functional and non-functional properties of our implementation. In this case, the functional property is measured by calculating the normalized cross correlation (NCC) between the output (interest) image of the modified MOP operator and the output (interest) image of the original MOP operator. The NCC is a common measure used in template matching, which is used to measure the similarity between the interest image and the expected result, the NCC is given in the range $[0, 1]$, where 1 indicates that the two images are almost identical (highly correlated) and 0 indicates that they are very different (not correlated).

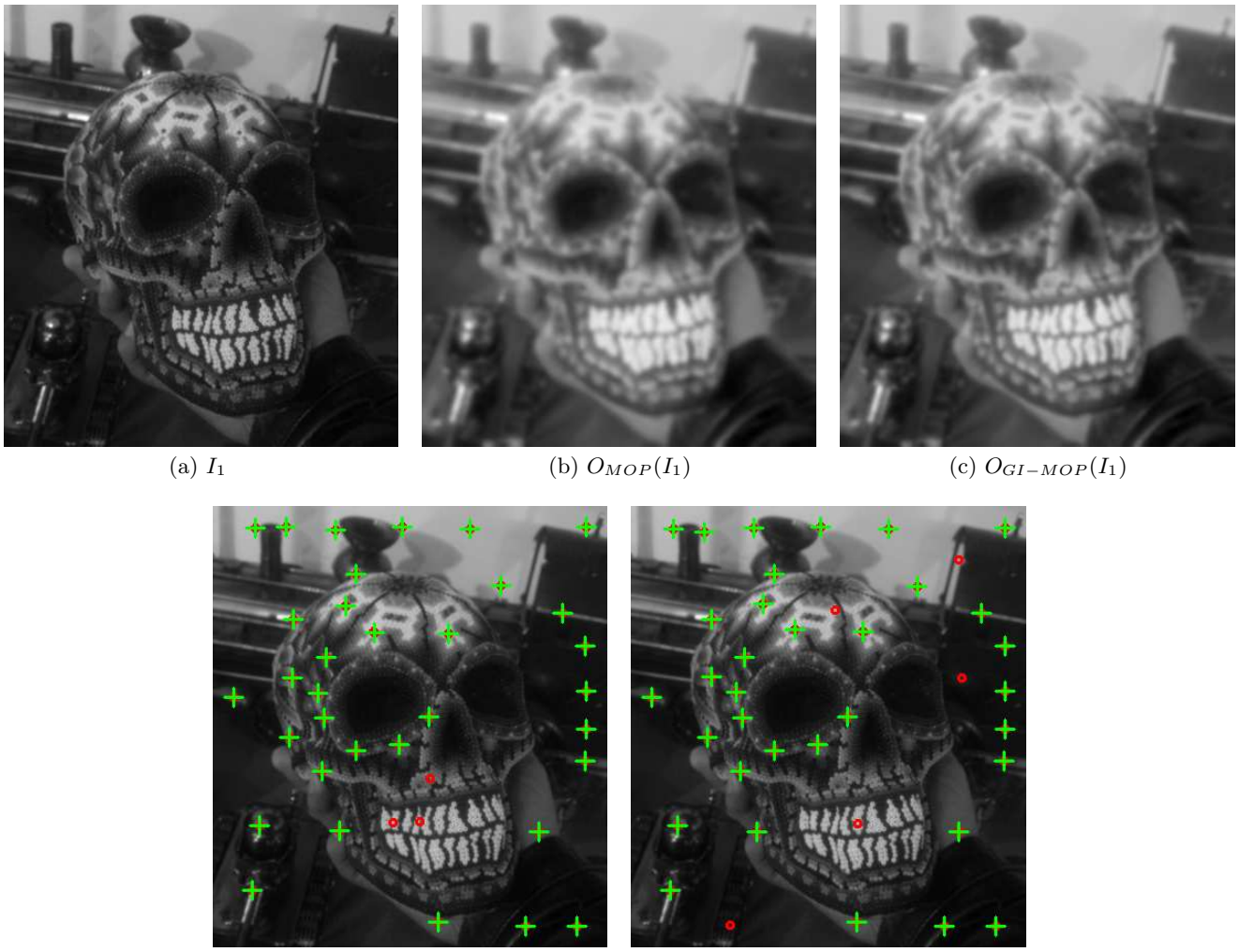
The non-functional property we measure is the related computational effort, measured by calculating the processing time of applying each operator, which is measured using the functions *getTickCount* and *getTickFrequency* provided by the OpenCV library. The MOP program was implemented in C++ using OpenCV v2.4.2 in Linux Ubuntu 12.04, and it is available in the GP for Computer Vision (GPCV) library published under the GPL license at http://www.tree-lab.org/index.php/gpcv. The GISMOE and MOP program were executed on a PC with a AMD Turion(tm) II Dual-Core Mobile M520 $\times$ 2 processor. It is important to note that the MOP operator function consists of 13 lines of code that the GISMOE approach attempts to improve. The original human implementation of the MOP operator is given in Figure 3. The MOP program was executed with $h = 0.10$.

We applied the GISMOE approach without the sensitivity analysis and the output bins due to the short amount of lines of code of the program and since only one test case was used. While using a single image as a test case seems low, it is actually quite appropriate for the interest point detection problem if we consider that a single image can have many different types of points with different local neighborhoods which are sufficiently varied. Indeed, that is the approach we took when evolving the original MOP operator as reported in [13, 14]. The test case used in GISMOE consists of a reference image I_1 shown in Figure 4(a), a grayscale image of size 384×430. I_1 is the input for the modified MOP program, which is then compared with the expected output interest image O_{MOP} (which is the same same size as I_1) of the MOP program, shown in Figure 4(b). The GISMOE parameters used in the experiment are:

- Representation: List of replacements, deletions and insertions into BNF grammar.
- Fitness: Based on compiling modified code and testing it for the NCC and processing time.
- Selection: An evolved version cannot be selected to be a parent unless it has an NCC more than a threshold $p = 0.5$ and the processing time is less than $t_{ref} = 0.1$ milliseconds. The first five are selected to have 2 children in the next generation. One child is a mutant, the second is a crossover between a selected parent and another member of the current population. Children must be different from each other and from the current population. If crossover cannot create a different child, the child is created by mutating the selected parent.
- Population: Panmictic, non-elitist, generational with 10 individuals.
- Others: Initial population of random single mutants. 50% append crossover. The 3 types of mutation (delete, replace, insert) are equally likely. No size limit is used for the evolved solutions, and the stopping criterion for the search is a maximum of 150 generations.

Our results show that the best individual generated by GISMOE achievs a NCC of 0.99 and a processing time of 0.781 milliseconds. The achieved correlation is nearly perfect, this can be visually inspected by comparing the interest image of the best individual O_{GI-MOP} with the interest image of the O_{MOP} as shown in Figure 4. We can see that there is almost no difference between Figure 4(b) and Figure 4(c), as was desired. To complement these results, another aspect to evaluate is the number of matches between the interest points detected by O_{GI-MOP} relative to the original O_{MOP}, as shown Figure 4(d) and Figure 4(e). In these figures crosses indicates the interest points that match in the same location between both interest point operators and the circles are points that did not match. The point matching of the figures was done by using the SURF descriptor method provided by the OpenCV library and the parameters of the SURF descriptor were manually tunned to obtained the best results. In both cases we can see that the majority of the interest points match, approximately 90% of the total of interest points between both images.

The improvement of the GI version is reflected in the processing time with a reduction in total computation time of 21%. This reduction is verified by computing the average processing time of 1000 executions of both the MOP program and its improved GI version. The reduction can be seen in the source code of the improved version presented in Figure 6, or in its translated mathematical expression, which is given by



(a) I_1 (b) $O_{MOP}(I_1)$ (c) $O_{GI-MOP}(I_1)$

(d) O_{MOP} interest points (e) O_{GI-MOP} interest points

Figure 4: Illustration of the test image I_1 used with GISMOE (a) and the interest image after applying the original $O_{MOP}(I_1)$ (b) operator and the resultant operator $O_{GI-MOP}(I_2)$ (c) after running GISMOE. The resulting interest points are shown in the second row: (d) for the original MOP operator and (e) for the GISMOE version. In these figures the crosses represent interest points that were detected with both operators, while circles are points that where detected by one operator but not the other.

$$O_{GI-MOP}(I) = G_2 * \left| log(I^2) + h \cdot G_2 * (G_1 * I - I) + \frac{G_1 * I}{I} \right|^2 . \quad (5)$$

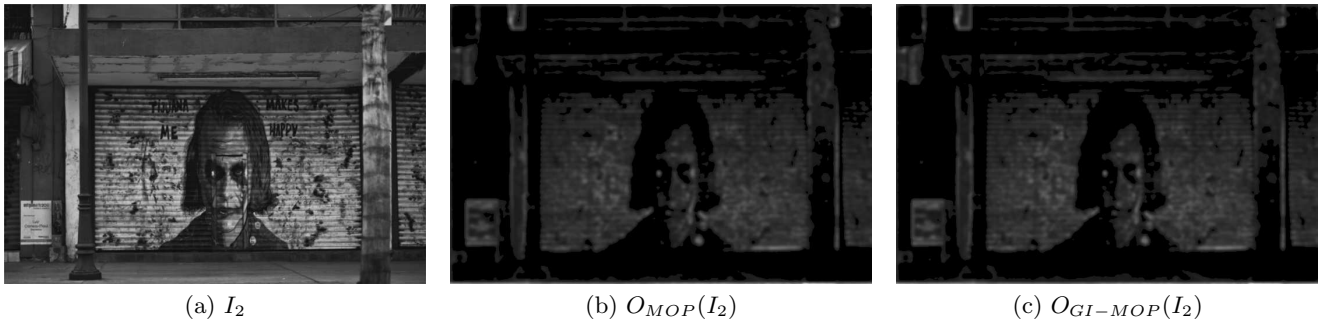
Notice that the improved individual discards one Gaussian filter G_1 from $G_1 * log(G_1 * I^2)$, a Gaussian filter G_1 from $log(G_1 * I^2)$ and the absolute value function from $|G_1 * I - I|$, all from Equation 5. Therefore by discarding 3 operations, or lines of code, we get the reported reduction in the processing time.

In addition to these results, we evaluate a second validation image I_2, with the reference image, interest images and detected interest points shown in Figure 5. As before, the NCC between the original interest image produced by O_{MOP} and the one produced by O_{GI-MOP} is 0.99, with a 95% match between the interest points detected by both image operators, as shown in Figure 5.

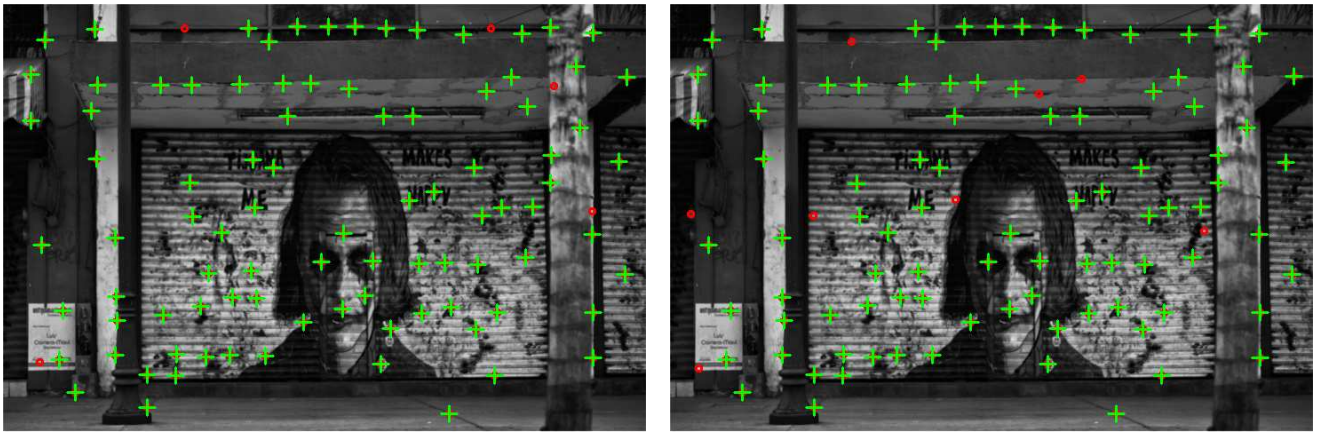
6. CONCLUSIONS

It is now becoming evident that the principles of artificial evolution could have a deep impact in the way software solutions are constructed or designed. Going beyond the automatic program induction or problem solving of traditional GP, this paper explores the possibility of using GP principles at two distinct stages of software development.

This paper provide a first, and admittedly rather simple, example of the role that GP can have in the software development process. First, GP is used to evolve a program design, a specialized operator that performs a specific task for higher-level systems. Then, the evolved design is ported to a different software system, using different programming languages and libraries, a scenario that could be quite com-



(a) I_2 (b) $O_{MOP}(I_2)$ (c) $O_{GI-MOP}(I_2)$



(d) $O_{MOP}(I_2)$ interest points (e) $O_{GI-MOP}(I_2)$ interest points

Figure 5: Illustration of a second validation image I_2 used with GISMOE (a) and the interest image after applying the original $O_{MOP}(I_2)$ (b) operator and the resultant operator $O_{GI-MOP}(I_2)$ (c) after running GISMOE. The resulting interest points are shown in the second row: (d) for the original MOP operator and (e) for the GISMOE version. In these figures the crosses represent interest points that were detected with both operators, while circles are points that where detected by one operator but not the other.

```
GaussianBlur(im, dst, Size(9,9),1.0,1.0);
absdiff(dst,im,dst);
GaussianBlur(dst, dst, Size(9,9),2.0,2.0);
im2=dst*W;
pow(im,2,dst);
log(dst,dst);
im2=dst+im2;
GaussianBlur(im, dst, Size(9,9),1.0,1.0);
divide(dst,im,dst);
dst=dst+im2;
pow(dst,2,im2);
GaussianBlur(im2, im2, Size(9,9),2.0,2.0);
```

Figure 6: The improved MOP operator in C++ using the OpenCV library.

mon in real-world domains. This task is performed by a human programmer. At this point, it is reasonable to assume that most GP-evolved solutions tend to be unorthodox in nature, particularly if we consider the possible presence of bloat or redundant code. If this assumption is true, then porting, modifying or optimizing the evolved design might not be trivially done. Here is where GP principles can be used to close the loop, applying the recently proposed GI approach to optimize the new implementation of the evolved design. GI is particularly useful in this case, where the human programmer is probably not an expert on the evolved design, indeed no one is since the design was evolved by GP and not derived analytically as most solutions are.

Our initial case study is for a CV operator for feature detection, a common and important low-level vision task that is often used in higher-level applications and provided by most CV programming libraries. The operator was evolved using a Matlab GP toolbox, relying heavily on Matlab to perform the image processing tasks. It was then ported to C++ and OpenCV, a widely used library that has become an unofficial standard in the field. We then use GISMOE to optimize our C++ implementation, achieving a 21% reduction in computation time, a substantial improvement, while maintaining the main functional behavior as evidenced by the almost perfect correlation between the expected output and the output generated by the improved version.

While the example is rather simple, it is sufficient to illustrate how two important parts of software development can be solved by using artificial evolution. The case study shows how GP can be used to evolve novel designs, and then optimize new implementations using the same general principles, from automatic design to improved implementation.

7. ACKNOWLEDGMENTS

We thank the grants SEP-TecNM (Mexico) 5620.15-P

and 5621.15-P, CONACYT (Mexico) Basic Science Research Project No. 178323, and the FP7-Marie Curie-IRSES 2013 European Commission program through project ACoBSEC with contract No. 612689. First author supported by CONACYT doctoral scholarship No. 302532.

8. REFERENCES

[1] M. Castelli, L. Trujillo, L. Vanneschi, and A. Popovič. Prediction of energy performance of residential buildings: A genetic programming approach. *Energy and Buildings*, 102:67 – 74, 2015.

[2] M. Harman, W. B. Langdon, Y. Jia, D. R. White, A. Arcuri, and J. A. Clark. The gismoe challenge: Constructing the pareto program surface using genetic programming to find better programs (keynote paper), 2012.

[3] C. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings from the Fourth Alvey Vision Conference*, volume 15, pages 147–151, 1988.

[4] G. S. Hornby, J. D. Lohn, and D. S. Linden. Computer-automated evolution of an x-band antenna for nasa's space technology 5 mission. *Evol. Comput.*, 19(1):1–23, Mar. 2011.

[5] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[6] J. R. Koza. Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*, 11(3-4):251–284, Sept. 2010.

[7] W. B. Langdon. Genetic improvement of software for multiple objectives. In M. de Oliveira Barros and Y. Labiche, editors, *Search-Based Software Engineering - 7th International Symposium, SSBSE 2015, Bergamo, Italy, September 5-7, 2015, Proceedings*, volume 9275 of *Lecture Notes in Computer Science*, pages 12–28. Springer, 2015.

[8] W. B. Langdon and M. Harman. Evolving a CUDA kernel from an nvidia template. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2010, Barcelona, Spain, 18-23 July 2010*, pages 1–8, 2010.

[9] W. B. Langdon and M. Harman. Optimizing existing software with genetic programming. *Evolutionary Computation, IEEE Transactions on*, 19(1):118–135, Feb 2015.

[10] W. B. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[11] T. McConaghy. *Genetic Programming Theory and Practice IX*, chapter FFX: Fast, Scalable, Deterministic Symbolic Regression Technology, pages 235–260. Springer New York, New York, NY, 2011.

[12] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(10):1615–1630, Oct. 2005.

[13] G. Olague and L. Trujillo. Evolutionary-computer-assisted design of image operators that detect interest points using genetic programming. *Image Vision Comput.*, 29(7):484–498, June 2011.

[14] G. Olague and L. Trujillo. Interest point detection through multiobjective genetic programming. *Appl. Soft Comput.*, 12(8):2566–2582, 2012.

[15] M. Orlov and M. Sipper. Flight of the finch through the java wilderness. *Evolutionary Computation, IEEE Transactions on*, 15(2):166–182, April 2011.

[16] J. Petke, W. B. Langdon, and M. Harman. Applying genetic improvement to minisat. In G. Ruhe and Y. Zhang, editors, *SSBSE*, volume 8084 of *Lecture Notes in Computer Science*, pages 257–262. Springer, 2013.

[17] J. Shi and C. Tomasi. Good features to track. In *Proceedings of the 1994 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94), June 1994, Seattle, WA, USA*, pages 593–600. IEEE Computer Society, 1994.

[18] S. Silva and J. Almeida. Gplab-a genetic programming toolbox for matlab. In *In Proc. of the Nordic MATLAB Conference (NMC-2003*, pages 273–278, 2005.

[19] L. Spector. *Automatic Quantum Computer Programming: A Genetic Programming Approach (Genetic Programming)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[20] L. Trujillo, P. Legrand, G. Olague, and J. LéVy-VéHel. Evolving estimators of the pointwise hölder exponent with genetic programming. *Inf. Sci.*, 209:61–79, Nov. 2012.

[21] L. Trujillo and G. Olague. Automated design of image operators that detect interest points. *Evol. Comput.*, 16(4):483–507, 2008.

[22] L. Trujillo, G. Olague, P. Legrand, and E. Lutton. A new regularity based descriptor computed from local image oscillations. *Optics Express*, 15(10):6140–6145, 2007.

[23] T. Tuytelaars and K. Mikolajczyk. Local invariant feature detectors: A survey. *Found. Trends. Comput. Graph. Vis.*, 3(3):177–280, July 2008.

[24] L. Vanneschi, M. Castelli, and S. Silva. A survey of semantic methods in genetic programming. *Genetic Programming and Evolvable Machines*, 15(2):195–214, June 2014.

[25] D. White, A. Arcuri, and J. A. Clark. Evolutionary improvement of programs. *Evolutionary Computation, IEEE Transactions on*, 15(4):515–538, Aug 2011.