Cartesian Genetic Programming for Image Processing Tasks

Héctor A Montes and Jeremy L Wyatt
{h.a.montes, j.l.wyatt}@cs.bham.ac.uk
School of Computer Science
University of Birmingham
B15 2TT, Birmingham, UK

Abstract

This paper presents experimental results on image analysis for a particular form of Genetic Programming called Cartesian Genetic Programming (CGP) in which programs use the structure of a graph represented as a linear sequence of integers. The efficency of this approach is investigated for the problem of Object Localization in a given image. This task is usually carried out by applying a series of well known image processing operators and commonly relies on the skills and expertise of the researchers. In this work, we present results from a number of runs on actual camera images, in which a set of fairly simple primitives were investigated.

Keywords: Genetic Programming, Image Processing, Object Localization

1 Introduction

The problem of localization can be associated to the well known *object recognition* problem in the computer vision field. The processing of images for object recognition usually begins with the detection of salient features, such as edges or corners. These features are typically detected as discontinuities of the intensity levels in the image, and achieved mathematically by computing derivatives of the image intensity function. The research in this area is vast and there exist a broad variety of well known methods to perform this task. Refer to [1] for a more extensive dissection.

The procedure or technique to locate and identify an interest object in a given image, commonly depends on the skills and expertise of the researchers. Usually researchers investigate which techniques are the less conditioned, possibly augment them to best suit their needs, and then develop a method to find a balance between speed and accuracy. When the task changes, it is necessary to make the proper adjustments to adapt the procedure to the new needs. Deriving automatically the process for such a complex tasks is therefore an attractive idea.

A number of previous work have reported the use of some sort of Evolutionary Algorithm for diverse computer vision tasks. This paper differs in that the programs are evolved through a form of Genetic Programming (GP) called *Cartesian Genetic Programming* (CGP). CGP was used to create a genotype in the form of a linear string of integers that represent a *directed graph*. A set of predefined inputs and functions, encoded as the graph nodes, are then sequentially executed to perform the desired *Object Localization Task*.

The paper is outlined as follows: Section 2 gives a summary of related work. Section 3 describes the type of GP used for this experiments. Section 4 describes our experiment setup. In section 5 we discuss our results and, finally, section 6 presents our conclusions.

2 Related work

The large quantity of data contained in a picture, makes visual perception one of the richest sources of information. Many researchers have taken advan-

tage of the relatively easy adaptation of evolutionary methods to many fields, and have offered a broad variety of solutions to the visual perception problem.

Tackett, for example, applied GP to the task of feature classification [14]. Yoda et al [18] propose an automatic acquisition method that consists of both dilation and erosion image operators to extract an object shape. Daida et al [3, 4] used GP to classify and extract features in remotely sensed satellite images. Van Hove and Vershoven [7] applied Genetic Algorithms (GA) for pattern recognition problems and proposed a series of appropriate genetic operators. Using GP, Johnson et al [8] evolve visual routines to find hands position in silhouettes of persons. In [11] Poli applied GP to the task of image segmentation. Ebner [5] used GP to evolve operators to extract edges from digitized images and evolved an approximation of the Moravec interest operator. In later work Ebner and Zell [6] used a similar approach to evolve control flow. Poli and Cagnoni [12] evolved algorithms for image enhancement using interactive program evolution. Teller and Veloso [17] propose a system named PADO that learns visual classification programs. In further work, PADO is also tested on the task of face recognition [16]. In [15], Teller also proposes a similar representation for PADO based on a non-random recombination scheme for the evolved programs. Work done by Brumby reported the use of GAs for feature detection in multi-spectral imaginary [2]. Finally, Martin [9] used GP to evolve programs for the visual obstacle avoidance task of a mobile robot. Unlike CGP, most of these works have used the typical GP tree representation for their evolved programs.

CGP has a number of similarities with other graph-based GP systems, such as PDGP [11] or PADO [17, 15]. The primary purpose of our experiments is to show that simpler representations in conjunction with a fairly simple set of functions and simple Genetic Algorithms, have a good performance on complex visual tasks.

3 Cartesian Genetic Programming

CGP was first proposed by Miller and Thomson in [10]. In CGP, unlike standard GP, a program is encoded as a linear string of integers that represent a directed graph. An interesting characteristic of CGP is that the population of strings are of fixed length, whereas their corresponding graphs are of variable length depending on the number of genes in use.

Reasons for selecting CGP for For the experiments reported here include:

- More powerful program encoding using graphs than using conventional GP tree-like representations.

- Simple graph representation mapped as linear strings of integers.

- Efficient evaluation derived from the intrinsic feature of *subgraph-reuse* exhibited by graphs.

- Less complicated graph recombination via the crossover and mutation genetic operators.

In CGP, unlike standard GP, a program is encoded as a linear string of integers that represent a directed graph. Each CGP program P is divided into N subsets (genes) of the same length representing the nodes of the graph. The last number assigned to each node N_i, represents a function taken from a predefined list of functions F specifically selected or designed for the problem being treated. The remaining numbers in each node, are associated either with a set I of inputs in the form of standard GP terminals, or with other nodes in the same program. CGP programs may have as many output nodes as necessary, and their corresponding number labels, are also part of the final string.

3.1 CGP programs

Consider the CGP program shown in Figure 1. Assume that a set of four inputs and a set of three functions have been already defined for this example. All elements in both input and function sets have each
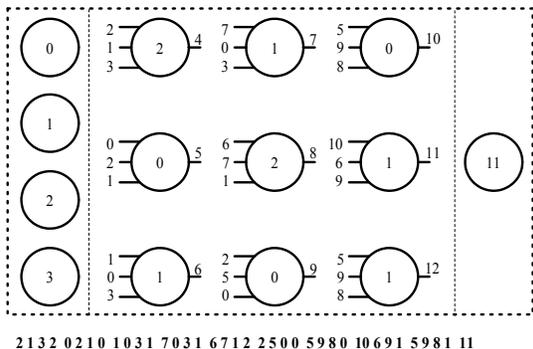
2 1 3 2 0 2 1 0 1 0 3 1 7 0 3 1 6 7 1 2 2 5 0 0 5 9 8 0 10 6 9 1 5 9 8 1 11

Figure 1: A CGP program with four input nodes and one output node. The program, distributed in a 3x3 grid, presents nodes with three input conections, a function, and its corresponding number label. Shown at the bottom, the string of integers representing the program.

a numeric label assigned, i.e. $I = \{0, 1, 2, 3\}$ and $F = \{0, 1, 2\}$. The program in this example is built with nine nodes. Each node is also labeled with a consecutive number $(4, 5, 6, \text{etc.})$ and has three inputs and one function (number inside the circle). The string of numbers to be evolved is formed with the inputs of each node and the function they execute. The program (graph) is built by simply connecting the numeric labels. For example, node 6 executes function 1 by taking inputs $1, 0, 3$ as parameters, whereas node 8 executes function 2 by taking nodes 6 and 7 and input 1 as parameters. Note that *not* all the nodes are connected and, therefore, will not be part of the final program (graph). These unconnected nodes are "spare material" called *introns*.

4   Task description

The goal of the task considered in this paper is to locate the *centroid* of an object clearly distinguishable for the human eye in a given image. The object of interest is assumed to be entirely shown in the image and over a solid background. The aim of the experiment is to evolve (learn) a program that performs

this task for different locations of the object in the image.

During the evolutionary (learning) phase, the location of the object in the image is given by the pair (x_q, y_q), whereas the response of the program is given by the pair (x_r, y_r). The best program is considered to be the one that keeps x_q and x_r, y_q and y_r, as close in value as possible.

On a particular test run, the input to the program consist of the set of pixel values $\{P_{ij}\}$ where P_{ij} is the intensity value of the pixel at row i, column j. The output from the program consists of the desired (x, y) centroid location of the object. Note that the program is not given any direct information about the location of the target.

This centroid location task is a relatively simple one if the aforementioned *classical* techniques were used. A program could be easily hand-coded for this task which would simple threshold the image to segment the object and then find its centroid. However, the problem is quite hard for an evolutionary algorithm to solve if specific image processing functions are not provided, or no image pre-processing is performed.

4.1   Experiment setup

The performance of evolved programs is greatly affected by the function set that is chosen to create them. If the system is given functions that are high-level and suitable for the task at hand, then it might be possible to come up with good programs by using a relatively small number of nodes. However, this approach and the hand-coded option are essentially the same. A more interesting option is to make the system as less dependant as possible from specifically designed functions and see the level of results that can be achieved.

Table 1 shows the selected functions for this experiment. These functions have the desired property of simplicity, which makes them easy and quick to compute. Note that **box** is the only function that retrieves information from the input data. It returns the average intensity of the pixels within a box of predefined size located at variable coordinates.

3

Table 1: Set of functions used for these experiments

Function	Description
sum, *sub*, *mul*, *div*	Arimethic functions
	Function *div* is protected
sqr, *cub*, *sqrt*, *sgn*, *neg*, *abs*	Mathematical functions
box	Function for accessing images
R $(-128 \leq R \leq 128)$	Random constant

These coordinates were altered during the recombination/mutation step with probability P_μ.

The fitness of the evolved programs is assessed at the beginning of every new generation. Each program is tested with a set of n different images and their individual location errors are measured. For this experiment the location error E for any given image is defined as follows:

$$E = \sqrt{(x_r - x_q)^2 + (y_r - x_q)^2} \qquad (1)$$

Where (x_r, y_r) is the program's response and (x_q, y_q) is the centroid location in the image currently presented to the program. The best program being the one that minimizes the fitness function F defined as the sum of the error E of the n different training images, namely:

$$F = \sum_{i=1}^{n} E_i \qquad (2)$$

All programs return a pair of values (x, y) that are interpreted as a $(column, row)$ location in the image. Program responses are bounded to values between $x_r^{\min}, x_r^{\max}$ and $y_r^{\min}, y_r^{\max}$. If a returned value falls outside these limits, it is mapped onto the valid range using the arithmetic mod operation.

The input images used in this experiment for training and testing are 256x256 still images with 256 level of grey. Four different objects taken against the same solid background were considered for this paper. Each picture shows a single object in various
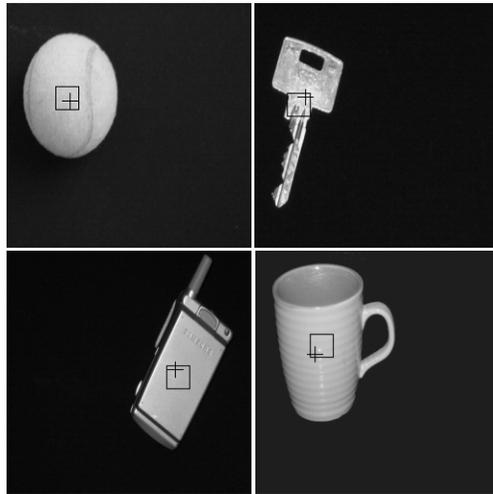

Figure 2: Set of images used in our experiments. The square shape is the target area and the hair cross shows the location returned by the evolved programas.

locations that occupies approximately 16% of the total surface. The distance from the object of interest to the camera was maintained constant for this experiment. A set of 8 images from each object were used for the training phase, and another set of 8 were used for the testing phase for a total of 64 different images. Figure 2 shows a sample of the images used and the response returned by the best program.

Although two different types of evolutionary algorithms have been tried in this experiment, the algorithm used in any case has essentially performed the following steps:

1. Generate initial population at random.

2. Evaluate fitness of individuals in population.

3. Promote fittest genotype to new population.

4. Fill remaining places in the new population with recombined/mutated versions of the fittest.

5. Return to step 2 until stopping criterion is reached

The way of breeding new individuals in step 4 made the difference in the both evolutionary algorithms. While crossover was the preferred operator for the runs in the first algorithm, mutation was the selected operator in the second attempt.

4.2 Experimental results

The image data used for this experiment was not pre-processed in any way. Normally a pre-processing routine is done either to reduce the dimensionality of the input data or to extract some desired features. In this work, we are more interested in investigating the performance of simple function primitives acting on *raw input* data.

As mentioned before, two different evolutionary algorithms were tested for this paper. In the first one, a variation of the standard crossover recombination operator was used: *uniform crossover*, in which each gene in the offspring is randomly picked up from the parents. Size five probabilistic tournament selection was used with the winner of the tournament being accepted as the first parent. The second parent was randomly selected among the other competitors in the tournament.

In the second algorithm, a simple form of *Evolutionary Strategy* (ES) [13] was used and no crossover was applied. This strategy is based on the mutation of $(1 + \lambda)$ individuals of the population with probability P_λ. Size two probabilistic tournament selection was used for this case with the winner of the tournament being accepted with a probability P_T, otherwise the looser was accepted. In any case, the algorithm used followed the steps listed in section 4.1.

A population of 100 individuals was used in both runs. All programs have two outputs, one for the component in x, the other for the component in y. All outputs are constrained to be in the range -128 to 128.

The algorithm employing uniform crossover was tested first. Initially, only 15 nodes were allowed per individual, but this program size did not produce any *useful* response, i.e. values within a predefined *neighborhood e* around the desired target. Even when the value of e was initially set big, the output of the programs was still considerably far from the target.

Clearly this was due to the small number of times a program "saw" any given input image. The alternative was then to increase the number of maximum nodes M allowed per program. Note that not necessarily all nodes form part of the final graph. Several values for M were considered. The last value tested was 80.

The results returned by the programs massively improved, but the neighborhood e value was still too big: $e = 45$ (the variable e is actually the magnitude of the error E that is considered to be admissible). Then the value of e was set to the more reasonable value of 15 (equivalent to 12 pixels) with the consequent fitness reduction. A number of crossover probabilities were tried ranging from 30 to 70 percent.

Given the disruptive nature of the uniform crossover, a new experiment was carried out using the aforementioned $(1+\lambda)$ ES. With the experience of the first experiment, the maximum number of nodes allowed per program was 80. Several values for λ were tested being $PopulationSize - 1$ the one used for the final runs. The results for the ES were better that the uniform crossover approach in terms of both fitness values and convergence time. For the first approach, 24% of all responses hit somewhere outside the object of interest, 76% inside, and 31% of all responses hit the area bounded by e. Whereas for the ES approach, 100% of the responses hit the object of interest and 60% of them landed within the bounds of e.

5 Observations

An essential aspect in this experiment is the flexibility that programs possess of "seeing" the input images. The only function available for the programs to examine the images is the function *box*. This function takes samples of the pixel intensities from different locations of the image. Other functions could have been included, but in order to adhere to our principal goal, we wanted to keep the function set as simple as possible.

Even when not always the response of the best program was exactly the desired centroid location, it is worth to note that all the responses for the ES ap-

proach hit the object of interest.

The maximum number of nodes allowed per program was 80. The important disadvantage with big programs, is the difficulty in understanding how they work. Understanding the programs might be useful if one wants to follow them step by step and make sure, for example, that the *interpreter* is working properly. However, at least in these experiments, small programs did not yield satisfactory results and big programs were too hard to decipher.

6 Conclusions

The goal of this research was to explore the performance of Cartesian Genetic Programming on image processing tasks. This paper has also briefly surveyed the published work on the use of evolutionary methods in the computer vision field. Two experiments were conducted using different evolutionary approaches, i.e. a Genetic Algorithm adopting uniform crossover and a simple $(1 + \lambda)$ Evolutionary Strategy. Our results showed that the Evolutionary Strategy performed better as to fitness response and convergence time.

The use of Cartesian Genetic Programming was successful in accomplishing the desired Object Localization task. Our implementation offers a good base for the experiments performed in this paper and should be a good reference for a broad range of future research work. Further development can be made to the program so that other challenging tasks could be tackled.

References

[1] J. Andrade-Cetto and A. Kak. Object recognition. *Wiley Encyclopedia of Electrical and Electronics Engineering*, vol. supplement 1:449–470, 2000.

[2] S. P. Brumby, J. Theiler, S. J. Perkins, N. Harvey, J. J. Szymanski, J. J. Bloch, and M. Mitchell. Investigation of image feature exstraction by a genetic algorithm. *Proceedings SPIE*, pages 24–31, 1999.

[3] J. Daida, T. F. Bersano-Begey, S. J. Ross, and J. F. Vesecky. Evolving feature extraction algorithms: Adapting genetic programming for image analysis in geoscience and remote sensing. *Proceedings of IGARSS'96*, 1996.

[4] J. Daida, J. D. Hommes, T. F. Bersano-Begey, S. J. Ross, and J. F. Vesecky. Algorithm discovery using the genetic programming paradigm: extracting low-contrast curvilinear features from the SAR images of the artic ice. *Advances in Genetic Programming*, pages 417–442, 1996.

[5] M. Ebner. On the evolution of interest operators using genetic programming. *Late Barking Papers at EuroGP'98: The First European Workshop on Genetic Programming*, pages 6–10, 1998.

[6] M. Ebner and A. Zell. Evolving a task specific image operator. *Proceedings of the First European Workshop on Evolutionary Image Analysis, Signal Processing and Telecomunications (EvoIASP'99 and EuroTel'99)*, pages 74–89, 1999.

[7] H. V. Hove and A. Verschoren. Genetic algorithms and recognition problems. *Genetic Algorithms for Pattern Recognition*, pages 145–166, 1996.

[8] M. P. Johnson, P. Maes, and T. Darrell. Evolving visual routines. *Artificial Life IV, Proceeding of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 198–209, 1994.

[9] M. C. Martin. *The Simulated Evolution of Robot Perception*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, June 2001.

[10] J. F. Miller and P. Thomson. Cartesian genetic programming. *In Proceeding of the Third European Conference on Genetic Programming*, 1802:121–132, 15-16 April 2000.

[11] R. Poli. Parallel distributed genetic programming. Technical Report CSRP-96-12, School of

Computer Science, University of Birmingham, U.K., 1996.

[12] R. Poli and S. Canogni. Genetic programming with user-driven selection: Experiments on the evolution of algorithms for image enhancement. *Genetic Programming 1997, Proceedings of the Second Annual Conference*, pages 269–277, 1997.

[13] H. P. Schwefel. *Evolution and Optimum Seeking.* Sixth-Generation Computer Technology. Wiley, New York, 1995.

[14] W. A. Tackett. Genetic programming for feature discovering and image discrimination. *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 303–309, 1993.

[15] A. Teller. *Algorithm Evolution with Internal Reinforcement for Signal Understanding.* PhD thesis, School of computer science, Carnegie Mellon University, Pittsburgh, PA, 1998.

[16] A. Teller and M. Veloso. Algorithm evolution for face recognition: What makes a picture difficult. *International Conference on Evolutionary Computation*, pages 608–613, 1995.

[17] A. Teller and M. Veloso. PADO: Learning tree structured algorithms for orchestration into an object recognition system. Technical Report CMU-CS-95-101, Deparment of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, 1995.

[18] I. Yoda, K. Yamamoto, and H. Yamada. An automatic acquisition of hierarchical mathematical morphology procedures by GA. *Proceedings of the 12th IAPR International Conference on Pattern Recognition*, vol 2:421–425, 1994.