A GP Artificial Ant for image processing : preliminary experiments with EASEA.

Enzo Bolis*, Christian Zerbi*, Pierre Collet†, Jean Louchet*, Evelyne Lutton‡

*ENSTA
32 bd Victor
F - 75739 Paris cedex 15

†Ecole Polytechnique
CMAPX/EEAAX
F - 91128 Palaiseau cedex

‡INRIA, projet FRACTALES
Rocquencourt, BP 105
F - 78153 Le Chesnay cedex

Abstract This paper describes how animat-based "food foraging" techniques may be applied to the design of low-level image processing algorithms. First, we show how we implemented the food foraging application using the EASEA software package. We then use this technique to evolve an animat and learn how to move inside images and detect high-gradient lines with a minimum exploration time. The resulting animats do not use standard "scanning + filtering" techniques but develop other image exploration strategies close to contour tracking. Experimental results on grey level images are presented.

1 Introduction

1.1 EASEA and genetic programming

EASEA [8] is a language dedicated to evolutionary algorithms. Its aim is to relieve the programmer of the painful chore of learning how to use evolutionary libraries and object-oriented programming by using the contents of a `.ez` source file written by the user. EASEA source files only need to contain the "interesting" parts of an evolutionary language, namely the fitness function, a specification of the crossover, the mutation, the initialisation of a genome plus a set of parameters describing the run. With this information, the EASEA compiler creates a complete C++ source file containing function calls to either the GAlib [10] or EO [9] library, depending on the one that was installed by the user. Therefore, the minimum requirement necessary to write evolutionary algorithms is the capability of creating non-object-oriented functions, specific to the problem which needs to be solved.

In our case, the genetic programming involved to program artificial ants was implemented using a tree structure on which genetic operators were defined. The EASEA compiler converted the specification into a compilable C++ source file using GAlib.

1.2 The artificial ant problem, animats and low-level image processing

The Artificial Ant problem [18][15] considers the task of navigating an animat [25][28] gathering its food scattered irregularly (e.g. using the Santa Fe trail) over a two-dimensional domain represented as a binary image. This problem, known as containing multiple levels of deception (GA-hard) [23], has often been used as a benchmark for Genetic Programming (GP). In the first part of this paper, we show an implementation of this problem using EASEA and following John Koza's implementation [18]. The animat's task is to detect all the food available in the image with the shortest possible path length. The output is a binary image showing the pixels detected by the animat.

In the second part of the paper, we propose an extension of this Animat methodology and genetic programming to automatically write low-level image processing tasks, without making any assumption about the exploration strategy used. The vast majority of low-level image processing algorithms uses standard, systematic image scanning and pipe-line data processing [14], which is not necessarily the most cost-effective solution (see Section 3). To this end, we use an animat fitted with simple sensors and actuators, able to move (from a position to one of its neighbours) inside the image and to see the values of its surrounding pixels, and optimise its internal program using genetic programming and a fitness function depending on the task to be performed (here, contour detection). The input is a natural grey level image: the animat's task is to find as many contour pixels as possible, using a minimal path length. Again, the output is a binary image showing the pixels the animat has detected.

The behavioural aspects of several different animat types have been studied [25] and linked to real-world behaviours in Biology [19][27][2], but as far as we know, only few applications to real image processing have been published. See [16,17] for crack detection with "image exploring agents".

2 Implementing a food foraging process using EASEA

2.1 Animat functions

The topology used throughout this paper is 4-connexity. In other terms, the animat's axis may only have 4 possible orientations, and its elementary movements are rotations or one-step translation ahead. With regard to the animat functions, we followed Koza's definitions [18], but added specific sensors to our artificial ant, resulting in the following two classes of functions:

- animat's sensor functions:
  - `if_food_ahead()`: if there is food present in the pixel currently just ahead of the animat, the first argument is executed; else the second one is executed.
  - `if_food_on_lf()`, same if there is food on the left side;
  - `if_food_on_rg()`, same if there is food on the right side;
  - `already_visited()`, if the animat is on a pixel already visited, the first argument is executed; otherwise the second one is executed. An image-sized buffer is used to memorise pixels already visited.
- animat's actuator functions:
  - `move()`, moves the animat ahead;
  - `left()`, rotates the animat at a right angle counterclockwise;
  - `right()`, rotates the animat at a right angle clockwise;
  - `progn2(action1, action2)`, the animat executes the actions 1 and 2 sequentially;
  - `progn3(action1, action2, action3)`, the animat executes the actions 1, 2, 3 sequentially.

2.2 Initialising trees

An ant's program is represented “à la Koza”, with a genetic programming tree containing the previously described actions in its nodes. The population of trees is initialised randomly, with a maximum depth of 5. The maximum depth has been predetermined although we remarked that the trees are most often well balanced, thanks to the functions `progn2` and `progn3` which grow the tree both in depth and width.

2.3 Crossover

The two-parent crossover uses two independently chosen crossover sites for each parent. In order to limit crossover between leaves, if one of the sites is on a leaf, then this crossover will be done with a 30% probability. The total crossover probability is determined by the EASEA parameter `PCross`.

2.4 Mutation

Two types of mutation are introduced: leaf mutation and subtree mutation. leaf mutations (65% of total mutations) consist in randomly choosing a leaf, then replace it with a randomly chosen function. Subtree mutations (35% of total mutations) consist in randomly choosing two sites in the tree, then swapping the two corresponding subtrees. The total mutation probability is determined by the EASEA parameter `PMut`. These probability values have been determined experimentally.

2.5 Fitness evaluation

As the primary goal is to find food, the quantity of food detected will be the main factor in the fitness function. However, it is also important to take into account the path complexity of the animat (measured by the number of elementary action steps) and the complexity of the genome. Appropriate weights *A* and *B* allow to control the relative impact of these factors.

$$fitness = (quantity\ of\ food\ found) - A \times (\#\ steps) - B \times (genome\ size)$$

The orders of magnitude are: $A = 10^{-2}$, $B = 10^{-4}$ (experimental values).

Additional constraints are taken into account by giving arbitrarily low values to the fitness if not satisfied:

- The `autonomy` parameter, which gives the maximum number of steps allowed without food.
- In order to prevent the animats from spinning around themselves, a credit of 4 rotations is given initially (in order to allow a complete 360° rotation and have a “panoramic sight”) and the subsequent extra rotations are penalised. The rotation counter is reinitialised to -4 each time a `move` is executed.

The number of steps taken into account to calculate the fitness is fixed to a value *nsteps* which decreases during the first part of evolution (to eliminate those individuals trying to scan all the image without any clever strategy) from the parameter value `Operations_max` to `Operations_min`, then is raised again (to allow a more efficient selection of individuals able to find most of the existing food) back to `Operations_max`.

2.6 Results

The algorithm has been tested on three binary images with increasing difficulty (Fig.

1). With the first one, food has been placed continuously along a *Santa Fe* trail [18, 20]. The path in the second image was built by removing the food in the corners of the Santa Fe trail, thus resulting in a trail made of disconnected straight lines. In the third image, food is distributed with important gaps, or even on isolated pixels.

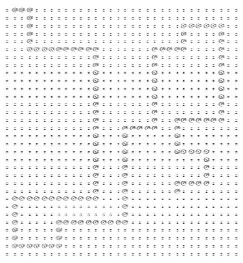


Image 1 : continuous Santa Fe trail.

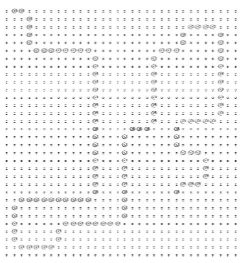


Image 2 : Santa Fe trail with removed corners.

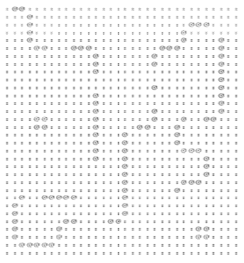


Image 3 : Santa Fe trail with gaps and isolated food patches.

Figure 1: binary test images used

The `autonomy` parameter has been fixed at 50 for the first data set, 100 for the 2nd, 150 for the 3rd one. Table 2 shows the best results we obtained on these data sets.

Terminal set	`left, right, move`
Function set	`if_food_ahead, if_food_on_lf, if_food_on_rg, already_visited, progn2, progn3`
Selection	`ranking`
Wrapper	`program repeatedly executed for a fixed number of motion steps`
Parameters	`Pmut=0.4 (total), Pcross=0.9 (total)`

Table 1: main features of the GP

Image #	run #	food present	food found	motion steps	autonomy	nodes in animat	population	# generations
1	1	144	144	144	50	3	500	5
2	2	123	123	145	100	25	1000	5
2	3	123	123	145	100	10	2000	5
2	4	123	123	145	100	7	1000	25
3	5	105	91	701	150	31	8000	20
3	6	105	91	701	150	20	15000	10
3	7	105	105	423	150	21	10000	30

Table 2: results obtained on binary test images

Runs 1-4 (on images 1 and 2) show that small populations and few generations are sufficient to get a satisfactory animat, however runs 3 and 4 illustrate the fact that larger populations or more generations help optimise the tree structure. Runs 5-7 (on the more difficult image 3) show that on more difficult data, both large populations and large number of generations are necessary. The best code resulting from run 7 is given in Appendix 1.

3 Contour detection

3.1 Grey level images and contour detection

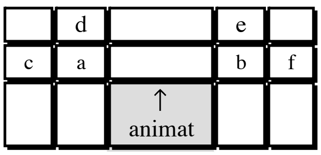
Contour detection in grey level images is a popular problem in image processing [1, 14]. While this problem may not always be as deceptive as the food foraging problem, its apparent similarity with the latter and the importance of its real-world applications led us to study the possibility of transposing the method presented in Section 2.

A contour in a grey level image is usually defined as a continuous line, whose points are local maxima of the image gradient norm. Most contour detection methods are based on standard image scanning. For each pixel a decision is taken, based e.g. on the first-order (and sometimes second order) grey level derivatives in the pixel's neighbourhood, or other characteristics like Hölder exponents [24]. Some approaches use contour tracking with local optimisation [14], or global optimisation methods [13], but most scan the image the usual way[1]. Among the few exceptions we can mention works on dynamic contour tracking and parallel algorithms developed for specific architectures. Our animat-based approach to contour detection does not make any assumption about the image exploration strategy that will be used.

3.2 Basic animat functions

The animat language primitives are similar to the ones used in the previous section. We have changed the perception functions, according to the new purposes of the application. We have replaced the `if_food_*` functions with the following ones:

- `if_contour_ah()`: if $|a - b|$ is greater than a fixed threshold, the first argument is executed; else the second one is executed;
- `if_contour_aw()`, same with $|c - f|$;
- `if_contour_af()`, same with $|d - e|$.



3.3 Genetic operators

The mutation and crossover operators are the same as those used in the previous section. As in section 2, similar additional constraints have been added to the fitness function (rotation credit, number of steps).

3.4 Results

Here the fitness function will favour animats whose trajectory contains highly contrasted pixels. Thus the first term of the fitness function is now the number of pixels detected whose contrast is greater than a given threshold. With this fitness function applied to a classic Lena image we obtained the following results:

[1] The new generation of CMOS image sensors allows random pixel access, which is giving a new opportunity to vision systems not reading the pixels the linear way. The constraint of parallel image scanning is thus becoming more of a cultural than a technical constraint, and may encourage more costly calculations. A clever image exploration strategy may help get more efficient image processing algorithms, especially in embedded real-time applications.

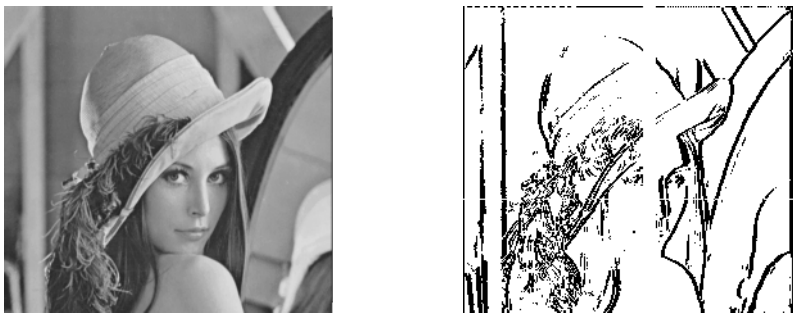


Figure 2: “Lena” image and result found using the best animat obtained after 30 generations

It is worth noting that with a toroidal representation of the image, the genetic programming created an animat scanning the full image[2]. Some pixels are missed but it is interesting to remark that the system discovered by itself that image scanning is an efficient method not to miss contours.

Then, in order to avoid time-consuming full image scanning and get more efficient contour detectors, we replaced the first term of the fitness function with an expression including the number of detected contour pixels, the number of scanned pixels and the number of consecutive detections (in order to encourage continuous detection of contours). To test the improved fitness function, we have trained the genetic algorithm on the test images 1 and 2 shown on Fig. 3, obtaining quite efficient contour following algorithms:

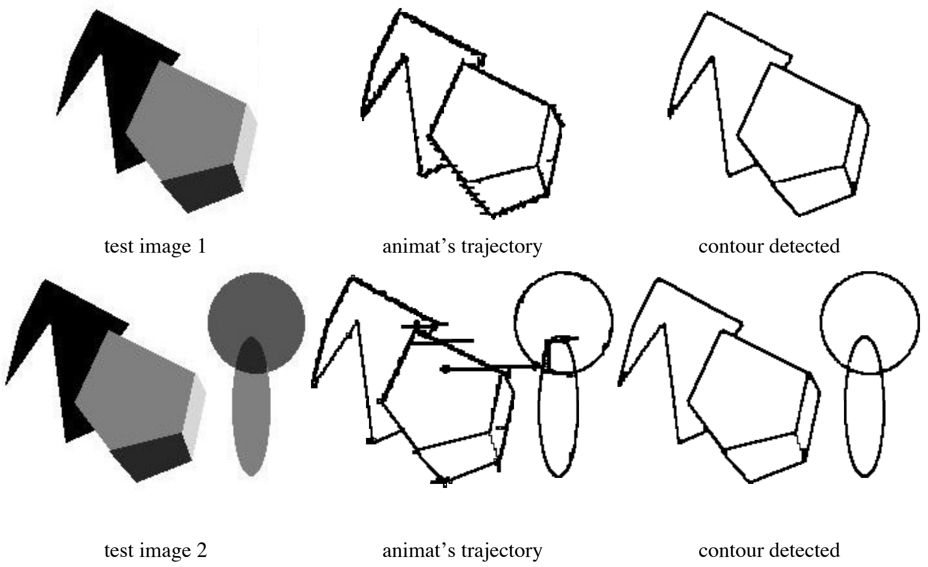


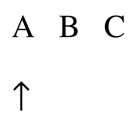
Figure 3: synthetic test images and results

[2] This is why we did not display the trajectory of the animat.

The change in exploration strategy comes essentially from the changes in the fitness function. We suppressed the constraint on path length which now becomes useless in the learning phase with the new fitness ($A = 0$). The resulting animat's program (given in Appendix 4) is functionally equivalent to the following pseudo-C code:

```
if (contour ahead)
else turn right;
move;                                  first part
move;
turn left;
if (contour ahead) {
  move;
  move;
}
else {                                 second part
  if (already visited) turn right;
  else {
    turn left;
    move;
    turn right;
  }
}
```

Code generated



Relative positions of ant and pixels A, B, C

Some interesting remarks can be made on the simplified procedure. First of all, in this example, the functions `if_contour_aw` and `if_contour_af` are not used, showing that the analysis of the image can be done with the sole `if_contour_ahead` instruction.

If we now put the animat into an unexplored part of the picture without any contour pixel, and if we let pixels A, B and C be the three pixels in a straight line in front of the animat, executing the first part moves the animat two pixels to the right, to C. Only pixels A and C are tested as contour pixels. In the second part, the animat backtracks to pixel B using a `left-move-right` sequence, then gets the proper orientation for the following execution. All pixels on a line are tested unless an already visited position is reached during backtracking. Then, the animat turns right and starts analysing a perpendicular line. Other configurations may be analysed the same way.

4 Comments on implementation.

The project was first written using EASEA, which provided an easy implementation and efficient interfacing with GAlib. We then edited manually the resulting C++ code. Thanks to this method, we avoided all the effort of defining data and program structures, reading the GAlib documentation or writing calls to the GAlib library, and could concentrate into more specific issues like fine tuning the fitness function and genetic parameters. On the whole, EASEA allowed us a much faster and more efficient implementation.

5 Conclusion and future work.

The aim of this paper is not to propose yet another "better" contour detection algorithm. We showed an example of how genetic programing is able to build from scratch, without using prior knowledge in image processing, a contour detection algorithm that gives results not competing but fairly comparable with those given by usual contour detectors [14]; perhaps more importantly, the experiment presented shows that EASEA can successfully evolve an artifical ant in a complex environment. Genetic programming and EASEA can be an interesting alternative way to quickly and easily develop algorithms in image processing and in other application domains where previous resolution methods are not available.

Future extensions include:

- using several images for learning [20], in order to get a more robust algorithm,
- fitting the animat with an individual memory, e.g. of the average direction followed in the near past [25],
- giving the animat the ability of having a non-deterministic behaviour,
- using supervised learning,
- using an “ant colony” rather than a single animat, may allow cooperative behaviour (via direct cooperation or via pheromones).

6 References

[1] D. H. Ballard and C. M. Brown, *Computer Vision*, Prentice Hall, 1982.

[2] R. J. V. Bertin and W. A. van de Grind, *The Influence of Light-Dark Adaptation and Lateral Inhibition on Phototaxic Foraging: A Hypothetical Animal Study*, Pages 141-167, Adaptive Behavior, Volume 5, Number 2, Fall 1996.

[3] V. Cantoni, M. Ferretti, S. Levialdi, R. Negrini and S. Stefanelli, *Progress in Image Analysis and Processing*, World Scientific,1991.

[4] D. Cliff and S. Bullock, *Adding “Foveal Vision” to Wilson's Animat*, Pages 49-72, Adaptive Behavior, Volume 2, Number 1, Summer 1993.

[5] P. Collet, E. Lutton, M. Schoenauer, J. Louchet, *Take it EASEA,* Parallel Problem Solving from Nature VI, vol 1917, Springer pp 891-901, Paris, September 2000.

[6] R. J. Collins and D. R. Jefferson. *Antfarm: Towards simulated evolution* . In S. Rasmussen, J. Farmer, C. Langton and C. Taylor, editors, Artificial Life II, Reading, Massachusetts, Addison-Wesley, 1991.

[7] F. L. Crabbe, Michael G. Dyer, *Observation and Imitation: Goal Sequence Learning in Neurally Controlled Construction Animats: VI-MAXSON*, SAB 2000, Paris.

[8] EASEA (EAsy Specification of Evolutionary Algorithms) home page: `http://www-rocq.inria.fr/EASEA/`

[9] EO (Evolutionary Objects) home page: `http://geneura.ugr.es/~jmerelo/EO.html`

[10] GAlib home page: `http://lancet.mit.edu/ga/`

[11] P. Gaussier. *Autonomous Robots interacting with an unknown world,* Special Issue on Animat Approach to Control, Robotics and Autonomous Systems, 16, 1995.

[12] R. C. Gonzalez, R. E. Woods, *Digital Image Processing*, Wiley, 1992

[13] J. Ivins, J. Porrill, *Statistical Snakes: Active Region Models,* British Machine Vision Conference, York, Sep. 1994.

[14] R.C. Jain, R. Kasturi, B.G. Schunck, *Machine Vision*, McGraw-Hill, 1995.

[15] D. Jefferson, R. Collins, C. Cooper, M. Dyer, M. Flower, R. Korf, C. Taylor, A. Wang, Evolution as a theme in artificial life: the Genesys/Tracker system, Artificial life II, vol. X, Santa Fe Institute Studies in the Sciences of Complexity, Addison-Wesley, Feb. 1992, 549-578.

[16] M. Köppen, B. Nickolay, *Design of image exploring agent using genetic programming*. In Proceedings of the 4th International Conference on Soft Computing, volume 2, pages 549--552, Fukuoka, Japan, 30. Sep - 5. Oct 1996. World Scientific, Singapore.

[17] M. Köppen, B. Nickolay, *Design of Image Exploring Agent using Genetic Programming*. Fuzzy Sets and Systems, Special Issue on Softcomputing, 103 (1999) 303-315.

[18] J. R. Koza, *Genetic Programming*, MIT Press 1992.

[19] J. R. Koza, J. Roughgarden and J. P. Rice, *Evolution of Food-Foraging Strategies for the Caribbean Anolis Lizard Using Genetic Programming*, Pages 171-199, Adaptive Behavior, Volume 1, Number 2, Fall 1992.

[20] I. Kuscu, *A genetic Constructive Induction Model*. In P. J. Angeline, Z. Michalewicz, M. Schoenauer, XinYao, and A. Zalzala, editors, Proceedings of the Congress on Evolutionary Computation , volume 1, pages 212-217, Mayflower Hotel, Washington D.C., USA, 6-9 July 1999. IEEE Press.

[21] W. B. Langdon, *Genetic Programming and Data Structures : Genetic Programming + Data Structures = Automatic Programming !*, Kluwer, 1998.

[22] W. B. Langdon and R. Poli, *Better Trained Ants for Genetic Programming,* Technical Report CSRP-98-12, April 1998, `http://www.cs.bham.ac.uk/wbl`.

[23] W. B. Langdon and R. Poli, *Why Ants are Hard,* Technical Report: CSRP-98-4, January 1998, `http://www.cs.bham.ac.uk/wbl`.

[24] J. Lévy Vehel, *introduction to the multifractal analysis of images*, in Fractal image encoding and analysis, Yuval Fischer ed., Springer Verlag, 1996.

[25] J. A. Meyer, A. Guillot, *From SAB90 to SAB94: Four Years of Animat Research,* Proceedings of Third International Conference on Simulation of Adaptive Behavior. Brighton, England, 1994.

[26] R. Moller, D. Lambrinos, R. Pfeifer, T. Labhart, and R. Wehner, *Modeling Ant Navigation with an Autonomous Agent,* From Animals to Animats 5, Proc. of the 5th Int. Conf. on Simulation of Adaptive Behavior, August 17-21, 1998, Zurich, Switzerland, edited by R. Pfeifer, B. Blumberg, J.-A. Meyer and S. W. Wilson

[27] T. J. Prescott, *Spatial Representation for Navigation in Animats*, Adaptive Behavior, Volume 4, Number 2, Fall 1995, 85-123

[28] S. W. Wilson, *Classifier systems and the animat problem,* Machine Learning 2 (1987), 199-228.

[29] S. W. Wilson, (1991). *The animat approach to AI* . In J. Meyer & S. W. Wilson (Eds), From Animals to Animats, Proceedings of the first International Conference on Simulation of Adaptive Behavior, Cambridge, MA: MIT Press, 15-21.

[30] M. Witkowski, *The Role of Behavioral Extinction in Animat Action Selection*, SAB 2000, Paris, 2000.

Appendix 7: best code generated in food pick-up

```
|-+-PRG3  3 successive actions
  \_LEFT    turn left
 |-+-IFOL   if food ahead on left
  |-+-PRG3    3 successive actions
   |-+-IAHR     if already visited      Steps: 423
     \_MOVE       move ahead            Food: 105 of 105
     \_RGHT       turn right            Current parameters:
   |-+-PRG3     3 successive actions    Max Operations: 850
     \_LEFT       turn left             Min Operations: 650
     \_MOVE       move ahead            Population size: 10000
     \_RGHT       turn right            Generations: 30
   \_MOVE       move ahead              PMutation: 0.4
   |-+-IFOR   if food ahead on right    PCrossover: 0.9
    |-+-PRG2    2 successive actions    Replacement: 0.4
     \_MOVE       move ahead            Elapsed time: 199.33 seconds
     \_LEFT       turn left              for 58184 evaluations.
   |-+-PRG3     3 successive actions
     \_MOVE       move ahead
     \_LEFT       turn left
     \_LEFT       turn left
  \_LEFT    turn left
```

Appendix 8: best code generated in contour detection (Lena image)

```
|-+-PRG3    3 successive actions
  |-+-ICAH    if contrast (a-b)
     \__RGHT    turn right
    |-+-PRG2    2 succ. actions
      |-+-PRG3    3 succ. actions
         \__RGHT    turn right
         \__LEFT    turn left           Steps:   202970
         \__RGHT    turn right          Contour pixels:         2702
       \__LEFT    turn left             Pixel visited:          4599
  |-+-PRG2    2 successive actions      Current parameters:
    |-+-PRG2    2 succ. actions         Max Operations : 203000
       \__MOVE    move ahead            Min Operations : 180001
       \__MOVE    move ahead            Population size: 24952
     \__LEFT    turn left               Generations: 33
  |-+-ICAH    if contrast (a-b)         PMutation: 0.4
    |-+-AHRE    if already here         PCrossover: 0.9
      |-+-PRG3    3 suc. actions        Replacement: 0.4
         \__LEFT    turn left           Autonomy: 500000
         \__MOVE    move ahead          Elapsed  time:  770.472  seconds  for
         \__RGHT    turn right          157291 evaluations.
       \__RGHT    turn right
    |-+-PRG2    2 succ. actions
      |-+-ICAH    if contrast (a-b)
         \__LEFT    turn left
         \__MOVE    move ahead
       \__MOVE    move ahead
```

Appendix 9: best code generated in contour detection (test 2)

```
|-+-PRG3     3 successive actions
  \__MOVE      move ahead
  |-+-ICAW     if contrast (c-f)
    |-+-AHRE     if already here
      |-+-ICAF     if contrast (d-e)    Steps:   41353
         \__RGHT     turn right         Contour pixels:          2378
         \__MOVE     move ahead         Pixel tested:            4178
      |-+-ICAF     if contrast (d-e)    Current parameters:
         \__LEFT     turn left          Max Operations : 90000
         \__LEFT     turn left          Min Operations : 60000
    |-+-ICAW     if contrast (c-f)      Population size: 20000
       \__MOVE     move ahead           Generations: 35
       \__RGHT     turn right           PMutation: 0.4
  |-+-ICAH     if contrast (a-b)        PCrossover: 0.9
    |-+-AHRE     if already here        Replacement: 0.4
      |-+-ICAF     if contrast (d-e)    Elapsed time: 959.262 seconds for
         \__LEFT     turn left          132147 evaluations.
         \__LEFT     turn left
       \__LEFT     turn left
    |-+-ICAF     if contrast (d-e)
       \__MOVE     move ahead
       \__MOVE     move ahead
```