GENETIC PROGRAMMING
THEORY AND PRACTICE X

GENETIC PROGRAMMING THEORY AND PRACTICE X

Edited by
RICK RIOLO
Center for the Study of Complex Systems
University of Michigan

EKATERINA VLADISLAVLEVA
Evolved Analytics Europe BVBA, Belgium

JASON H. MOORE
Institute for Quantitative Biomedical Sciences, Dartmouth Medical School

MARYLYN D. RITCHIE
...

Kluwer Academic Publishers
Boston/Dordrecht/London

Inhaltsverzeichnis

Chapter 1

CARTESIAN GENETIC PROGRAMMING FOR IMAGE PROCESSING

Simon Harding and Jurgen Leitner and Jurgen Schmidhuber
Dalle Molle Institute for Artificial Intelligence (IDSIA), Manno, Switzerland

Abstract

Combining domain knowledge about both imaging processing and machine learning techniques can expand the abilities of Genetic Programming when used for image processing. We successfully demonstrate our new approach on several different problem domains. We show that the approach is fast, scalable and robust. In addition, by virtue of using off-the-shelf image processing libraries we can generate human readable programs that incorporate sophisticated domain knowledge.

Keywords: Cartesian Genetic Programming, Image Processing, Object Detection

1. Introduction

Genetic programming (GP) has often been used to solve problems in image processing; however previous attempts typically use a small set of mathematical functions to evolve kernels or a small number of basic image processing functions (such as erode and dilate). Given the maturity of the field of image processing, it should be possible to construct programs that use much more complicated image operations and hence incorporate domain knowledge.

In this work, we present a technique based on Cartesian Genetic Programming (CGP), that allows for the automatic generation of computer programs using a large subset of the OpenCV image processing library functionality (Bradski, 2000). Our design choices for the approach are intended to match human design choices - hence the use of an industry standard API, generation of code in a standard programming language, and the inclusion of additional domain knowledge.

We demonstrate the efficacy of this approach in several different domains: basic image processing, medical imaging, and object detection in robotics.

Previous work

There exists a vast body of work on all aspects of image processing, using both classical and machine learning approaches. We therefore focus this review on Genetic Programming based approaches. Many GP approaches work as convolutional filters. In this type of approach, a sliding window moves across an image. At each pixel location, an evolved expression takes the in neighboring pixels' values, and computes a new value for the centre pixel (Gonzalez and Woods, 2006). Typically these programs operate at a mathematical level, where operations such as $+, -, \times$ and $\div$ are used (Harding, 2008). Implementations on Field Programmable Gate Arrays (FPGAs) for noise reduction use a mixture of binary operations (e.g. OR, XOR, AND, etc) and mathematical functions (Slany and Sekanina, 2007). Similarly, Wang also uses a mixture of function types, in this instance binary and some basic morphological operations, such as dilate (Wang2 and Tan, 2011). Our first demonstration in Section 5 revisits the noise reduction problems.

Segmentation is the process of separating foreground from background or one object class from the background. Using GP, Spina evolved programs to perform segmentation based on features calculated from partially labelled foreground and background (Spina et al., 2009). In an approach similar to CGP, Shirakawa evolved segmentation programs that use many high-level operations such as mean, maximum, minimum, Sobel, Laplacian, sum and product (Shirakawa and Nagao, 2007). This approach is probably the most similar to CGP-IP. In later work, the technique was successfully applied to texture classification (Shirakawa et al., 2009). In the domain of medical imaging, Poli used GP with basic mathematical operators, working at a per pixel level to segment features from MRI scans (Poli, 1996). In this paper we also demonstrate CGP-IP on a medical imaging problem.

Classification of images using GP has been applied to many different domains. For example, adding loops to GP enabled Wijesinghe to classify simple shapes using just primitive mathematical operators (Wijesinghe and Ciesielski, 2007). Using a simple function set, but operating on large number of precomputed statistics, Zhang was able to perform domain-independent and multiple class object recognition (Zhang et al., 2003). This paper also has a good overview of previous work in object detection in GP. Terrain classification using GP has been applied to satellite imagery (Silva et al., 2010), including work on hyperspectral images (Uto et al., 2009).

CGP itself has previously been used for image processing tasks. Several examples can be found in (Sekanina et al., 2011; Harding, 2008; Smith et al., 2005).

CGP-IP draws inspiration from much of the previous work in the field. As will be demonstrated later in this chapter, it uses a mixture of primitive mathe-

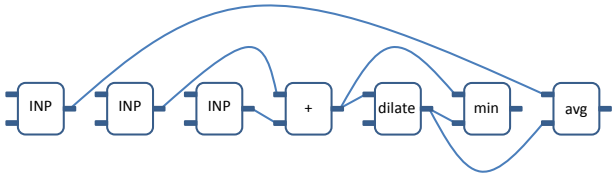


Figure 1-1. Example illustration of a CGP-IP genotype. Internally each node is represented by several parameters, as described in Section 2. In this example, the first three nodes obtain the image components from the current test case (i.e. a grey scale representation and the red and green channels). The fourth node adds the green and red images together. This is then dilated by the fifth node. The sixth node is not referenced by any node connected to the output (i.e. it is neutral), and is therefore ignored. The final node takes the average of the fifth node and the grey scale component from the current test case.

matical and high level operations. It uses CGP, which appears to be a popular choice for the representation in this domain. It encompasses domain knowledge, and could even be easily adapted to include the findings from previous work in the function set.

2. Cartesian Genetic Programming

Cartesian Genetic Programming (CGP) is a form of Genetic Programming in which programs are encoded in partially connected feed forward graphs (Miller, 1999; Miller, 2011). The genotype, given by a list of nodes, encodes the graph. For each node in the genome there is a vertex, represented by a function, and a description of the nodes from where the incoming edges are attached. This representation has a number of interesting properties.

For instance, not all of the nodes of a solution representation (the genotype) need to be connected to the output node of the program. As a result there are nodes in the representation that have no effect on the output, a feature known in GP as 'neutrality'. This has been shown to be very useful (Miller and Smith, 2006) in the evolutionary process. Also, because the genotype encodes a graph, there can be reuse of nodes, which makes the representation distinct from a classically tree-based GP representation. An example is shown in Figure 1-1.

Compared to classical CGP, CGP-IP needs a number of additional values encoded in each node. This is because the available functions often require one or more parameters, and these parameters have requirements as to their type and range. Hence, each node in a CGP-IP graph contains the following elements:

- Function: Integer representing a function from the set of available functions.

- Connection 0, Connection 1: Integer representing how many nodes back in the current graph this node should connect to obtain the inputs to the function.
- Parameter0: Real number, typically used as a constant value.
- Parameter1, Parameter 2: Integers in the range -16 to +16, used as a parameter to an image operation.
- Gabor Filter Frequency: Integer in the range 0 to 16, used as a parameter for Gabor filter operations.
- Gabor Filter Orientation: Integer in the range -8 to 8, used as a parameter for Gabor filter operations.

If a relative address extends beyond the extent of the genome it is connected to one of the inputs. Specialised 'Input' functions are also provided (INP, INPP, SKIP), which manipulate a pointer that indexes the available inputs and return the currently indexed input. A full description can be found in (Harding et al., 2010b) and (Harding et al., 2010a). The output is taken from the last node in the genome. An illustrative example is shown in Figure 1-1.

All genes in the genotype have an equal probability of being mutated. The type of mutation that occurs depends on the type of gene being mutated. For the function genes, a new function is selected at random from the available functions. The connection genes are mutated to a new value from 1 to the length of the genotype. For the Gabor Frequency, a random integer between 0 and 16 is used. For the Orientation, mutation selects an integer between -8 and 8. The mutation for the Parameter0 gene is slightly more complicated. If that gene is selected for mutation, with equal probability it will be set to either a random value between -255 and +255, or noise (maximum of +/-10%) will be added.

In addition to the program graph, the genotype also includes a real number value that is used for thresholding during binary classification problems. The threshold has a range of 0 to 255. During mutation, this parameter is modified with a 1% probability, and uniform noise of +/-10% is added.

The genotype for CGP-IP contains a large number of elements that need to be evolved. Such a complex representation would normally be considered too large to efficiently evolve, however the results shown in Section 5 indicated otherwise. Although further investigation is required, we suggest that this complexity increases neutrality (which is understood to be useful to CGP) and also increases the flexibility of the function set (leading to an increase in the number of possible solutions). A large number of more specialized parameters in each node avoids the necessity to perform casting and interpretation of an arbitrary set of values in each node, which aids readability. Further, the specialization of the parameters enables mutation to act appropriately.

Executing the genotype is a straightforward process. First, the active nodes are identified to perform the genotype-phenotype mapping. This is done recursively, starting at the output node and following the connections used to provide the inputs for that node. In CGP-IP the final node in the genotype is used as the output.

Next, the phenotype can be executed on an image. The input image (or images) are used as inputs to the program, and then a forward parse of the phenotype is performed. If the problem is a binary classification, the output image is then thresholded. For efficiency, if there are multiple images to be processed, only this second step is needed. CGP is also efficient in the sense that only active nodes need to be computed and that the results from these nodes can be stored and reused.

3. Function Set

One of the main benefits of Genetic Programming is the ability to easily include domain specific knowledge. This can be performed in two ways. The first method, which is also applicable to other machine learning techniques, is to provide useful pre-processed inputs. For example, here CGP-IP, color input images are split into Hue, Saturation, and Value and Red, Green, and Blue components – and all these components are available as program inputs.

The other approach is to provide GP with domain specific knowledge through an appropriate function set. In CGP-IP the function set not only contains primitive mathematical operations, but also high-level image functions, that benefit from previous work in image processing. In this implementation, a large number of OpenCV functions are available.

With over 50 unique functions, the function set is considerably larger than those typically used with Genetic Programming. This does not appear to hinder evolution, and again we speculate that the increased number of functions provides greater flexibility in how the evolved programs can operate. Functions in CGP-IP use the additional values stored in the genes for each node as parameters. A summary of the function set follows.

As discussed previously, inputs are accessed using the special functions *INP*, *INPP* and *SKIP* which move a pointer 1, -1, or by *Parameter0* through the available inputs. *const* generates an image where every pixel has the value *Parameter0*. *NOP* acts as a pass-through, and returns the output of the node it is connected to. *add*, *sub*, *mul* each take two images as inputs, and pixel-wise perform the relevant mathematical operation. *log*, *exp*, *sqrt* work similarly, but take just one input image. *addc*, *sub*, *mulc* take one image as an input, and then adds (etc.) the value of *Parameter0* to each pixel value. *dilate*, *erode*, *Laplace*, *Canny* perform the well known image processing operations. *gauss* performs a Gaussian blur, *gauss2* also performs a blur but over the window size

given by *Parameter1* and *Parameter2*. *min*, *max*, *avg*, *absDifference* compare each pixel in two input images and output the minimum (etc.) value of the corresponding pixels. Similarly, *minc*, and *maxc* compare the pixels' values in an image to *Parameter0* and return the minimum or maximum value. The *normalize* function scales the pixel values to between 0 and 255. The *sobel*, *sobelx* and *sobely* functions perform edge detection with various parameters and directions. *threshold* performs a binary threshold of an image, outputing a white pixel if a value is more than *Parameter0*, black otherwise. *smoothMedian*, *smoothBilateral*, *smoothBlur* and *unsharpen* all perform the relevant smoothing operation. Parameters are taken from the node's parameters. Images can be shifted (or more accurately, a circular shift) by *shift*, *shiftUp*, *shiftDown*, *shiftLeft* and *shiftRight*. *shift* takes two parameters to say how far to shift in the horizontal and vertical directions. Shifting is an important feature, as it allows for processing of the neighbourhood (see (Harding, 2008) for details). *reScale*, downsamples the image by *Parameter0* factor and then upscales it again to the original size. *gabor* performs the Gabor filter which is useful in finding textures. *resizeThenGabor* downsamples an image and then performs the Gabor filter. Parameters for the Gabor function are taken from the calling node. *minValue*, *maxValue*, *avgValue* find the minimum, maximum or average pixel value in the input image and output an image where every pixel has that value. *localMin*, *localMax*, *localAvg*, *localNormalize* take a neighbourhood of pixels (*Parameter1* by *Parameter2* pixels in size) and compute the appropriate statistic for that region. These are related to the image processing operations of open and close.

4. Parameters

As with other CGP implementations, CGP-IP does not require many parameters to be set. The main parameters are:

- Graph length (i.e. the number of nodes in the genotype), which is set to 50 in this case.
- Mutation rate, 10% of all genes in the graph are mutated when an offspring is generated. The threshold parameter is mutated with a probability of 1%.
- Size of mutations
- Number of islands, this depends on the available computing resources. CGP-IP has been tested successfully from 1 to 24 islands. For all the experiments shown here, 24 islands are used.
- Number of individuals per island, which is set to 5 in keeping with the typical 1+4 evolutionary strategy used with CGP.
- Synchronization interval between islands. Here each island compares their best individual to the server's individual every 10 generations.

It is important to note that in the work presented here the parameters have not been optimized other than by casual experimentation. It may be possible to improve the performance of CGP-IP by more carefully selecting these parameters. In particular, we would expect the mutation rate, genotype size and number of islands to be the most important parameters to adjust. All parameters are kept constant throughout the experiments presented below. Again, whilst it may be possible to improve performance for a given problem by optimizing the parameters, we believe that the apparent parameter robustness is an important feature of this technique.

5. Experiments

In this section of the chapter, we demonstrate CGP-IP on several different domains. All experiments use the same configuration, with the exception of choosing the appropriate fitness function. Depending on the application, two different fitness functions are available in CGP-IP. For all fitness measures, a lower value indicates a better solution (with 0 being a perfect score).

For many problems, the simplest fitness function is to compute the difference between the image output by an evolved program and a target image on a per pixel basis. In CGP-IP, if this method is used, the fitness of an individual is the average error for all pixels in all images in the training set.

For binary classification problems, the output image from the GP is thresholded and treated as a binary image. Each pixel in the image set is then treated as a binary classification test case. This is then compared to the target image using the Matthews Correlation Coefficient (MCC) (Matthews, 1975; Wikipedia, 2012), which has previously been observed to be useful for classification problems solved using CGP (Harding et al., 2012). First the 'confusion matrix' is found, which is the count of the true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). The MMC is calculated as follows:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

An MCC of 0 indicates that the classifier is working no better than chance. A score of 1 is achieved by a perfect classifier, -1 indicates that the classifier is perfect, but has inverted the output classes. Finally, the fitness of an individual is given by $fitness = 1 - |MCC|$, with values closer to 0 being more fit.

The MCC is insensitive to differences in class size, making it a nice choice for many binary classification tasks. The confusion matrix also allows for easy calculation of the classification rate as a percentage of correctly classified pixels.

(a) Image corrupted with salt and pepper noise.

(b) Output of the evolved filter.

(c) Image corrupted with Gaussian noise.

(d) Output of the evolved filter.

Figure 1-2. Example result for noise removal for both 'Salt and Pepper' and Gaussian noise.

Basic Image Processing: Noise Reduction

To demonstrate CGP-IP, we start with the well studied noise reduction problem. Several key examples also use CGP (Harding, 2008; Smith et al., 2005; Vasicek and Sekanina, 2007; Mart‟nek and Sekanina, 2005), and these examples all use mathematical and logical operators to define a simple convolution operation. Here we show CGP-IP addressing both 'salt and pepper' and Gaussian noise. In keeping with previous work, grey-scale images from the USC dataset are used.

The fitness of an individual is the average pixel difference between the output and the expected image. Ten experiments were run for each type of noise. Programs were trained on 4 images, and tested on 6 unseen images.

For the 'salt and pepper' noise removal, 5% of the pixels are corrupted by setting them to either black or white. For the Gaussian noise problem, noise was added with a standard deviation of 16 (i.e. $\sigma = 16$).

From Table 1-1, we see that CGP-IP compares well to previously published techniques. It is hard to do an exact comparison of the results as previous work may use a different subset of the images; for example in the Gaussian noise

validation set, the mean pixel error per image varies from 8.6 to 14.4, suggesting that the results are highly dependent on the image being processed. From Figure 1-2 it can be seen that the evolved filters remove most of the noise, with only a small amount remaining.

Table 1-1. Fitness results for the noise removal problem. The fitness of an individual is the average pixel difference between the output and the expected image. Results are comparable to previous work on this problem; CGP on GPU (Harding, 2008), CGP on FPGA (Mart‟nek and Sekanina, 2005), IRCGP and ECGP (Smith et al., 2005)

Approach	Salt and Pepper	Gaussian
CGP-IP (Average)	0.57	11.00
CGP-IP (Std. dev.)	0.16	0.53
CGP-IP (Best)	0.41	9.58
CGP on GPU (Average)	0.39	n/a
CGP on FPGA (Best)	0.48	6.43
IRCGP (Best)	n/a	6.32
ECGP (Best)	n/a	6.32

Medical Imaging: Cell Mitosis

Detecting, and then counting, cancer cells undergoing mitosis is useful diagnostic measurement in breast cancer screening. However, the cells are small and have a large variety of shapes. This challenge has led to a competition at the 2012 International Conference on Pattern Recognition (ICPR) [1], where entrants are invited to submit methods for solving this problem. As the competition is still under-way, the results of other participants are currently unknown.

To test the problem with CGP-IP, the training set was sliced in a number of patches (i.e. small sections of the image). Half of the patches contain one or more mitoses to detect, the other half contains randomly selected empty patches. In total 420 images patches were used, with 356 used for training and the remaining 64 reserved for validation. As this is a binary classification problem, the MCC based fitness function was used.

Due to time constraints, CGP-IP was run only 6 times. In future work we will perform a more thorough investigation in the performance of CGP-IP on this problem. Statistical results for these runs are shown in Table 1-2. Figure 1-3 shows the input image, expected and predicted classes for validation images for the best performing individual. These results are based on per-pixel analysis, and suggest that CGP-IP provides excellent segmentation. With additional analysis of the segments, the classification rate of the mitoses can also be de-

[1] The competition details can be seen here: `http://ipal.i2r.a-star.edu.sg/event/icpr-2012`

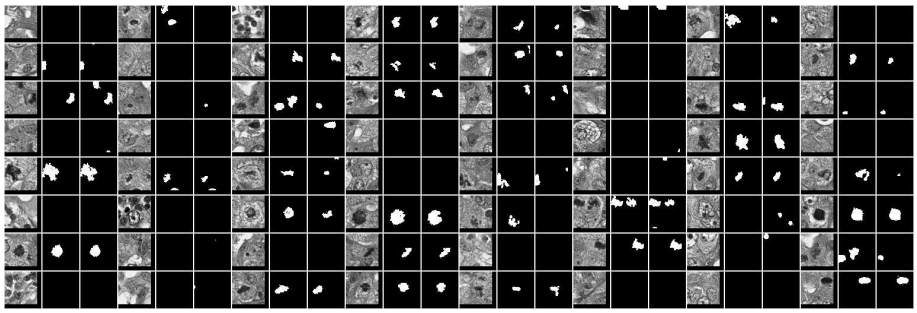
termined. Out of 42 mitoses, CGP-IP correctly identified 36 of them i.e. 86% of the mitoses were correctly identified. There were 12 false-positives, and 6 false-negatives.

It will be interesting to compare CGP-IP to other published methods once the competition results are available.

Table 1-2. Results for the ICPR dataset. The classification accuracy is per-pixel. The fitness function was uses the MCC based score.

	% Classification accuracy	MCC (Fitness)
Average	98	0.36
Minimum	97	0.28
Maximum	98	0.46
Std. Dev.	0.3	0.08

Figure 1-3. Validation set for the ICPR Mitosis dataset. The columns with the images show the input images, the next column to the right shows the expected output and the following column shows the predicted classifications.



Robotics: Vision on the iCub Humanoid Robot

Vision in robotic systems is a challenging problem. In industrial robotics, scenes tend to have well controlled lighting, and the objects being examined are well characterized. Further, the lighting and viewing angles can be optimized to meet the vision requirements. For robots that work in more 'real world' environments, lighting and viewing angles are often variable. Also such vision systems are expected to work in cluttered environments, with lots of different objects visible in the scene.

The iCub humanoid robot is a research platform for investigating various aspects of cognition and control. Here, we apply CGP-IP to detecting objects in images supplied by its cameras. We show that the approach is able to work

robustly in different lighting conditions and when the target object is moved or rotated.

To train CGP-IP, a number of images are collected from the iCub cameras. For each frame, the target object (and other objects in the scene) are repositioned. Hence our training set implicitly contains multiple views of the object with different angles, scales and lighting conditions. Each image in the training set is then hand segmented to highlight the target object. It should be noted that a rough segmentation still allows CGP-IP to learn efficiently, which reduces the need to spend time performing an accurate segmentation.

The filters are trained to produce a binary classification (i.e. target object and, not object) using the MCC based fitness function.

Nine training images were used, with a single object of interest per image, as shown in Figure 1-4. As before, the objects on the table were shuffled between images. Figure 1-5 shows some examples of the evolved filter running on the robot. It can be seen that the evolved filter is able to cope with variations in scale, orientation and lighting.

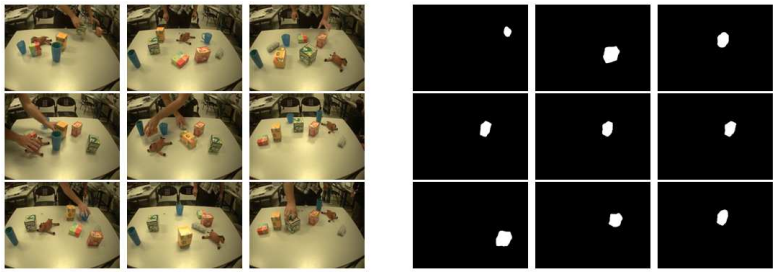


Figure 1-4. Training images for the iCub vision. The set on the left shows the images from the iCub camera. The set on the right shows the binary classification as determined by a human, where a particular box is highlighted in white. For each frame the location of the items on the table is shuffled.

Figure 1-5. Examples of an evolved filter running in real time on the iCub. Considering the limitations of the training set (Figure 1-4), the evolved filters show good ability to generalize to variations in lighting, scale and rotation.

Further examples of using evolved image filters on the iCub robot can be found in (Leitner et al., 2012a), (Leitner et al., 2012b), and (Leitner et al., 2012c).

6. Conclusions

In this chapter we have demonstrated CGP-IP working in several different domains. Whilst the results are hard to compare with other published work, due to lack of suitable datasets being made publicly available, the results indicate that CGP-IP is highly competitive.

Further, we have seen that CGP-IP can work with well-known image processing operations. Using such operations makes the generated programs much more human readable, and compared to other approaches, such as neural networks, this may make GP more attractive in industry. Using OpenCV has the basis for the implementation and also allows for high-speed programs to be found. This is also important in industrial scenarios.

In future publications, we will show CGP-IP working in other domains and give examples of the generated code.

7. *Acknowledgments

The authors would like to thank Julian Miller for his help in refining this paper.

References

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Gonzalez, Rafael C. and Woods, Richard E. (2006). *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Harding, Simon (2008). Evolution of image filters on graphics processor units using cartesian genetic programming. In Wang, Jun, editor, *2008 IEEE World Congress on Computational Intelligence*, pages 1921–1928, Hong Kong. IEEE Computational Intelligence Society, IEEE Press.

Harding, Simon, Banzhaf, Wolfgang, and Miller, Julian F. (2010a). A survey of self modifying cartesian genetic programming. In Riolo, Rick, McConaghy, Trent, and Vladislavleva, Ekaterina, editors, *Genetic Programming Theory and Practice VIII*, volume 8 of *Genetic and Evolutionary Computation*, chapter 6, pages 91–107. Springer, Ann Arbor, USA.

Harding, Simon, Graziano, Vincent, Leitner, Juergen, and Schmidhuber, Juergen (2012). Mt-cgp: Mixed type cartesian genetic programming. In *Genetic and Evolutionary Computation Conference: GECCO 2012, Philidelphia, USA, July 2012*. ACM Press.

Harding, Simon, Miller, Julian F., and Banzhaf, Wolfgang (2010b). Developments in cartesian genetic programming: self-modifying CGP. *Genetic Programming and Evolvable Machines*, 11(3/4):397–439. Tenth Anniversary Issue: Progress in Genetic Programming and Evolvable Machines.

Leitner, J., Harding, S., Frank, M., Førster, A., and Schmidhuber, J. (2012a). Humanoid robot learns visual object localisation. RSS. submitted.

Leitner, J., Harding, S., Frank, M., Førster, A., and Schmidhuber, J. (2012b). icVision: A Modular Vision System for Cognitive Robotics Research. In *International Conference on Cognitive Systems (CogSys)*.

Leitner, J., Harding, S., Frank, M., Førster, A., and Schmidhuber, J. (2012c). Transferring spatial perception between robots operating in a shared workspace. IROS. submitted.

Mart"nek, Tomas and Sekanina, Lukas (2005). An evolvable image filter: Experimental evaluation of a complete hardware implementation in fpga. In Moreno, Juan Manuel, Madrenas, Jordi, and Cosp, Jordi, editors, *ICES*, volume 3637 of *Lecture Notes in Computer Science*, pages 76–85. Springer.

Matthews, B. W. (1975). Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta*, 405(2):442–451.

Miller, J. F. and Smith, S. L. (2006). Redundancy and computational efficiency in cartesian genetic programming. In *IEEE Transactions on Evoluationary Computation*, volume 10, pages 167–174.

Miller, Julian F. (1999). An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In Banzhaf, Wolfgang et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1135–1142, Orlando, Florida, USA. Morgan Kaufmann.

Miller, Julian F., editor (2011). *Cartesian Genetic Programming*. Natural Computing Series. Springer.

Poli, Riccardo (1996). Genetic programming for image analysis. Technical Report CSRP-96-1, University of Birmingham, UK.

Sekanina, Lukas, Harding, Simon L., Banzhaf, Wolfgang, and Kowaliw, Taras (2011). Image processing and CGP. In Miller, Julian F., editor, *Cartesian Genetic Programming*, Natural Computing Series, chapter 6, pages 181–215. Springer.

Shirakawa, Shinichi and Nagao, Tomoharu (2007). Feed forward genetic image network: Toward efficient automatic construction of image processing algorithm. In Bebis, George et al., editors, *Advances in Visual Computing: Proceedings of the 3rd International Symposium on Visual Computing (ISVC 2007) Part II*, volume 4842 of *Lecture Notes in Computer Science*, pages 287–297, Lake Tahoe, Nevada, USA. Springer.

Shirakawa, Shinichi, Nakayama, Shiro, and Nagao, Tomoharu (2009). Genetic image network for image classification. In Giacobini, Mario et al., editors, *Applications of Evolutionary Computing, EvoWorkshops 2009: EvoCOMNET, EvoENVIRONMENT, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoINTERACTION, EvoMUSART, EvoNUM, EvoSTOC, EvoTRANSLOG*, volume 5484 of *Lecture Notes in Computer Science*, pages 395–404, Tubingen, Germany. Springer.

Silva, Sara, Vasconcelos, Maria J., and Melo, Joana B. (2010). Bloat free genetic programming versus classification trees for identification of burned areas in satellite imagery. In Di Chio, Cecilia et al., editors, *EvoIASP*, volume 6024 of *LNCS*, pages 272–281, Istanbul. Springer.

Slany, Karel and Sekanina, Lukas (2007). Fitness landscape analysis and image filter evolution using functional-level CGP. In Ebner, Marc et al., editors, *Proceedings of the 10th European Conference on Genetic Programming*, volume 4445 of *Lecture Notes in Computer Science*, pages 311–320, Valencia, Spain. Springer.

Smith, Stephen L., Leggett, Stefan, and Tyrrell, Andrew M. (2005). An implicit context representation for evolving image processing filters. In Rothlauf, Franz et al., editors, *Applications of Evolutionary Computing, EvoWorkshops2005: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, EvoSTOC*, volume 3449 of *LNCS*, pages 407–416, Lausanne, Switzerland. Springer Verlag.

Spina, T. V., Montoya-Zegarra, Javier A., Falcao, A. X., and Miranda, P. A. V. (2009). Fast interactive segmentation of natural images using the image foresting transform. In *16th International Conference on Digital Signal Processing*, pages 1–8.

Uto, Kuniaki, Kosugi, Yukio, and Ogatay, Toshinari (2009). Evaluation of oak wilt index based on genetic programming. In *First Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing, WHISPERS '09*, pages 1–4.

Vasicek, Z. and Sekanina, L. (2007). Evaluation of a new platform for image filter evolution. In *Adaptive Hardware and Systems, 2007. AHS 2007. Second NASA/ESA Conference on*, pages 577 –586.

Wang2, Jun and Tan, Ying (2011). Morphological image enhancement procedure design by using genetic programming. In Krasnogor, Natalio et al., editors, *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1435–1442, Dublin, Ireland. ACM.

Wijesinghe, Gayan and Ciesielski, Vic (2007). Using restricted loops in genetic programming for image classification. In Srinivasan, Dipti and Wang, Lipo, editors, *2007 IEEE Congress on Evolutionary Computation*, pages 4569–4576, Singapore. IEEE Computational Intelligence Society, IEEE Press.

Wikipedia (2012). Matthews correlation coefficient — wikipedia, the free encyclopedia. [Online; accessed 21-March-2012].

Zhang, Mengjie, Ciesielski, Victor B., and Andreae, Peter (2003). A domain-independent window approach to multiclass object detection using genetic programming. *EURASIP Journal on Applied Signal Processing*, 2003(8):841–859. Special Issue on Genetic and Evolutionary Computation for Signal Processing and Image Analysis.

Index