

Fall 2011

## COSC 5P71 GP: Assignment 2

Instructor: B. Ross

**Due date:** Tuesday November 8, start of class.

**Lates:** Friday November 11 12:00 noon (-25% deduction).

**Goal:** Genetic Programming and Computer Vision.

**Hand in:** A report about your experiments (see assignment 1 for format). Copies of your source and parameter files, and the best 2 evolved results. Examples of training images. Example of testing image with performance output (if applicable). Performance graphs. Use Excel to create performance graphs, to be included in your report.

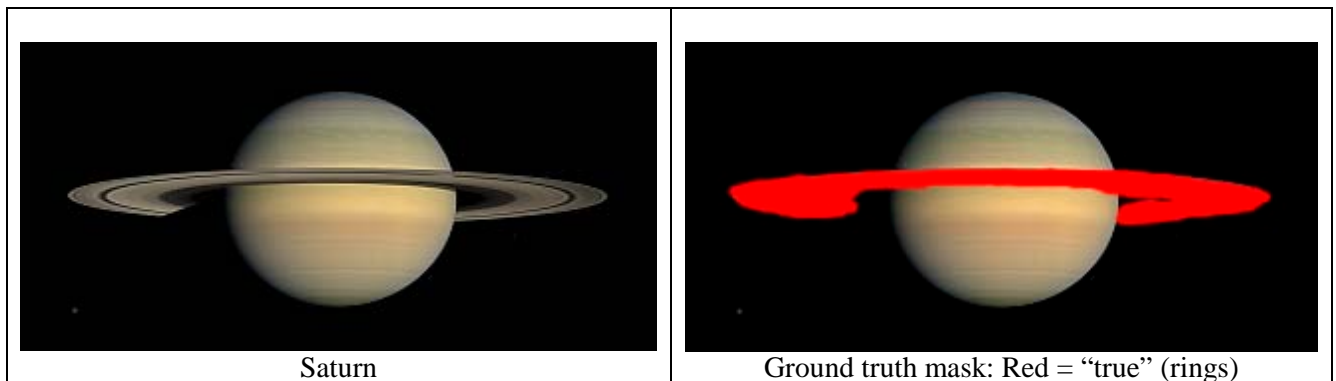
**System:** Any GP system of your choice. For Java image I/O, see:  
<http://java.sun.com/products/java-media/jai/iio.html>

### Problem description:

You are going to use GP for a computer vision application. The problem area to be examined will be an application in **astronomy**. However, you will have the opportunity to choose the exact astronomy problem to examine. The following are a number of problems from which you can choose. Alternatively, you might think of another astronomy problem to do (but please talk to me beforehand about it).

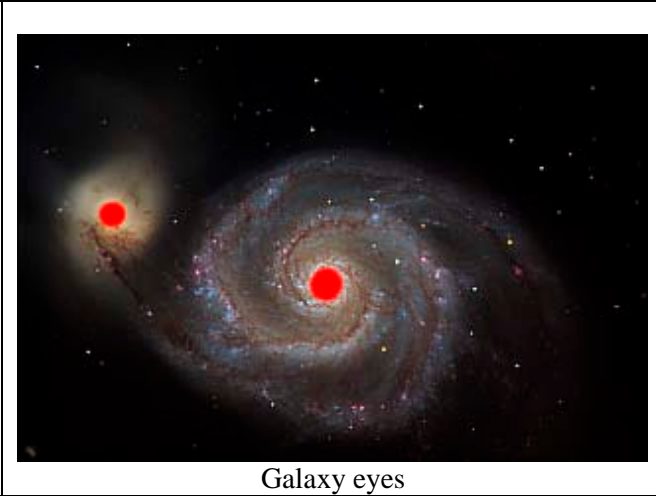
All of the following problems are identification problems, in which a GP is trained to identify the existence or not of some object at a pixel in an image. For example, in the case of identifying Saturn's rings, when an evolved solution is placed over a pixel, the program will either say *true* (it is a ring) or *false* (not a ring). If you run the program over an entire image, hopefully mostly the rings will be coloured *true*. In other words, the program acts like a "Saturn rings filter".

Here are some potential problems from which you can select. The left image is the raw image, while the right image is a solution image (also called *ground truth* image). The red portion marks the area that are to be identified as *true*. Note that you will have to paint these solutions into images, for training and testing purposes!

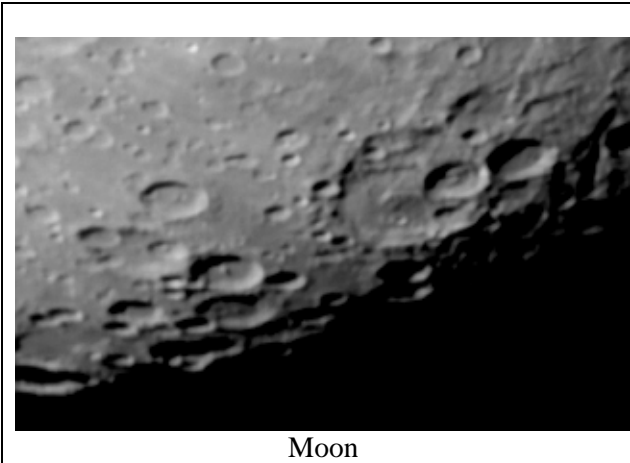




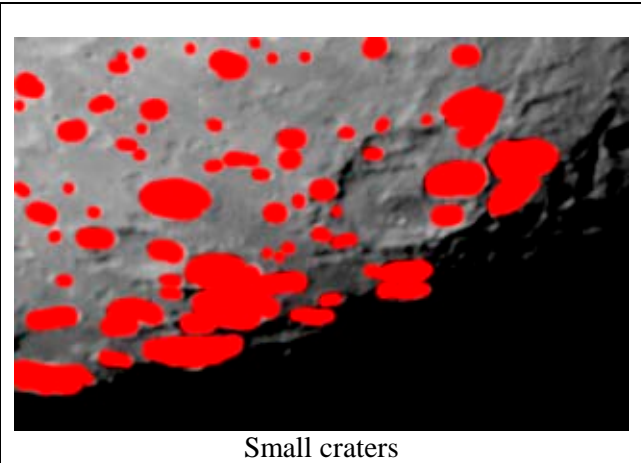
Galaxy



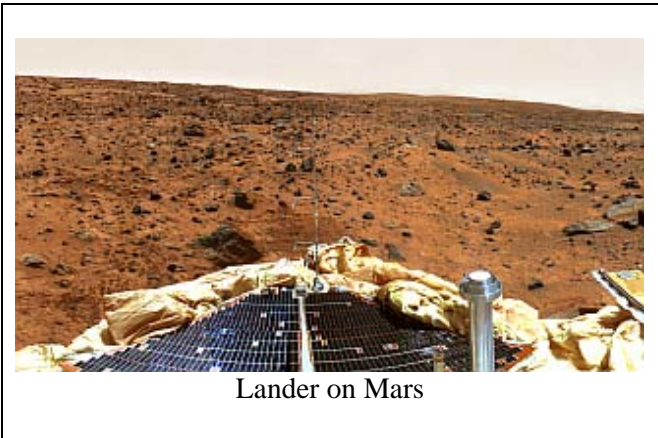
Galaxy eyes



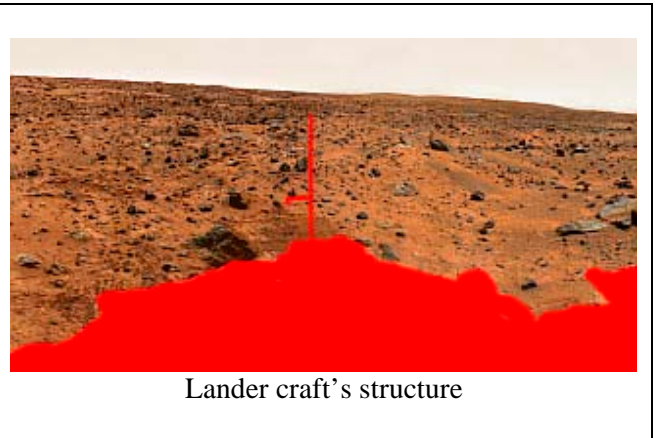
Moon



Small craters



Lander on Mars



Lander craft's structure

You will be evolving a *spatial* image analyzer. This will be an image analyzer that will take in a small area around a given pixel location, perform various processing on that area, and then return a true or false result for that pixel, depending on whether it thinks that pixel resides on the object being detected. Your program will be called by a wrapper code, which will basically scan your GP program

Fall 2011

along all the pixels of an image. Hopefully, your program will be able to detect the object marked areas and mark those pixels as *true*. These are called *true positives*. Pixels that are correctly identified as not the object are *true negatives*. Sometimes errors may occur. Pixels falsely identified as objects are *false positives*, while object pixels that are missed are *false negatives*.

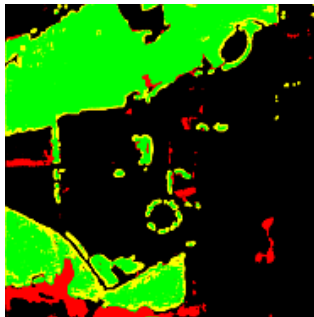
For training and testing, you need to read in the source image (such as the left-image above) and the solution image. Either a B&W or colour RGB source image may be used. The solution image is can be B&W, where the red above can be pure white, and the rest black. (You can convert the above images in Photoshop.) The training process will presume that any portion of the image that is not painted over represents pixels that are to be classified as *false*. Note that using the entire image is probably too much (and very time consuming). A recommended strategy is to use (say) K pixels residing on red areas as positive examples, and L pixels on non-red as negative examples. You can find them randomly, OR pre-select them by hand.

If your GP works perfectly well, it should exactly identify your ground truth image. Hence your fitness function will simply tally the overall percentage of correctly identified pixels. But when computing a fitness value, the fitness should be normalized with respect to total true positives and true negatives; otherwise, it may benefit a GP to simply identify everything as “false”, because the majority of area of an image may not have a galaxy eye!

For testing, you need to find unprocessed pixels of the image, and then run your program on these unseen pixels. You can do this with individually identified pixels (positive and negative cases), and then tally the total number correct, both in terms of true positive, true negative, false positive, and false negative. It is more impressive, however, to do a testing mask analysis, in which your program output is represented by a pixel-by-pixel colour coded scheme, which is compared to a ground-truth mask you have made. Please use these colours:

- Green = true positive
- Black = true negative
- Red = false positive
- Yellow = false negative

This gives you a visual representation of your program’s testing performance. It will let you see exactly what confuses your program, and what you need to correct in the training data. It should be accompanied by numeric scores (% true positive, %true neg, etc.). For example, here is a result image for tree identification problem::



At the end of your run, you should also run your best solution on a **new image!** I have put many others on the web site. You can also find lots more online. Note that you should use the same relative magnification as the image you used for training. All the images used should have roughly the same sized objects, and same quality of photography images. You will have to paint some ground

truth masks for the new image. Compute the true positive and other statistics for this new image during testing, and include it and the colour-coded image in your report.

**GP Language:** I have put online a few papers with some ideas for suitable functions and terminals. You may wish to use a floating point GP expression, in which positive values are *true*, and negative are *false*. Because it is a spatial language, your functions need to perform calculations on **localized areas** of the image, possibly of varying sizes. For example, perhaps you use a 7x7 average. This takes the 7x7 area of pixels around a center pixel, and finds the average intensity. Another possibility is to give the 7x7 average an I and J offset (integers, processed modulo max offsets), which will return the average of the pixel centered I and J away from the current pixel. The calculation returned at the root should be a value interpretable as true (pixel is on an object) or false (not an object). Note that you don't need to have any coordinates passed to your program. You can assume that the current pixel being processed is implicit to all primitives, in other words, global X and Y coordinate is updated and used by all functions as appropriate. Note that the size of your areas will depend upon the sizes of objects in the images you process. For large objects, you may need very large areas to process!

Some other comments:

1. Convert your images to greyscale, and presume that all calculations and processing are performed on greyscale values. (If you are keen, you may wish to use both greyscale AND colour image data. They would be separate pixel information (terminals). You will need separate filters for them as well, if your filters apply to both.) There is more useful information in colour images, by the way.
2. GP runs will speed up if you pre-compute some of the spatial data. For example, the 7x7 average mentioned above could be pre-calculated. This is essentially a pre-applied filter. Then your GP must merely read the 7x7 average value from another array (terminal) when required, rather than re-compute it every time it is needed.
3. There are lots of filters that could help your program, for example, threshold filters. You can pre-calculate any such filters that you want. Then pre-filtered data is simply denoted by new terminals which return appropriate values when referenced. See the example papers for more ideas. By the way, you can generate some filtered output using Photoshop, CorelPaint, or Irfanview ([www.irfanview.com](http://www.irfanview.com)).
4. Converting your source image into lower-resolution versions is also useful. You can do this with averaging pixel values. The GP language can read all these versions as required; they are all different terminals.
5. The ground truth (solution mask) is hand-drawn by yourself. This means there will be some errors, especially around edges. Don't be too concerned with that. It just means that it will be difficult to get totally perfect answers!
6. Don't let your spatial primitives go outside the image boundaries. This may mean you process within the boundaries of the image, to give room for your largest spatial operators.
7. Do multiple runs, and collate the performance of your runs together. Your report should have a table summarizing all the runs, for both the training scores and testing scores. You should report the average best solution scores, as well as the scores for the overall best solution from

Fall 2011

all runs. You should also include performance graphs showing the population fitness and best fitness plotted over generations (averaged for all the runs you do).

8. As done in assignment 1, your report should describe all relevant aspects of your experiment, so that someone else could reproduce your results. Be sure to describe your GP language, as well as your fitness evaluation methodology. Include programs for your 2 best rules obtained.
9. Please note the similarities between this assignment problem, and the diabetes problem in assignment 1.

**References:** <http://www.google.ca> : (find lots of great astronomy photos to use!)