Experiments in Particle Swarm Optimizations with Active Contour Models

By Alan Litteneker, Cory Bradford-Watts, and Karthik Ram

Abstract - A program that implements and is used to compare various methods of delineating object outlines from an image. The techniques used include Active Contour Models (snakes), Gradient Descent (GDA), and Particle Swarm Optimization (PSO). A study of implementation details and experimental results from this program helps to highlight performance differences, advantages, and disadvantages of these various methods.

CONTENTS

I. Introduction

The various approaches to delineating object boundaries in images are key components to visual modeling, specifically for image segmentation, manipulation and identification. One of the most popular bases for image segmentation is Active Contours, a tool that is typically used to find object boundaries using gradient descent. This method has been the seed point of many other visual modelling techniques, many of which purport to improve upon the initial design.

Particle Swarm Optimization of Active Contours is one such technique. PSO is a novel technique implemented during the boundary search phase of snakes, an Active Contour Models typically used to select an object in an image. The PSO method may provide advantages over other boundary finding techniques, and it may also provide an alternative for other methods as well (such as annealing).

However, some of the proposed improvements on snakes proposed by PSO do not seem to take into account all the techniques employed and suggested by the original snake Active Contour Model and may misinterpret the original intent of snakes as a user-initiated and user-directed object selection tool, meaning that these problems were intentional. All-in-all, this creative use of PSO (an artificial intelligence based algorithm) in image object selection provides a good way to explore the original methods and assumptions of the algorithms that continue to form the heart of many visual modelling techniques today.

II. Active Contour Models

IIa. Active Contour Models Introduction

Intuitively, to find an object boundary within an image one needs a metric that can distinguish between a boundary pixel versus others, as well as a mechanism that records the line created by a set of boundary pixels: enter snakes, or Active Contour Models. Snakes, a type of Active Contour model, uses imaginary forces (determined by the implementation) to attract a mathematically defined spline toward a goal state. Typically this goal state is a curve that accurately defines an edge within the picture, but other types of Active Contour techniques have been used in a variety of ways. [1]

The original snakes utilizes user interaction to help select a desired object from an image. Since the desired object can be arbitrary from image to image, this baseline approach is robust and sensical. However, when a similar object is needed to be selected from image to image, techniques can be added that reduce the need for user directed initialization. After initialization, minimization of energy terms helps direct snakes to its optimal solution. Additional user interaction throughout the process can be used to help reach a goal, avoiding non-optimal solutions due to the starting position alone. Throughout this process, snakes as an “active” contour model is continually minimizing its energy functions. [1]

Generally, a snake’s energy functional is made up of internal forces, external constraint forces, and image forces. The internal energy functional is controlled by a term that promotes

behavior that is similar to that of a rubber band ($\alpha(s)$), and one that promotes rod-like behavior ($\boldsymbol{\beta}(s)$).

Parametric Position: $\mathbf{v}(s) = (x(s), y(s))$

$$E^*_{snake} = \int_0^1 E_{snake}(\mathbf{v}(s))\, ds$$
$$= \int_0^1 E_{internal}(\mathbf{v}(s)) + E_{image}(\mathbf{v}(s)) + E_{constraints}(\mathbf{v}(s))\, ds$$

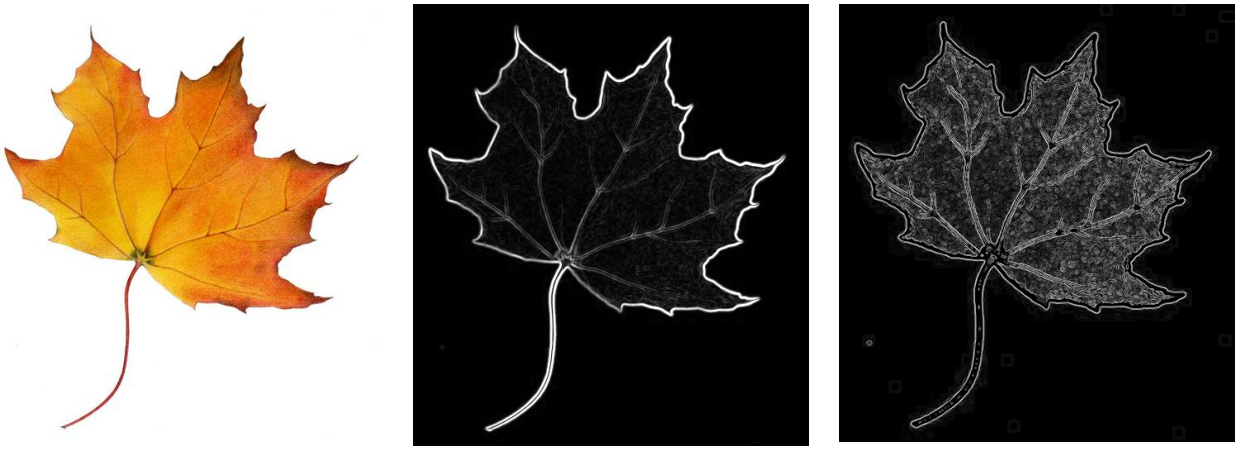
$$E_{internal} = (\alpha(s)|\mathbf{v}_s(s)|^2 + \boldsymbol{\beta}(s)|\mathbf{v}_{ss}(s)|^2/2 \qquad [1]$$

IIb. Active Contour Models Implementation

With the exception of image loading and application window graphics, all of the functionality implemented for this project was built from scratch in Java. Our code structure separates the active contour model energy calculations from the various minimizations, and, as such, its code is relatively simple. Aside from threadsafe mutators and accessors, it mainly consists of a number of methods for calculating the energy and delta energy of a given contour state.

While we were implementing this, however, we also explored a number of methods for gradient construction. We began by implementing a simple kernel convolution system, complete with normalization and simple multithreading. We added several basic operators to this, including Gaussian blurring, high pass filters, and Sobel operators, among others. In addition, we used a sequence of these kernels to build our initial gradient, beginning with a monochromatic sobel operator (sobel of pixel intensities). While this produced gradients that related to light and dark areas, it was not a particularly good metric of edge direction.

To attempt to solve this, we took the gradient of the lengths of these kernels. This approach yielded better results, pulling the snake toward the desired edge. However, this too had an issue. Sections of an edge with much higher gradient values would pull the snake points inwards, creating unwanted snake tension that caused many areas of interest to be hidden. We attempted to counteract this issue by capping the intensity of the gradient values, which appears to have worked.



From left to right: Original image, the first level gradient, and the second gradient, with capped values.

III. Gradient Descent Algorithm

IIIa. GDA Introduction

The gradient descent algorithm is used with Active Contours to help find the boundaries of an image. GDA employs basic gradient descent, which involves taking steps proportional to the negative of the gradient of the function at each control point of the snake. Unfortunately, since GDA progressively "steps" down a curve until reaching a minima and then stops if the next step yields a worse outcome, it has the disadvantage of possibly getting caught in a local minima instead of finding the global minima.

IIIb. GDA Implementation

For our implementation, we tried out different values of the GDA parameters to speed up the algorithm as much as possible. We also added a stepped continuation method to GDA, which is when the snake's gradient begins heavily blurred and is progressively deblurred. We also implemented GDA using vectors and again using matrices (LDL decomposition for easier inversion), in order to compare. We found that our specific matrix implementation was slower by about 15% with 300 control points. This appears to largely be the result of minor oddities in our matrix implementation, as, in theory, a matrix implementation should be slightly faster than a vector implementation if both are serially executed. Rather than pursue further optimizations, we chose to move on to implementing other methods.

IV. Particle Swarm Optimization

IVa. PSO Introduction

As already mentioned, snakes continually moves its control points while minimizing its energy functionals. The search window for each control point is typically small, which can limit the curve's ability to find boundary concavities. Tseng, Hsieh, and Jeng propose in Active Contour Model via Multi-Population Particle Swarm Optimization (2008) that employing multi-population particle swarms for each control point's search method can improve a snake's ability to find boundary concavities while also improving performance [3]. PSO also proposes an improvement on a purported problem of snakes that improper initialization of it can lead to a goal state that is undesired [3]. Our program implements PSO and attempts to test these assertions.

When the algorithm is initiated, the particles in each swarm begin in random positions and velocities. As they move throughout the search space they calculate the fitness of their current position by calculating the energy functional for the curve if their control point were at that location. The search space is limited (or "clipped") to avoid the possible issue of the snake crossing itself [3].

$$|d(\vec{p}_{i,l}, \vec{c}_{i-1}) - d(\vec{p}_{i,l}, \vec{c}_{i+1})| > e$$

where e is a preset threshold, and $d(\vec{p}_{i,l}, \vec{c}_{i-1})$ and $d(\vec{p}_{i,l}, \vec{c}_{i+1})$ are the distances between the particle $\vec{p}_{i,l}$ and the neighboring control points $\vec{c}_{i-1}$ and $\vec{c}_{i+1}$. If the search window clipping inequality is not satisfied the point's position will not be updated [3].

The method keeps in memory the personal best (local) position and the swarm's global best, which is that swarm's control point's location. The acceleration of each move is randomly weighted to push the particle toward the local and the global best value, which should lead the particles to eventually converge around the best value discovered by the swarm globally, enhancing the search in the surrounding area.

$$\vec{v}_i \leftarrow \vec{v}_i + U(0,\varphi_1) \otimes (\vec{p}_i - \vec{x}_i) + U(0,\varphi_2) \otimes (\vec{p}_g - \vec{x}_i)$$
$$\vec{x}_i \leftarrow \vec{x}_i + \vec{v}_i$$

where $\vec{x}_i$ and $\vec{v}_i$ are the position and the velocity of the i^{th} particle, $\vec{p}_i$ and $\vec{p}_g$ are the individual and global best positions, φ_1 and φ_2 are the individuality and sociality coefficients, $U(0,\varphi_a)$ represents a vector of random numbers uniformly distributed in $[0,\varphi_a]$ which is randomly generated at each iteration and for each particle, and $\otimes$ is component-wise multiplication [3]. When the control point reaches a certain level of stability the swarm stops searching. Also, the original paper recommends implementing some search restrictions to keep improper particles from updating, which can help improve performance.

IVb. PSO Implementation

Our PSO algorithm was also built from scratch in Java. In our implementation, each control point has its own swarm where every member is updated during each iteration of the algorithm. Every particle in a swarm is updated based upon the equations referenced above, methodically

looking for personal and global best positions based upon the curve's energy functional. Each swarm continues to step until they have iterated a set number of times without finding a new best value. Since each point changes is considered independently, we found that the algorithm runs pretty slowly (especially compared to GDA), but faster than snakes with a point-by-point square search method. Parallelism and selective particle updates improve our PSO algorithm's speed, but not enough to approach GDA's swiftness.

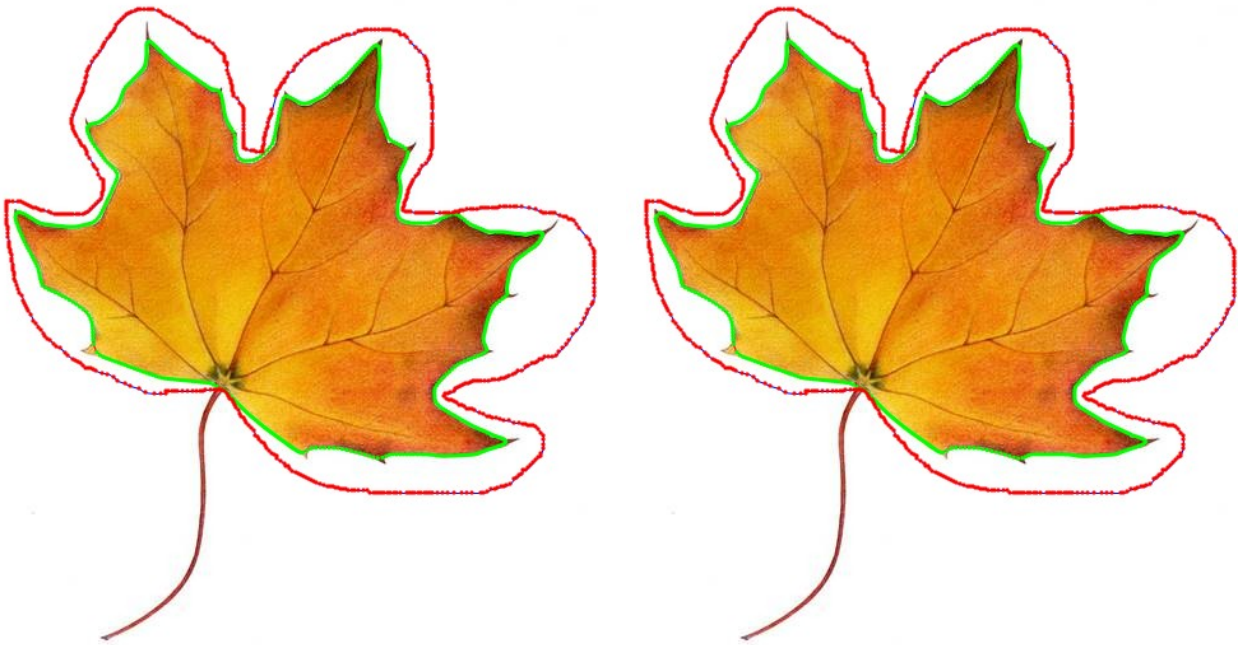
IVc. Usage and Results

After tweaking and much experimentation, we continued to find that our PSO implementation is much slower than our GDA, and surprisingly, it is generally poorer at converging to boundary concavities. We believe that this is largely due to its stochastic nature, having to check multiple energy states before settling on a new global best. However, we were able to replicate many of the results from the original paper. More specifically, PSO performs considerably better than a point-by-point square search method, both at finding concavities and in terms of speed. For comparison, on the same 500 by 500 pixel image with 300 control points, square search took 42 seconds to fully converge, PSO took 31 seconds to converge, and GDA with continuation took 4 seconds to converge.

V. PSO as Stochastic GDA/Annealing Alternative

Over the course of our experiments, after finding that PSO performed quite slowly compared to GDA, we thought of using PSO as an alternative means of a stochastic GDA method, such as with annealing. For this, we implemented the state of the whole system as a PSO, rather than implementing it as a point-by-point function. The position of each particle is a vector of gammas for current descent. The algorithm runs until the same stabilization metric is seen as before. In addition, we also limit the positions (search window clipping) to those with positive coordinates [3].

Positively, with these alterations, the model converges after about the same time as GDA with continuation. This is impressive given the slow speed of our algorithm before the changes, and it shows that PSO as an alternative means of stochastic GDA is a promising method. However, even this method is susceptible to local minima.



Left: Result of GDA with continuation, Right: Result of Stochastic GDA with PSO without continuation.

VI. Future Work

Future combinations of PSO with other optimizations and other techniques might possibly run faster than Gradient Descent, so further experimentation in this regard is a possibility that we believe has not been widely explored. It seems highly likely that some combination of techniques might allow for faster, more accurate edge convergence with less need for user clarification.

Separately, it would be very interesting to see whether a PSO annealing approach would be capable of working with some kind of feature extraction model, such as with a corner detection Hough Transform. As already demonstrated, even continuation methods can be susceptible to sharp edge convexities. It seems possible that a PSO-annealing approach might be more robust with such an approach.

Also, on a more implementation-specific note, we would like to explore why the matrix version of GDA was unexpectedly slower that the vector version. Currently, we assume that the issues arise in oddities in our implementation. We would also like to explore the speed differences further between our different software implementations (GDA, square search, PSO, PSO as annealing alternative) since it is likely that there are numerous avenues for optimization we have not explored. In addition, it would be helpful to explore different ways to avoid the user needing to repeatedly set/change constants and parameters for specific images.

VII. Conclusion

Each of our software implementations performs successfully when it comes to image object selection. After several experiments as detailed above, we conclude that our PSO algorithm seems to be better at detecting boundary concavities as compared to snakes with a point-by-point square search method, but not as compared to GDA in the current implementation. We also show that our PSO algorithm fares very well as an annealing alternative, far better than as a replacement of GDA.

References

[1] - M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active Contour Models," International Journal of Computer Vision 1 (4), pp. 321-331, 1988

[2] - R. Poli, J. Kennedy, and T. Blackwell, "Particle Swarm Optimization - An overview," Swarm Intelligence 1, pp. 33–57, 2007

[3] - C. Tseng, J. Hsieh, and J. Jeng, "Active contour model via multi-population particle swarm optimization," Expert Systems with Applications 36, pp. 5348-5352, 2009