International Journal of
Emerging Research in Management &Technology
ISSN: 2278-9359 (Volume-5, Issue-4)

Research Article | **April 2016**

Image Edge Detection using Adaptive Bacterial Foraging Algorithm with Lifecycle and Social Learning a Review

[1]Tanya Kumari[*], [2]Soni Goyal
[1] Dept of CSE, Delhi Technological University, New Delhi, India
[2] Software Developer, Aricent, New Delhi, India

Abstract—

In digital image processing edge detection is a very important technique which is used for image segmentation, feature extraction etc. An edge is a boundary between an object and the background or between overlapping objects. Edge pixels in an image are identified by sharp change in intensity. Bacterial foraging optimization is one of the important algorithms for edge detection. In this paper a bacterial foraging is introduced which is self-adaptable with lifecycle and social learning. This can be used to improve the various parameters needed in edge detection. A review of some previous work is also presented.

Keywords— Edge detection, bacterial foraging optimization, lifecycle model of bacterium, social learning, adaptive search, BFOLS

I. INTRODUCTION

Edge detection refers to the process of identifying and locating sharp discontinuities in an image. The discontinuities are abrupt changes in pixel intensity which characterize boundaries of objects in a scene. Classical methods of edge detection involve convolving the image with an operator (a 2-D filter), which is constructed to be sensitive to large gradients in the image while returning values of zero in uniform regions.

Variables involved in the selection of an edge detection operator include:

- **Edge orientation:** The geometry of the operator determines a characteristic direction in which it is most sensitive to edges. Operators can be optimized to look for horizontal, vertical, or diagonal edges.
- **Noise environment**: Edge detection is difficult in noisy images, since both the noise and the edges contain high-frequency content. Attempts to reduce the noise result in blurred and distorted edges. Operators used on noisy images are typically larger in scope, so they can average enough data to discount localized noisy pixels. This results in less accurate localization of the detected edges.
- **Edge structure:** Not all edges involve a step change in intensity. Effects such as refraction or poor focus can result in objects with boundaries defined by a gradual change in intensity. The operator needs to be chosen to be responsive to such a gradual change in those cases. Newer wavelet-based techniques actually characterize the nature of the transition for each edge in order to distinguish, for example, edges associated with hair from edges associated with a face.

There are many ways to perform edge detection. However, the majority of different methods may be grouped into two categories:

- **Gradient:** The gradient method detects the edges by looking for the maximum and minimum in the first derivative of the image.
- **Laplacian:** The Laplacian method searches for zero crossings in the second derivative of the image to find edges. An edge has the one-dimensional shape of a ramp and calculating the derivative of the image can highlight its location. Suppose we have the following signal, with an edge shown by the jump in intensity below:

Gradient-based algorithms such as the Prewitt filter have a major drawback of being very sensitive to noise. The size of the kernel filter and coefficients are fixed and cannot be adapted to a given image. An adaptive edge-detection algorithm is necessary to provide a robust solution that is adaptable to the varying noise levels. Canny's edge detection algorithm is computationally more expensive compared to Sobel, Prewitt and Robert's operator. However, the Canny's edge detection algorithm performs better than all these operators under almost all scenarios.

II. PREVIOUS WORK

A single threshold edge detection algorithm based on multi-scale fusion was proposed by W. Yue-Liang, C. Yuan-Da and L. Dun [1]. In this paper they proposed a new single threshold edge detection algorithm based on multi-scale edge eigen value. In the range of effective edge envelope, using finite element edge track algorithm for edge fusion and detection to local edge point according to FE cell topology, they experimented on a single threshold for accurate local different type edge.

Z. Musoromy, F. Bensaali, S. Ramalingam and G. Pissanidis [2] presented a comparison of real-time DSP-based edge detection techniques for license plate detection. In this paper they claimed the use of DSP processors to accelerate

LP (license Plate detection process and achieve the real-time performance at higher LP detection rate using edge detection methods

Op verma et all [3] presented A new approach for edge detection using a combination of bacterial foraging algorithm (BFA) and probabilistic derivative technique derived from Ant Colony Systems. The performance of the proposed technique is compared against that of the traditional edge detectors such as Canny, Edison, Rothwell, Sobel and SUSAN. Method performs better than many other standard methods. However the results show some disconnected and parallel edges.

In the paper by AD Joshi, JV Shinde, Savitribai Phule [4] a technique is presented to be used for detecting edges in color image. The Adaptive threshold Histogram technique is used. It splits up image into three primary colors. As per the color intensity values present in an image three separate histograms are generated for each image. Then cumulative frequencies are calculated and average mean value is generated. Then finally the result is fused. The Experiments shows the superiority of the new method as compared to previous specially in selecting static threshold value which results in over segmentation of image

III. BACTERIAL FORAGING OPTIMIZATION

BFO algorithm was first proposed by Passino [5] in 2002. It is inspired by the foraging and chemotactic behaviors of bacteria, especially the Escherichia coli (E. coli). By smooth running and tumbling, E. coli can move to the nutrient area and escape from poison area in the environment. The chemotactic is the most attractive behavior of bacteria.

However, the original BFO has some shortages: [6]

- Dispersal, reproduction, and elimination each happens; after a certain amount of chemotaxis operations. The appropriate time and method for dispersal and reproduction are important. Otherwise, the stability of the population may be destroyed.
- The tumble angles in the chemotactic phase are generated randomly. As a result, the algorithm is more like a random searching algorithm except it will try further in better directions. The bacteria swarm lacks interaction between individuals. Good information carried by those individuals in higher nutritional areas cannot be shared with and utilized by other bacteria.
- The swim step length in the original BFO algorithm is a constant. In most cases, the bacterium will run one more step if the position is better than its last position. If the swim step is large at the end stage (e.g., larger than the distance between its current position and the optimal point), it will skip the optimal point repeatedly. This will make the bacteria hard to converge to the optimal point.

The pseudocode for original BFO is as follows:-

[Step 1] Initialize parameters n, S, N_c, N_s, N_{re}, N_{ed}, P_{ed}, $C(i)$ (i=1,2,…,S),Θ^i . Where,

n: Dimension of the search space,
S: The number of bacterium,
N_c: chemotactic steps,
N_s: swim steps,
N_{re}: reproductive steps,
N_{ed}: elimination and dispersal steps,
P_{ed}: probability of elimination,
$C(i)$: the run-length unit during each run or tumble.

[Step 2] Elimination-dispersal loop: $l = l+1$.

[Step 3] Reproduction loop: $k = k+1$.

[Step 4] Chemotaxis loop: $j = j+1$.

[sub step **a]** For i =1, 2,…, S, take a chemotactic step for bacteria i as follows.

[sub step **b]** Compute fitness function, $J(i,j,k,l)$.

[sub step **c]** Let $J_{last} = J(i,j,k,l)$ to save this value since we may find better value via a run.

[sub step **d]** Tumble: Generate a random vector.

$$\Delta(i) \epsilon R^n$$

[sub step **e]** Move:Let

$$\theta_i^{t+1} = \theta_i^t + C(i)\emptyset(i) \quad (3.1)$$

$$\emptyset(i) = \frac{\Delta_i}{\sqrt{\Delta_i \Delta_i^T}} \quad (3.2)$$

[substep **g]** Swim:

(**i**) Let $m = 0$ (counter for swim length).

(**ii**) While $m < N_s$ (if have not climbed down too long)

- Let $m = m+1$.
- If $J(i,j+1,k,l) < J_{last}$, let Jlast = $J(i,j+1,k,l)$. then another step of size $C(i)$ in this same direction will be taken as

equation (1) and use the new generated (j+1,k,l) to compute the new $J(i,j+1,k,l)$.

- Else let m = N_s.

[Step 5] If $j < N_c$, go to step 3. In this case, continue chemotaxis since the life of the bacteria is not over.

[Step 6] Reproduction:

[substep **a]** For the given k and l, and for each i = 1, 2, …, S, let J_{health} be the health of the bacteria. Sort bacterium in the order of ascending values.

$$J_{health}^{i} = \sum_{j=1}^{N_C+1} J(i,j,k,l) \tag{3.3}$$

[substep **b]** The S_r bacteria with the highest J_{health} values die and the other S_r bacteria with the best values split and the copies that are made are placed at the same location as their parent.

[Step 7] If $k < N_{re}$ go to step 2. In this case the number of specified reproduction steps is not reached and start the next generation in the chemotactic loop.

[Step 8] Elimination–dispersal: For i = 1, 2, …, S, with probability p_{ed}, eliminate and disperse each bacteria, which results in keeping the number of bacteria in the population constant. To do this, if a bacterium is eliminated, simply disperse one to a random location on the optimization domain. If $l < N_{ed}$, then go to step 2; otherwise end.

IV. ADAPTIVE BFO WITH LIFECYCLE AND SOCIAL LEARNING[6]

To improve the optimization ability of BFO algorithms, many variations are proposed. In the proposed BFOLS algorithm, three strategies are used to improve the original BFO.

A. Lifecycle Model of Bacterium

As mentioned above, in original BFO algorithm, there are three loops for the population. Bacteria will reproduce after N_c times of chemotactic steps and dispersal after N_{re} times of reproduction. As a result, the parameter settings of N_c and N_{re} are important to the performance of the algorithm. Unsuitable parameter values may destroy the chemotactic searching progress .To avoid this, we remove the three loops in BFOLS algorithm. Instead, for each bacterium, we will decide it to reproduce, die, or migrate by certain conditions in the bacteria's cycle.

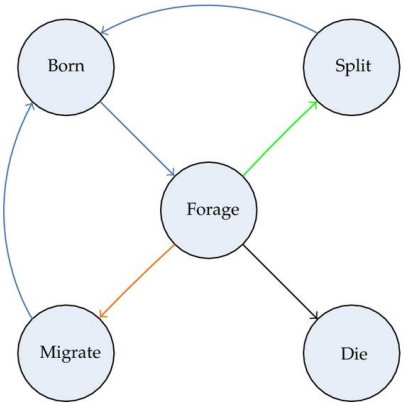

Fig.1 State transition in lifecycle model of bacteria in BFOLS

In the model, a bacterium could be represented by a six-tuple as follows:

B = {P, F, N, T, D, C}, (4.1)

where P, F, N, T, D, C represent the position, fitness, nutrient, state, tumble direction, and step length, respectively.

Define the nutrient updating formula as (4.2). F(last) represents the fitness of the bacterium's last position (for a minimum problem, fitness is larger when the function value is smaller). In initialization stage, nutrients of all bacteria are zero. In the bacterium's chemotactic processes, if the new position is better than the last one, it is regarded that the bacterium will gain nutrient from the environment and the nutrient is added by one. Otherwise, it loses nutrient in the searching process and its nutrient is reduced by one.

$$N(i) = \begin{cases} N(i)+1 \; if \; F(i) > F(last) \\ N(i)-1 \; if \; F(i) < F(last) \end{cases} \tag{4.2}$$

There are five states defined in the lifecycle model: born, forage, split, die, and migrate. That is, T = {Born | Forage | Split | Die | Migrate}. The bacteria are born when they are initialized. Then they will forage for nutrient. In the foraging process, if a bacterium obtains sufficient nutrient, it will split into two in the same position; if the bacterium enters bad area and loses nutrient to a certain threshold, it will die and be eliminated from the population; if the bacterium is with a bad nutrient value but has not died yet, it may migrate to a random position according to certain probability. After split or migrate, the bacterium is regarded as new born and its nutrient will be reset to 0.

The split criterion and dead criterion are listed in Formula (4.3) and (4.4). N_{split} and N_{adapt} are two new parameters used to control and adjust the split criterion and dead criterion. S is the initial population size and Si is the current population size. It should be noticed that the population size will increase by one if a bacterium splits and reduce by one if a bacterium dies. As a result, the population size may vary in the searching process. At the beginning of the

algorithm, as S equals to Si, the bacterium will split when its nutrient is larger than N_{split} and die when its nutrient is smaller than 0. To avoid the population size becoming too large or too small, a self-adaptive strategy is introduced: if Si is larger than S, for each N_{adapt} of their differences, the split threshold value will increase by one. And if S is larger than Si, for each N_{adapt} of their differences, the death threshold value will decrease by one.

$$Nutrient(i) > max\left(N_{split}, N_{split} + \frac{S^i - S}{N_{adapt}}\right) \quad (4.3)$$

$$Nutrient(i) < min\left(0{,}0 + \frac{S^i - S}{N_{adapt}}\right) \quad (4.4)$$

B. Social Learning

In chemotactic steps of original BFO, the tumble directions are generated randomly. Information carried by the bacteria in nutrient rich positions is not utilized. In BFOLS, we assume that all bacteria can memory the best position they have reached and share the information to other bacteria. And in chemotactic steps, a bacterium will decide which direction to tumble using the information of its personal best position and the population's global best position. Based on the assumption, the tumble directions in our BFOLS are generated using (4.5). Where θ_{gbest} is the global best of the population found so far and $\theta_{i,pbest}$ is the i^{th} bacterium's personal historical best.

$$\Delta_i = (\theta_{gbest} - \theta_i) + (\theta_{i,pbest} - \theta_i) \quad (4.5)$$

By social learning, the bacteria will move to better areas with higher probability as good information is fully utilized.

C. Adaptive Search Strategy

There are various step length varying strategies. In BFOLS, we use the decreasing step length. The step length will decrease with the fitness evaluations, as shown in (4.6). C_S is the step length at the beginning, C_e is the step length at the end, nowEva is the current fitness evaluations count, TotEva is the total fitness evaluations. In the early stage of BFOLS algorithm, larger step length provides the exploration ability. And at the later stage, small step length is used to make the algorithm turn to exploitation:

$$C = C_S - \frac{(C_S - C_e) \times nowEva}{TotEva} \quad (4.6)$$

To strengthen the idea further, an elaborate adaptive search strategy is introduced based on the decreasing step length mentioned above. In the new strategy, the bacteria's step lengths may vary from each other. And their values are related with their nutrient, which are calculated using (4.7):

$$C(i) = \begin{cases} \frac{C}{Nutrient(i)} & if\ Nutrient(i) > 0 \\ C & if\ Nutrient(i) < 0 \end{cases} \quad (4.7)$$

The pseudo code of BFOLS algorithm is as follows:-

1. Initialization
2. **While** (termination conditions are not met)
3. S = size of the last population; i = 0;
4. **while** i < S
5. i = i + 1;
6. Jlast = J(i)
7. Generate a tumble angle for bacterium n;
8. Update the position of bacterium n
9. Recalculate the J(i)
10. Update personal best and global best;
11. m = 0
12. **While** (m < N_s)
13. **If** J(i) < Jlast
14. Jlast = J(i)
15. Run one more step using (3.1);
16. Recalculate the J(i);
17. Update personal best and global best;
18. m = m + 1;
19. **Else**
20. m = Ns;
21. **End if**
22. **End while**
23. **If** (Nutrition (i) is larger than split threshold value)
24. Split bacterium i into two bacteria; Break;
25. **End if**
26. **If** (Nutrition (i) is less than dead threshold value)

27. Remove it from the population;
28. i = i − 1; S = S − 1; Break;
29. **End if**
30. **If** (Nutrition (i) is less than 0 and rand < P_{ed})
31. Move bacterium i to a random position;
32. **End if**
33. **End while**
34. **End while**

V. RESULT

The BFOLS algorithm was tested on a set of benchmark functions such as Rosenbrock with dimensions of 2 and 20

The population sizes S of all algorithms were 50. In original BFO algorithm, the parameters are set as follows: N_c *=50,* N_s*=4,* N_{re}=4, N_{ed}*=10,* P_{ed}=0.25**,** C = 0.1, and S_r = S/2 = 25. In our BFOLS algorithm, *N*, N_{re} , N_{ed} , and S_r are no longer needed. N_s = 4, P_{ed} = 0.25. The started step C_S= 0.1(Ub−Lb); ended step C_e= 0.00001(Ub−Lb), where Lb and Ub refer to the lower bound and upper bound of the variables of the problems. This will make the algorithm suitable for problems of different scales. The step of the whole population decreases from C_S to C_e linearly, and step of each bacterium is calculated by (4.7) mentioned above. The values of the two control parameters N_{split} and N_{adapt} are set to be 30 and 5.

Original BFO, PSO, and GA algorithms are used for comparison. With dimension of 2, it outperforms BFO and GA but is little worse than PSO. With dimension of 20, it shows significant better performance than GA and BFO. At the level of α = 0.10, it is significant better than PSO. The varying situations of population size are also tracked. With dimensions of 2 and 20, they show regularities and correspond with the population survival phenomenon in natural. Overall, the proposed BFOLS algorithm is a powerful algorithm for optimization. It offers significant improvements over original BFO and shows competitive performances compared with other algorithms on higher-dimensional problems.

VI. CONCLUSION

In this paper a review of some edge detection methods in images is given along with some basics of edge detection. The basic concept of bacterial foraging optimization has also been explained. The adaptive bacterial foraging algorithm with life cycle and social learning has been introduced for better parameter values and threshold selection in edge detection. This work has been compared with other algorithms such as original BFO, PSO and GA . The BFOLS has shown competitive performance.

REFERENCES

[1] W. Liang, C. Da and L. Dun, "A single threshold edge detection algorithm based on multi scale fusion," *Computational Intelligence and Security, Guangzhou*, vol. 2, pp. 1710-1715, Nov. 2006.

[2] Musoromy, Zuwena, Faycal Bensaali, Soodamani Ramalingam, and Georgios Pissanidis. "Comparison of real-time DSP-based edge detection techniques for license plate detection." *Information Assurance and Security (IAS), 2010 Sixth International Conference* on, pp. 323-328. IEEE, 2010.

[3] Verma, Om Prakash, Madasu Hanmandlu, Puneet Kumar, Sidharth Chhabra, and Akhil Jindal. "A novel bacterial foraging technique for edge detection." *Pattern recognition letters* 32, no. 8 (2011): 1187-1196.

[4] Joshi, A. D., Savitribai Phule, and J. V. Shinde. "A Novel Approach for Color Image Feature Extraction using Swarm Intelligence." *International Journal of Computer Applications* 111.7 (2015): 29-35..

[5] K. M. Passino, "Biomimicry of bacterial foraging for distributed optimization and control," *IEEE Control Systems Magazine*, vol. 22, no. 3, pp. 52–67, 2002.

[6] X. Yan, Y. Zhu, H. Zhang, H. Chen, and B. Niu, "An Adaptive Bacterial Foraging Optimization Algorithm with Lifecycle and Social Learning", *Discrete Dynamics in Nature and Society*, Volume 2012, Article ID 409478, 20 pages, 2012.

