Zuzanna KRAWCZYK, Jacek STARZYŃSKI

Politechnika Warszawska, Wydział Elektryczny

doi:10.15199/48.2017.04.02

Parallel implementation of the Artificial Ant Colony Algorithm applied to medical images segmentation

Abstract. *Artificial Ant Colony algorithm (AAC) can be applied to segmentation of bone structures out of CT data series. AAC procedure produces promising results in regions of adjacent bones and joints which are hard to distinguished by common segmentation algorithms. The article presents parallel implementation of the AAC which allows for significant speed-up of the segmentation procedure. The results of the segmentation for various bone structures in the area of the human pelvis are presented.*

Streszczenie*. Algorytm kolonii mrówkowej (AAC) pozwala na segmentację struktur kostnych z serii obrazów tomografii komputerowej. AAC daje obiecujące wyniki dla przylegających do siebie fragmentów kości i stawów, które trudno rozróżnić przy pomocy często używanych filtrów obrazu. Artykuł przedstawia równoległą implementację algorytmu pozwalającą znacznie przyspieszyć operację segmentacji. Zaprezentowano w nim wyniki algorytmu dla wybranych struktur kostnych w obrębie miednicy. (**Równoległa implementacja algorytmu mrówkowego zastosowanego do segmentacji obrazów medycznych**).*

Keywords: image processing, artificial ant colony algorithm, parallel programming, CT.
Słowa kluczowe: przetwarzanie obrazu, algorytm mrówkowy, programowanie równoległe, CT.

Introduction

Automatic segmentation of medical images is a widely investigated yet not fully solved problem [1]. Techniques that are often applied to the segmentation of computer tomography images include: threshold based, edge detection based, region growing based, deformable model based, fuzzy based or neural network based methods [2, 3]. Most of the above mentioned techniques (except straightforward threshold based segmentation and edge detection algorithms) need prior initialization by an expert user or extensive training on prepared data set.

Cortical parts of bone structures are easily recognizable in CT images due to their high radiodensity. Therefore they can be easily segmented by threshold based algorithms. Trabecular bone regions near the bones boundaries (e.g. the area of hip joint) are much harder to distinguish. Thus, the segmentation result contains often several different bone structures in one region.

The heuristic, iterative method of the Artificial Ant Colony Algorithm (AAC), belonging to the swarm intelligence class of algorithms, delivers promising results in the case of 'difficult' image segmentation described above. However the successful application of the method requires to perform a large number of iterations and thus it is time consuming. The article presents parallel implementation of the AAC [4, 5, 6] with which significant speed-up in comparison to the sequential version has been achieved.

The article is organized as follows: In the first section the idea of the AAC and its version adjusted to the image segmentation task is presented, in the subsequent chapter the implementation details of the parallel version are described. The third section is focused on the example segmentation results in the area of human pelvis and its surrounding. The article ends with conclusions and description of the further work planned.

The Artificial Ant Colony Algorithm

The AAC, first presented in [4], was initially aimed at finding the optimal path in the graph. Further studies [5] and [6] showed the potential of the modified version of the algorithm for image segmentation and edge detection. The application of the AAC to the segmentation of lungs structures was discussed in [7] and in [8] the method was employed to the hippocampus segmentation.

The AAC implements a heuristic method inspired by the behavior of the ant colony. The optimal solution of the problem is found by the group of agents called “ants”, each performing simple set of local operations. The agents do not have any centralized supervisor responsible for their actions.

The version of the algorithm employed in the current paper follows the one described in [6]. In our case, the two-dimensional, digital image is considered as a search space on which ant agents are placed and moving. Each ant is characterized by its current position and direction. In each iteration of the algorithm, an ant deposits a computed value, called pheromone, in the cell at which it currently resides. Values of the pheromone level in the neighborhood of an ant are the only information exchanged between the agents. The probability of an ant to move depends on the ant direction in the previous iteration (it is more probable that the ant will continue its movement in the same direction than that it will perform an abrupt turn) and the pheromone levels deposed in neighboring cells. Unlike in the standard version of the AAC only one ant can occupy one cell at a time.

At the start of the AAC m ants are randomly placed in the array with size equivalent to the image subject to segmentation. In the next step in each of the n iterations, for each ant the following steps are performed:

1. Computation of the possible move directions.
2. Execution of the ant move to one of the neighboring cells (8 adjacent cells) for which the move probability is the highest.
3. Pheromone computation and deposition in the current cell.

After each iteration a small part of the pheromone evaporates from all the cells. The pheromone deposition function is given by the equation (1):

$$T = \eta + p\Delta \,, \tag{1}$$

where η is a small pheromone value deposed in the cell after each iteration, p is the importance coefficient of the delta function and Δ is an author-defined function described by the equation (2):

$$\Delta(i) = \begin{cases} a \;\; if \;\; avg(f) < avg(w) \\ b \;\; if \;\; avg(f) \geq avg(w) \end{cases} , \tag{2}$$

where $avg(f)$ is the average image intensity, $avg(w)$ is the average intensity of the window of pixels with i-th pixel

inside of it. The radius of the window is user-defined, a and b are functions given by the equations (3) and (4):

$$a = round\left(\frac{f(i)}{\max(f)}\right), \tag{3}$$

$$b = \begin{cases} 1 \ \ if \ \ f(i) > avg(w) \\ 0 \ \ if \ \ f(i) \le avg(w) \end{cases}, \tag{4}$$

where $f(i)$ is the intensity of the current pixel and $max(f)$ is the maximal intensity of the whole image. Equation (3) is also the Δ function proposed in [4].

After the final iteration the pheromone image is segmented: pixels with the value higher or equal than an average pheromone value are set to the white color, whereas rest of the pixels is set to the black color.

The algorithm implementation was written in the Python language with the use of numpy module allowing to perform matrix operations in a more efficient manner. The code was run on the set of images created out of CT data series with single image size of 767 on 770 pixels. The average computation time for single image performed on computer with Intel(R) Xeon(R) CPU X5460, 3.16GHz with 4 cores is approximately 435 minutes 54 seconds.

Profiling operation (performed with the cProfile module) shows, that most of the computational time is spend in the method which evaluates delta function and in method responsible for the movement probability computations. Detailed information about computations time is presented in table 1.

Table 1. Profiling results for the image of size 194x189 pixels, 10 algorithm iterations

Procedure	Number of procedure calls	Cumulative execution time [s]
Whole program	1	25.083
Delta function	109990	11.643
Movement probability	109990	9.959
Ant direction computation	109990	1.109

Parallel implementation

In order to speed-up the computation time of the most time consuming operations the program was redesigned into its parallel version.

All of the agents can move along and apply pheromone in the whole search space of the image. Therefore arrays containing pheromone values and map of ant positions should be shared between processes performing computations.

The computations are taken in parallel by number of processes defined by user. The additional method managing the creation of processes and even ants distribution along them was written.

The general scheme of the algorithm can be described as follows: In the sequential, preparation phase, like in the original version of the algorithm, ant agents are randomly placed on the map with a size equivalent to the size of the image. K objects of the class performing algorithm computations, called workers, are created and information about position of ants is copied to the local tables in the processes. Then, every worker performs its own computations, synchronizing changes in critical sections. In each iteration, for every ant in a given worker, following steps are performed:

1. The ant neighborhood is copied to the local array.
2. Computation of three most probable ant moves is performed. (The probabilities are ranked in the descending order.)
3. In the critical section it is checked if ant from another process has not occupied the neighboring destination cell to which agent should be moved.
4. If the cell is not occupied, the ant performs its move. Otherwise the procedure is repeated with the next probable move on the list. If no move is available ant stays in place.
5. The shared map of ants positions is updated in critical section.
6. The pheromone deposition rate is computed.
7. The pheromone map is updated in the critical section.

Only the update of the pheromone on the global map is synchronized. The delta function computations are performed fully in parallel.

The algorithm parallelization was performed with the use of multiprocessing [9] module employed for processes management and ctypes module [10] used in order to create shared arrays. It is noteworthy, that the usage of multiprocessing module allows to eliminate the time overhead generated by the Global Interpreter Lock (GIL) in Python multithreaded programs.

The 5.94 times speed-up was achieved for the image of size 767 on 770 pixels and 5.49 speed-up for the smaller size of task (image of size of 194 on 189 pixels), for 8 parallel processes. The speed-up is defined as the ratio between the execution time of the sequential version of the program and the parallelized one. The details of the parallel algorithm performance are shown in the tables 2 and 3 for medium and small size of images respectively. Figure 1 depicts the execution time of the algorithm in the function of processes number.

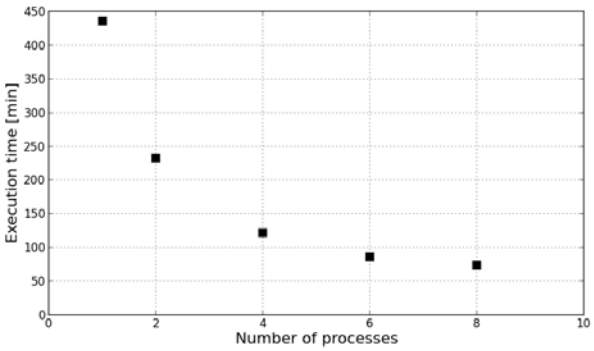

Fig. 1. Computation times for the image of size of 767x770 pixels, lunched for different number of processes on Intel(R) Xeon(R) CPU X5460 processor

Table 2. Performance statistics for the test image of size 767x770, 200 iterations of the algorithm

Processors number	Execution time [min]	Speed-up	Sequential time [min]	Efficiency
Sequential version	435min 54s	-	-	-
2	232min 28s	1.88	14.51	0.94
4	121min 15s	3.6	12.28	0.9
6	86min 17s	5.05	13.63	0.84
8	73min 20s	5.94	18.84	0.74

Table 3. Performance statistics for the test image of size 194x189, 200 iterations of the algorithm

Processors number	Execution time [sec]	Speed-up	Sequential time [sec]	Efficiency
Sequential version	434	-	-	-
2	278	1.56	61	0.78
4	141	3.08	32.5	0.77
6	97	4.47	24.7	0.74
8	79	5.49	24.75	0.69

The more parallel processes run the program, the more agents from different processes can change the state of the neighborhood of the ant. Procedure enters the critical section more often. Thus, the efficiency of the algorithm (understood as a ratio between the speed-up and number of parallel processes) is dropping with the greater number of processes.

Figure 2 shows the speed-up difference between the medium and small problem size. The parallelization procedure is slightly more effective for the larger computational tasks, which is consistent with the theory.

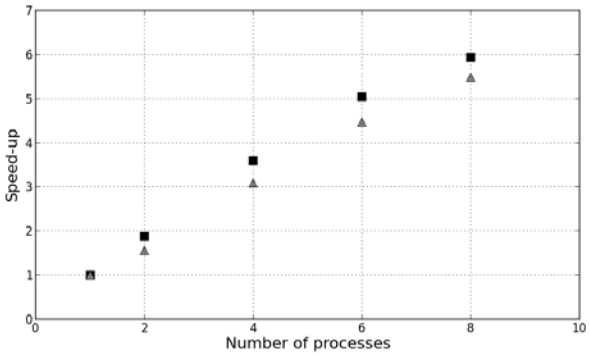

Fig. 2. Speed-up for medium (767x770 pixels) images – squares and small (103x125 pixels) images – triangles

Example results

The algorithm has been tested on the set of images obtained from the CT data series including the full size images and smaller cut pieces containing only critical regions with adjacent bones. The results of the procedure were compared with other segmentation method - Otsu thresholding and with Canny edge detection procedure.

Otsu method calculates optimum threshold separating pixel classes, in such a way that their intra-class variance is minimal. In the paper multithreaded Otsu threshold was used, in order to detect and eliminate from the segmentation process artificial elements visible on the CT scans like treatment table or markers, which are brighter than bones. Canny edge detection is multi-stage, state of the art, edge detection algorithm, able to detect wide range of edges in the image. In the computations only the pixels with intensity values between 70 and 250 were taken into consideration (again in order to eliminate additional artificial elements on the image and soft tissue), the pixel values were normalized to the range between 0 and 255. The variance used in the Gaussian filter applied at the beginning of the Canny edge detection procedure was set to 2.0. In case of both above methods their implementations from SimpleITK library were employed [10].

Figure 3 presents cut input data, whereas the results of the segmentation algorithms are shown at figures 4 and 5.

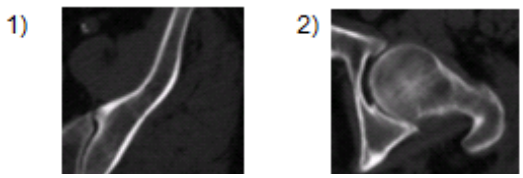

Fig.3. The original input data: 1) part of ilium and sacrum, 2) femur bone

It can be noticed that the segmentation performed with the use of the AAC is better suited to the detection of bone edges than other two methods. In case of the Otsu filter some parts of bones are under or over segmented. The Canny edge detector filter produces to many additional edges. AAC is also able to separate two bone structures placed close to each other.

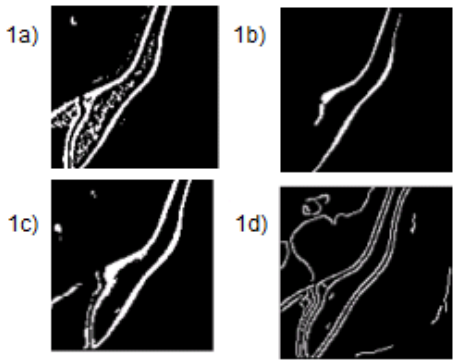

Fig.4. Ilium cross-section: results of the image processing obtained with: 1a) AAC, 1b) Otsu segmentation applied to the whole image, before the piece was cut, 1c) Otsu segmentation applied to the cut piece, 1d) Canny edge detection

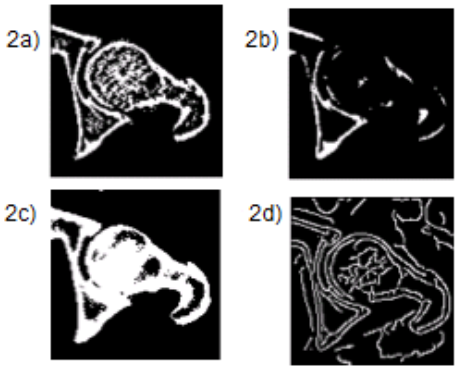

Fig.5. Femur cross-section: results of the image processing obtained with: 2a) AAC, 2b) Otsu segmentation applied to the whole image, before the piece was cut, 2c) Otsu segmentation applied to the cut piece, 2d) Canny edge detection

In case of full size images other structures than bones were detected by the AAC as well. In order to filter them out the outcome of the AAC was combined with the results of other procedures: Otsu thresholding, edge detection algorithm and AAC with b function (Eq. 4) treated as the delta function. Only the pixels segmented by 3 out of 4 different algorithms where set visible in the final image. The result of the procedure is presented at the figure 6.

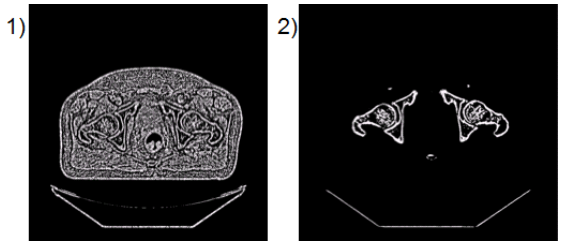

Fig.6. 1) The result of AAC segmentation, after 100 iterations; radius of the window of delta function is equal to 5. 2) Segmentation result after the fusion with other algorithms

Estimation of ants number

The number of ant agents placed in the search space is determined as a 30% of the image pixels number,

according to the parameters used in [5]. Too little number of ants can result in under-segmentation of the image. Excessive amount of agents unnecessarily increases computation time, while obtaining similar segmentation result. Segmentation results obtained with the different number of ants are presented in the figure 7.

Fig.7. Segmentation result after 200 iterations with the number of ants equal to: 1) 15% 2) 30% 3) 50% of the pixels number in the image

The alternative strategy of determining the number of ants in the search space was performed. The colony was modified in such a way that, each ant has its "life force" - LF attribute, which describe how strong the agent is. At the beginning of the simulation the attribute is set to the value of 10, for each ant. If ant has deposed only a little amount of pheromone (the η factor) in the cell, the LF is decreased by the value of 0.5 (the ant has not found the food in the current iteration and it is becoming weaker). If the amount of the deposed pheromone is bigger than the η factor the LF is increased by the value of 1 (the ant is getting stronger and is able to survive by the greater amount of iterations). If the value of the LF attribute is lower than 0, the ant dies and is removed from the search space. In such way only the ants that have reached the neighborhood of the bone regions are able to survive.

Figure 8 shows that ants quickly recognize regions to be segmented, however great amount of ants stays in the regions without pheromone (out of bone regions).

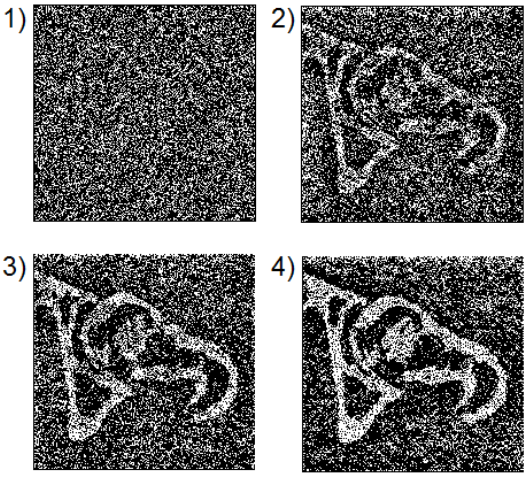

Fig.8. Ants position at the search space before: 1) first iteration 2) 15 iteration 3) 30 iteration 4) 100 iteration

In order to obtain good quality of segmentation and eliminate unproductive ants, the described above strategy of dying colony was used with the high starting number of ants (50% of the pixels number). After 20 iterations the population of ants drops drastically (Fig.9.): for the small test image from the number of 18333 ants to 7520 ants, leaving only agents around the bone area. In further iterations the number of ants drops slightly to the 5880 agents after 200 iterations. The procedure has allowed to speed-up the computations with the use of 8 parallel processes, approximately by the factor of 1.38 for small size images and by the factor of 1.92 for the medium images.

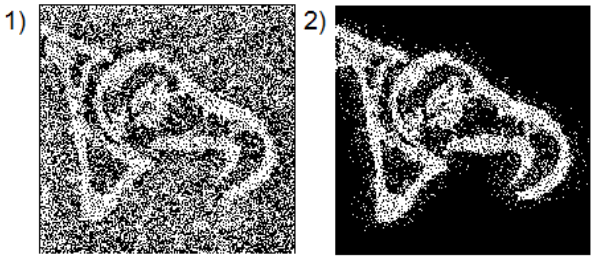

Fig 9. Segmentation procedure with starting number of ants equal to 50% of pixel numbers. Ants positions after a) 19 iterations b) 20 iterations

Conclusions

The Artificial Ant Colony algorithm gives promising results in the area of adjacent bone structures or joints which are hard to be distinguish by common segmentation algorithms. The parallel implementation of the algorithm has shorten its execution time even over 5 times. The application of the 'dying-colony' strategy has improved this result. In order to obtain further speed-up implementation of the AAC for the GPU is considered.

In the future, research on the skeletonization of the obtained segmentation results will be conducted.

The CT data obtained due to courtesy of Maria Skłodowska-Curie Memorial Cancer Center.

***Authors**: mgr inż. Zuzanna Krawczyk, dr hab. inż. Jacek Starzyński Politechnika Warszawska, Instytut Elektrotechniki Teoretycznej I Systemów Informacyjno-Pomiarowych, ul. Koszykowa 75, 00-662 Warszawa, E-mail: zuzanna.krawczyk@ee.pw.edu.pl, jstar@ee.pw.edu.pl*

LITERATURE

[1] Sharma N,, Aggarwal LM., Automated medical image segmentation techniques. *Journal of Medical Physics / Association of Medical Physicists of India* , nr 35 (2010), 3-14
[2] Prasantha H.S. et. al. *Medical Image Segmentation*, International Journal on Computer Science and Engineering (IJCSE), Vol. 02, No. 04 (2010), 1209-1218
[3] McInerney T., Terzopoulos D., Deformable Models in Medical Image Analysis: A Survey, *Medical Image Analysis*, 1(2) (1996), 91-108
[3] M. Dorigo, Optimization, Learning and Natural Algorithms, *PhD thesis*, Politecnico di Milano, Italy, (1992)
[4] Ramos V., Almeida F., Artificial Ant Colonies in Digital Image Habitats - A Mass Behaviour Effect Study on Pattern Recognition, *Proc. of ANTS 2000 - 2nd Int. Works. on Ant Algorithms (From Ant Colonies to Artificial Ants), Marco Dorigo, Martin Middendorf, Thomas Stuzle (Eds.)*, Brussels, Belgium, 7-9 Sep. (2000), 113
[5] A. V. Alvarenga, Artificial Ant Colony: Features and applications on medical image segmentation, *2011 Pan American Health Care Exchanges*, Rio de Janeiro, (2011), 96-101
[6] Cerello P. et. al., 3-D object segmentation using ant colonies, *Journal Pattern Recognition*, 43 (2010), Issue 4, April, 1476-1490
[7] Fiorina E., Fully automated hippocampus segmentation with virtual ant colonies, *Computer-Based Medical Systems (CBMS), 2012 25th International Symposium on*;, 978-1-4673-2049-8 (2012), 1-6
[8] https://docs.python.org/2/library/multiprocessing.html
[9] https://docs.python.org/2/library/ctypes.html
[10] http://www.simpleitk.org/