Watershed implementation by union-find

Jaemin Lee, 2019-13351 이재민
Department of Electrical and Computer Engineering,
Seoul National University
Seoul, Korea
dinojaemin@snu.ac.kr

Abstract

The watershed algorithm is an algorithm for image segmentation in computer vision task. Conventional watershed algorithms such as immersion and rainfall utilize priority queue to segment the image. However, the conventional methods were time-consuming. Therefore, we invented a new watershed algorithm based on union-find algorithm. Union-find algorithm's time-complexity is lesser than O(N) while priority queue has O(Nlog(N)) complexity. We proved that our new algorithm operates much faster than conventional algorithms and durable to the image form. In further project we will prove the superiority of the new algorithm in the case of bigger images. And find the break-through for the early stopping incompetency.

1. Introduction

1.1. Image segmentation

Image segmentation, a fundamental technique in computer vision and digital image processing, involves dividing a digital image into distinct segments or regions, which are essentially sets of pixels. The primary objective of image segmentation is to streamline and potentially transform the representation of an image into a more interpretable and analyzable form.[1]

Its common application involves identifying objects, as well as delineating boundaries such as lines and curves within images. To accomplish this, image segmentation assigns a specific label to each individual pixel, thereby grouping pixels with similar characteristics together.

1.2. Watershed algorithm

The watershed algorithm is a process applied to a grayscale image. It draws its name metaphorically from a geological watershed or drainage divide, which acts as a boundary between adjacent drainage basins. Like a topographic map, the watershed transformation interprets the image as a representation of elevation, where the brightness of each point represents its height. It identifies the lines that follow the crests of ridges in the image.[2]

To determine watersheds, existing approaches utilize a priority queue container. The time complexity of sorting and storing N elements in this container is O(Nlog(N)).

1.3. Watershed algorithm by immersion

The initial method for determining watersheds is referred to as immersion. It was originally proposed by Beucher and Lantuejoul in 1979 with the intention of contour detection in images.[3] Immersion follows a bottom-up methodology, commencing with the identification of local minima.

ALGORITHM 1: IMMERSION BASED WATERSHED ALGORITHM.

```
// Step 1: Initialization
1. Create an empty queue Q
2. Create a label matrix M with the same size as the input image
3. Initialize a variable "label" to 0

// Step 2: Identify regional minima
4. For each pixel P in the input image:
    - Find the neighbors of P (e.g., 8-connectivity or 4-connectivity)
    - Check if P is a regional minimum (i.e., lower than all its neighbors)
    - If P is a regional minimum:
        - Assign a unique label to P in the label matrix M
        - Add P to the queue Q

// Step 3: Flooding process
5. While Q is not empty:
    - Dequeue a pixel P from Q
    - For each neighbor N of P:
        - If N is unlabeled in the label matrix M:
            - Label N with the same label as P in the label matrix M
            - Add N to the queue Q

// Step 4: Post-processing
```

```
6. For each pixel P in the input image:
    - If P is unlabeled in the label matrix M:
        - Assign a new label to P in the label matrix M

// Step 5: Output
7. Return the label matrix M
```

1.4. Watershed algorithm by rainfall

The second strategy is called rainfall. Be a contrast to Immersion, Rainfall is a top-down approach which starts with finding local maxima.[4]

ALGORITHM 2: RAINFALL BASED WATERSHED ALGORITHM.

```
// Step 1: Initialization
1. Create a priority queue Q to store pixels based on their grayscale intensity values
2. Create a label matrix M with the same size as the input image
3. Initialize a variable "label" to 0

// Step 2: Assign initial labels
4. For each pixel P in the input image:
    - Assign a unique label to P in the label matrix M
    - Add P to the priority queue Q with priority equal to the grayscale intensity value of P

// Step 3: Flooding process
5. While Q is not empty:
    - Dequeue a pixel P from Q
    - For each neighbor N of P:
        - If N is unlabeled in the label matrix M:
            - Label N with the same label as P in the label matrix M
            - Set the priority of N in Q to be the maximum of its current priority and the grayscale intensity value of N
            - Add N to Q

// Step 4: Post-processing
6. For each pixel P in the input image:
    - If P is unlabeled in the label matrix M:
        - Assign a new label to P in the label matrix M

// Step 5: Output
7. Return the label matrix M
```

1.5. Disjoint Set & union-find

In the realm of computer science, a disjoint-set data structure, also known as a union-find data structure, is a mechanism for storing a collection of sets that do not overlap with each other. Similarly, it maintains a division of a set into separate and non-overlapping subsets. This data structure offers functionalities such as adding new sets, merging sets (combining them into their union), and determining a representative member of a set. The ability to efficiently determine whether two elements belong to the same or different sets is facilitated by the last operation.

Disjoint-set forests was initially described by Bernard A. Galler and Michael J. Fischer in 1964.[5]

Later, in 1973, Hopcroft and Ullman established that their time complexity is bounded by $O(\log^*(N))$, where $\log^*(N)$ represents the iterated logarithm of N.[6]

In 1975, Robert Tarjan became the first to prove the upper bound of $O(m\alpha(N))$ (inverse Ackermann function) on the algorithm's time complexity.[7]

In 1989, Fredman and Saks provided evidence that any disjoint-set data structure must access at least $\Omega(\alpha(N))$ (amortized) words per operation.[8] This discovery confirmed the optimality of the data structure.

1.6. Analyzing conventional algorithms

Existing methods which are utilizing priority queue divide steps. First, they find local minima or maxima of an image and then recursively collect the pixels around them. These approaches do not utilize the full information gained from first the step (finding local minima or maxima) where they only use the Boolean information whether the center pixel of searching window was local minima or maxima. Also queueing every pixel to make few pixels into one segment might be time consuming method.

Finding local minima from image consumes only $O(N)$ time-complexity. Therefore, it would not affect the total time-complexity of algorithm.

However, the number of local minima can influence the running time of the conventional algorithms. Because of the priority queue size handling, the image that has more local minima would maintain large-size queue for longer time and consumes more time which represents the dependence on the image form.

1.7. Introducing a new algorithm

Utilizing every information of searching windows in first step as the parent node setting and labeling without priority by merging will much more time-efficient method. Since union-find can merge every N pixel in $O(N)$ time at worst case while priority queue does in $O(N\log(N))$ time-complexity.

Also, the union-find based method would be durable to the image form.

2. Methods

2.1 Implementations

We implemented watershed algorithms with both methods; existing and union-find. The Immersion based watershed algorithm was implemented as the existing method utilizing priority queue. And we newly invented an algorithm with union-find function.

METHOD1: IMMERSION BASED WATERSHED ALGORITHM.

```
// Step 1: Initialization
1. Create an empty queue Q
2. Create a label matrix M with the same size as the input image
3. Initialize a variable "label" to 0

// Step 2: Identify regional minima
4. For each pixel P in the input image:
    - Find the neighbors of P (8-connectivity)
    - Check if P is a regional minimum
    - If P is a regional minimum:
        - Assign a unique label to P in the label matrix M
        - Add P to the queue Q

// Step 3: Flooding process
5. While Q is not empty:
    - Dequeue a pixel P from Q
    - For each neighbor N of P:
        - If N is unlabeled in the label matrix M:
            - Label N with the same label as P in the label matrix M
            - Add N to the queue Q

// Step 4: Post-processing
6. For each pixel P in the input image:
    - If P is unlabeled in the label matrix M:
        - Assign a new label to P in the label matrix M

// Step 5: Output
7. Return the label matrix M
```

METHOD2: UNION-FIND BASED WATERSHED ALGORITHM.

```
// Step 1: Initialization
1. Create a label matrix M with the same size as the input image
2. Initialize a variable "label" to itself.

// Step 2: Identify regional minima
4. For each pixel P in the input image:
    - Find the neighbors of P (8-connectivity)
    - Check which pixel is the window minimum
    - If P is not a regional minimum:
        - Assign the window minimum pixel as the parent of P

// Step 3: Merging process

5. For each pixel P find and merge to the root
      (greatest parent) without sequence
// Step 4: Output
6. Return the label matrix M
```

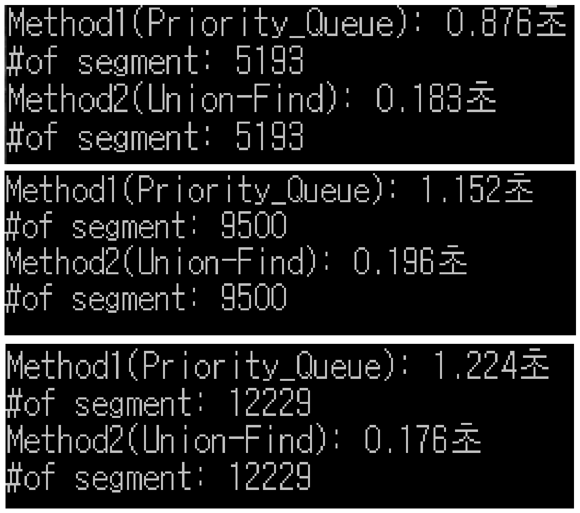
2.2 Implementation details & environment

We segmented 256x256 gray scale images with both algorithms. We used clock function of c++ measure the running time of both algorithms.

Also, after task finished and clock stopped, we calculated the number of segments formed from the watershed tasks by the number of unique values contained in output matrix. The verified the output matrix by allocating the same value of root to each pixel and print the image.

We implemented every code in the “Visual Studio Community 2019 ver.16.7.2”. The language was c++ and the os was windows10 pro 64bit.

3. Results



```
Method1(Priority_Queue): 0.876초
#of segment: 5193
Method2(Union-Find): 0.183초
#of segment: 5193
```

```
Method1(Priority_Queue): 1.152초
#of segment: 9500
Method2(Union-Find): 0.196초
#of segment: 9500
```

```
Method1(Priority_Queue): 1.224초
#of segment: 12229
Method2(Union-Find): 0.176초
#of segment: 12229
```

Figure 1: The duration and result number of segments after segmenting three 256x256 gray images. (From the top; Lena, Caman, NWU)

Figure 2: Original images before the segmentation. (From the left; Lena, Caman, NWU)

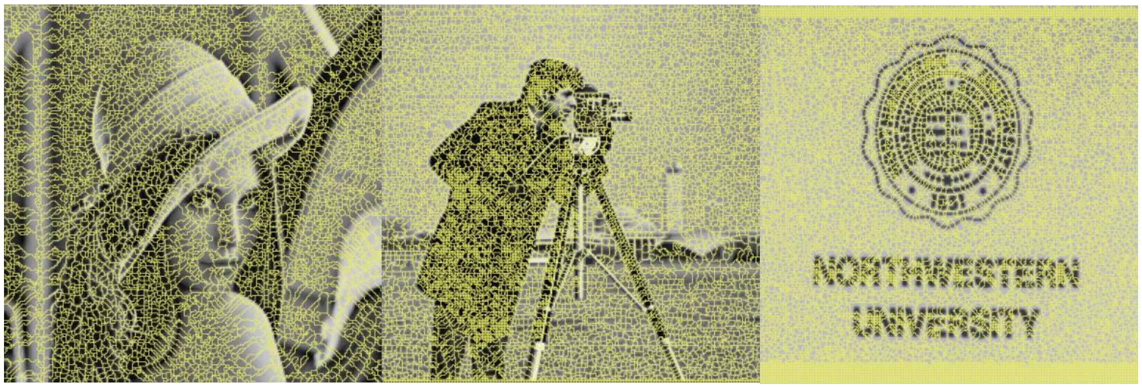


Figure 3: Result images after the segmentation. (From the left; Lena, Caman, NWU)

3.1. Segmentation results

The numbers of segments and printed images of both methods were same which means the two algorithms make same outputs.

3.2. Running time

The watershed algorithm based on union-find consumed meaningfully lesser time than the watershed algorithm based on immersion.

The first method required about a second to execute watershed segmentation task. However, our new method required lesser than 1/5 time of formal method.

3.3. Durability

Priority-queue based algorithm's running time had dependence on number of segments of watershed results. However, union-find based algorithm had no dependency on number of segments of watershed results. Therefore, we could surmise that the speed differential between conventional methods and the new method would grow as the image size get larger.

4. Discussion

4.1. Achievements

As it was predicted by the time-complexity of algorithms, the real running time of watershed segmentation was improved significantly by implementing the algorithm based on Disjoint set & union-find.

Union-find based algorithm also consumed consistent time independent of the image's form while the conventional algorithm had strong dependency. The independency would save the time meaningfully when process parallelly. (e.g pre-processing for further computer vision task or machine learning task)

4.2. Limitations

However, union-find based algorithm couldn't perform kind of early stopping technic. Union-find based algorithm is incompetent to stop the process before merge every pixel to get segments of particular value of pixels. Early stopping is often applied to gradient image of original images effectevely.

Due to the codes which were wrote only for the 256x256 gray images we couldn’t ensure the better performance for the larger images.

5. Conclusion

The Research proved that the newly invented watershed algorithm has better general performance compared to the conventional algorithms. Especially it would be useful for the preprocessing of the further research on complex computer vision tasks.

However, experiment on larger images is required further. Also, the new algorithm will not be able to substitute conventional algorithms completely due to the incompetence on early stopping for gradient images.

References

[1] L. G. Shapiro and G. C. Stockman, *Computer vision*. Pearson, 2001.

[2] R. Romero-Zaliz and J. Reinoso-Gordo, "An updated review on watershed algorithms," *Soft computing for sustainability science*, pp. 235-258, 2018

[3] B. Beucher and C. Lantuejour, "Use if watersheds in contour detection," in *The international workshop on image processing: real time and motion detection/estimation, Rennes, France*, 1979.

[4] J. Cousty, G. Bertrand, L. Najman, and M. Couprie, "Watersheds, minimum spanning forests, and the drop of water principle," 2007.

[5] B. A. Galler and M. J. Fisher, "An improved equivalence algorithm," *Communications of the ACM*, vol. 7, no. 5, pp. 301-303, 1964.

[6] J. E. Hopcroft and J. D. Ullman, "Set merging algorithms," *SIAM Journal on Computing*, vol. 2, no. 4, pp. 294-303, 1973.

[7] R. E. Tarjan and J. Van Leeuwen, "Worst-case analysis of set union algorithms," *Journal of the ACM (JACM)*, vol. 31, no. 2, pp. 245-281, 1984.

[8] M. Fredman and M. Saks, "The cell probe complexity of dynamic data structures," in *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, 1989, pp. 345-354.