Theoretical Computer Science 412 (2011) 1419–1433

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

The optimal All-Partial-Sums algorithm in commutative semigroups and its applications for image thresholding segmentation

Xie Xie [a,*], Jiu-Lun Fan [a], Yin Zhu [b]

[a] School of Communications and Information Engineering, Xi'an University of Posts and Telecommunications, Xi'an 710061, China
[b] Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, China

ARTICLE INFO

Keywords:
Partial sum
Straight-line program computation model
Image segmentation
Thresholding method
Piling algorithm

ABSTRACT

The design and analysis of multidimensional All-Partial-Sums (APS) algorithms are considered. We employ the sequence length as the performance measurement criterion for APS algorithms and corresponding thresholding methods, which is more sophisticated than asymptotic time complexity under the straight-line program computation model. With this criterion, we propose the piling algorithm to minimize the sequence length, then we show this algorithm is an optimal APS algorithm in commutative semigroups in the worst case. The experimental results also show the algorithmic efficiency of the piling algorithm. Furthermore, the theoretical works of APS algorithm will help to construct the higher dimensional thresholding methods.

1. Introduction

Image segmentation has long been an important problem in image processing and computer vision, whose purpose is to extract objects from their background. The thresholding technique [19,22,13] is a major image segmentation scheme, which usually segments the image with few thresholds. Many thresholding methods have been proposed, and they are usually categorized in six groups [22], such as clustering-based methods, and entropy-based methods. However, most thresholding methods have one thing in common, that is they choose the optimal thresholds by maximizing or minimizing a certain objective function.

We usually distinguish the thresholding methods by their objective functions. For example, Otsu's thresholding method [15] employs the between-class variance while the entropic thresholding method [18,12] employs entropy as the objective function. The objective function must be sensible, since it decides the effect of image segmentation. Therefore, the previous papers mainly focused on how to construct objective functions for getting sound thresholds, and various objective functions are proposed to enhance the effectiveness of segmentation [22].

The thresholding methods can also be distinguished by the dimension of the histogram. The proposed thresholding methods are classified to one-dimensional (1D) thresholding methods [15,18,12,16,3], two-dimensional (2D) thresholding methods [1,5,14,20,21] and three-dimensional (3D) thresholding methods [11,8], which are based on the 1D, 2D and 3D histogram respectively. In fact, these methods can be treated as special cases in certain dimensions. We will introduce the generalized neighborhood gray values and then construct the n-dimensional (nD) histogram, with which we can propose a general framework of the nD thresholding method.

Generally speaking, the higher dimensional thresholding methods can obtain better segmentation results compared with the lower ones, but they also require more resources, especially CPU time. Many algorithms [10,25,8,23,24] have been proposed to reduce the running time. With the current CPU, the common 2D thresholding methods cost little, while the 3D

* Corresponding author. Tel.: +86 029 85383404.
E-mail addresses: xiexiexx@gmail.com (X. Xie), jiulunf@xupt.edu.cn (J.-L. Fan), yinz@cse.ust.hk (Y. Zhu).

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.11.039

thresholding methods still require much CPU time. For example, when we implement the 3D Otsu's thresholding method, its running time is still a barrier in practice, and we may need some optimization algorithms such as shuffled frog-leaping algorithm to solve it [24]. When we want to construct the higher dimensional (e.g. 4D) thresholding methods, we have to enhance their time performance firstly. Therefore, we must focus on the efficiency of these methods, especially in the higher dimensional cases. In other words, we need to view them in an algorithmic perspective.

Design and analysis are two important issues in the area of algorithms. Although various algorithms for thresholding methods have been proposed, there is no systematic analysis of them. More importantly, the asymptotic time complexity is not sufficient to measure the time performance of different algorithms for thresholding methods. For example, if we segment an image which has L gray levels, the time complexity of algorithms in [8,23] are both $O(L^3)$, while the algorithm in [23] can reduce the running time by about a half compared with the algorithm in [8]. Therefore, we need new criterion for measuring the time performance of these algorithms before any further theoretical works of them.

We also need to know the special properties of the algorithms for solving thresholding methods. Since such thresholding methods are always associated with the cumulative distribution function, it means we need to calculate lots of partial sums. On the other hand, the optimal thresholds for segmentation are usually obtained by an exhaustive search [10,25,8,23], thus the key point of algorithms for solving thresholding methods is to calculate all the partial sums. Although the partial sum problem is a classic problem and has been studied for decades, most previous works of the partial sum problem such as [9,26,4,7,17] focused on the 1D case and took the asymptotic time complexity as the measurement criterion, thus these works are not suitable for the nD thresholding method, especially in the higher dimensional cases.

In this paper, we will abstract the multidimensional (or nD) All-Partial-Sums (APS) problem from the nD thresholding method and use the sequence length as the measurement criterion in the algebraic framework. Under the straight-line computation model, we will define the nD APS problems in the different universes, especially in the commutative semigroup. More importantly, it is an extremely useful but difficult problem to find how well the algorithms for solving APS problem can do. We will propose a piling algorithm to enhance the performance of general nD thresholding method and show its optimality in commutative semigroups in the worst case. The piling algorithm is suitable to solve APS problems in any dimensional case, and we will see the algorithms in [25,23] are special cases of our algorithm.

We organize this paper as follows. In Section 2, we introduce the nD thresholding method and give a simple definition of the APS problem. In Section 3, we define the multidimensional APS problem in $\mathbb{R}$ and present a new performance measurement criterion under the straight-line program computation model. In Section 4, we define the multidimensional APS problem in commutative semigroups and propose an optimal APS algorithm in the worst case. Finally, we show the optimality of our algorithm in Section 5 and discuss experimental results in Section 6.

2. nD thresholding method

In this section, we first construct the nD histogram based on the generalized neighborhood gray values. With some examples of thresholding methods in the different dimensions, we illustrate the importance of partial sums and then formalize the APS problem. Finally, we present a general framework of the nD thresholding method.

Now we introduce some basic notations. Let $\mathcal{L} = \{0, 1, \ldots, L-1\}$ denote the set of all the gray levels. For any digital image $\mathcal{I}$ of size $M \times N$, let $\Delta_{d\times d}(x, y)$ denote the $d \times d$ neighborhood of pixel (x, y). The gray value at pixel (x, y) of $\mathcal{I}$ is denoted as $f(x, y)$, while the mean gray value and the neighborhood gray median in the $\Delta_{d\times d}(x, y)$ are denoted as $g(x, y)$ and $h(x, y)$ respectively.

2.1. The nD histogram

We know the ranges of $f(x, y)$, $g(x, y)$ and $h(x, y)$ are all $\mathcal{L}$. In fact, these values can be treated as a certain kind of information measure of $\Delta_{d\times d}(x, y)$.

Definition 1 (*Generalized Neighborhood Gray Value*)**.** Function $\tilde{G}(x, y)$ is called the generalized neighborhood gray value, if $\tilde{G}(x, y)$ is an integer function of the gray values in $\Delta_{d\times d}(x, y)$ and the range of $\tilde{G}(x, y)$ is $\mathcal{L}$.

Now we know $f(x, y)$, $g(x, y)$ and $h(x, y)$ are all generalized neighborhood gray values, and we can find more generalized neighborhood gray values according to the above definition.

Example 1. Let $f_{\max}(x, y)$ be the maximum gray value while $f_{\min}(x, y)$ the minimum gray value in $\Delta_{d\times d}(x, y)$, then we know $f_{\max}(x, y)$, $f_{\min}(x, y)$ and $[(f_{\max}(x, y) + f_{\min}(x, y))/2]$ are all generalized neighborhood gray values.

Definition 2 (*nD Histogram*)**.** Given generalized neighborhood gray values $\tilde{G}_1(x, y)$, $\tilde{G}_2(x, y)$, $\ldots$, $\tilde{G}_n(x, y)$, the nD histogram is defined as the set $\{(c_1, c_2, \ldots, c_n, p_{c_1,c_2,\ldots,c_n}) | (c_1, c_2, \ldots, c_n) \in \mathcal{L}^n\}$, where $p_{c_1,c_2,\ldots,c_n}$ is the frequency of the event $\cup_{i=1}^{n}\{\tilde{G}_i(x, y) = c_i\}$.

The common 1D, 2D and 3D histograms of image $\mathcal{I}$ are defined based on $f(x, y)$, $\big(f(x, y), g(x, y)\big)$ and $\big(f(x, y), g(x, y), h(x, y)\big)$.

Example 2 (*1D Histogram*)**.** Let $\tilde{G}_1(x, y) = f(x, y)$, then we can define the set $\{(i, p_i) | i \in \mathcal{L}\}$ as a 1D histogram.

Example 3 (*2D Histogram*). Let $\tilde{G}_1(x, y) = f(x, y)$ and $\tilde{G}_2(x, y) = g(x, y)$, then we can define the set $\{(i, j, p_{ij})|(i, j) \in \mathcal{L}^2\}$ as a 2D histogram.

Example 4 (*3D Histogram*). Let $\tilde{G}_1(x, y) = f(x, y)$, $\tilde{G}_2(x, y) = g(x, y)$ and $\tilde{G}_3(x, y) = h(x, y)$, then we can define the set $\{(i, j, k, p_{ijk})|(i, j, k) \in \mathcal{L}^3\}$ as a 3D histogram.

Many thresholding methods such as [22,20,21,11,8] based on the above 1D, 2D, and 3D histograms are presented. The difference between these methods are the objective function and the dimension, while the common property of many objective functions is to calculate lots of partial sums. We will illustrate it by some examples in the different dimensions.

2.2. An example of a 1D thresholding method

The 1D Otsu's method [15] is a 1D thresholding method based on the 1D histogram $\{(i, p_i)|i \in \mathcal{L}\}$ in Example 2. The objective function of this method is the between-class variance $\sigma_B^2(t)$, which can be rewritten as

$$\sigma_B^2(t) = \nu_0(t)\Big(\frac{f_0(t)}{\nu_0(t)} - \mu_f\Big)^2 + \big(1 - \nu_0(t)\big)\Big(\frac{\mu_f - f_0(t)}{1 - \nu_0(t)} - \mu_f\Big)^2, \tag{1}$$

where

$$\nu_0(t) = \sum_{i=0}^{t} p_i, \tag{2}$$

$$f_0(t) = \sum_{i=0}^{t} i p_i, \quad \mu_f = f_0(L - 1). \tag{3}$$

The optimal global threshold is

$$t^* = \operatorname{Arg}\max_{t \in \mathcal{L}} \big\{\sigma_B^2(t)\big\}, \tag{4}$$

which can be obtained by Alg-1D-Example.

Alg-1D-Example

```
1  for t ∈ L
2      Calculate ν0(t) and f0(t)
3  μf = f0(L − 1)
4  for t ∈ L
5      Calculate σB²(t) by Eq. (1)
6      Search for t* by comparing σB²(t)
```

We can see $\nu_0(t)$ and $f_0(t)$ are both partial sums of the index variable i. If these partial sums are obtained, the objective function can be calculated directly.

2.3. An example of a 2D thresholding method

The 2D Tsallis–Havrda–Charvát entropic method [21] is a 2D thresholding method based on the 2D histogram $\{(i, j, p_{ij})|(i, j) \in \mathcal{L}^2\}$ in Example 3. This method segments the image with the Tsallis–Havrda–Charvát entropies.

Given a positive real parameter $\alpha \neq 1$, let

$$H_b^\alpha(t_1, t_2) = \frac{1}{\alpha - 1}\left(1 - \sum_{i=0}^{t_1}\sum_{j=0}^{t_2}\Big(\frac{p_{ij}}{\nu_0(t_1, t_2)}\Big)^\alpha\right) \tag{5}$$

and

$$H_w^\alpha(t_1, t_2) = \frac{1}{\alpha - 1}\left(1 - \sum_{i=t_1+1}^{L-1}\sum_{j=t_2+1}^{L-1}\Big(\frac{p_{ij}}{1 - \nu_0(t_1, t_2)}\Big)^\alpha\right), \tag{6}$$

where

$$\nu_0(t_1, t_2) = \sum_{i=0}^{t_1}\sum_{j=0}^{t_2} p_{ij}. \tag{7}$$

The objective function of the 2D Tsallis–Havrda–Charvát entropic method [21] is $\Phi_\alpha(t_1, t_2)$, whose definition is

$$\Phi_\alpha(t_1, t_2) = H_b^\alpha(t_1, t_2) + H_w^\alpha(t_1, t_2) + (1 - \alpha)H_b^\alpha(t_1, t_2)H_w^\alpha(t_1, t_2). \tag{8}$$

The optimal global thresholds are

$$(t_1^*, t_2^*) = \operatorname{Arg}\max_{(t_1, t_2) \in \mathcal{L}^2} \big\{\Phi_\alpha(t_1, t_2)\big\}, \tag{9}$$

which can be obtained by Alg-2D-Example.

ALG-2D-EXAMPLE

1 **for** $(t_1, t_2) \in \mathcal{L}^2$
2 Calculate $\nu_0(t_1, t_2)$
3 **for** $(t_1, t_2) \in \mathcal{L}^2$
4 Calculate $1-(\alpha-1)H_b^\alpha(t_1, t_2)$ and $1-(\alpha-1)H_w^\alpha(t_1, t_2)$
5 // It means we can obtain $H_b^\alpha(t_1, t_2)$ and $H_w^\alpha(t_1, t_2)$
6 **for** $(t_1, t_2) \in \mathcal{L}^2$
7 Calculate $\Phi_\alpha(t_1, t_2)$ by Eq. (8)
8 Search for (t_1^*, t_2^*) by comparing $\Phi_\alpha(t_1, t_2)$

We can also see $\nu_0(t_1, t_2)$, $1-(\alpha-1)H_b^\alpha(t_1, t_2)$ and $1-(\alpha-1)H_w^\alpha(t_1, t_2)$ are all partial sums of index variables i, j. Note that $1-(\alpha-1)H_w^\alpha(t_1, t_2)$ is the reverse partial sum. These partial sums can directly lead to the objective function.

2.4. An example of a 3D thresholding method

The 3D Otsu's method [11,8,24] is a 3D thresholding method based on the 3D histogram $\{(i, j, k, p_{ijk})|(i, j, k) \in \mathcal{L}^3\}$ in Example 4. This method uses the trace of between-class scatter matrix $\mathrm{tr}\big(S_B(t_1, t_2, t_3)\big)$ as the objective function. For simplicity, we denote (t_1, t_2, t_3) by $\boldsymbol{t}$, then the objective function can be rewritten as

$$\mathrm{tr}\big(S_B(\boldsymbol{t})\big) = \frac{\big(f_0(\boldsymbol{t}) - \nu_0(\boldsymbol{t})\mu_f\big)^2 + \big(g_0(\boldsymbol{t}) - \nu_0(\boldsymbol{t})\mu_g\big)^2 + \big(h_0(\boldsymbol{t}) - \nu_0(\boldsymbol{t})\mu_h\big)^2}{\nu_0(\boldsymbol{t})\big(1-\nu_0(\boldsymbol{t})\big)}, \tag{10}$$

where

$$\nu_0(\boldsymbol{t}) = \sum_{i=0}^{t_1}\sum_{j=0}^{t_2}\sum_{k=0}^{t_3} p_{ijk}, \tag{11}$$

$$f_0(\boldsymbol{t}) = \sum_{i=0}^{t_1}\sum_{j=0}^{t_2}\sum_{k=0}^{t_3} i p_{ijk}, \quad \mu_f = f_0(L-1, L-1, L-1), \tag{12}$$

$$g_0(\boldsymbol{t}) = \sum_{i=0}^{t_1}\sum_{j=0}^{t_2}\sum_{k=0}^{t_3} j p_{ijk}, \quad \mu_g = g_0(L-1, L-1, L-1), \tag{13}$$

$$h_0(\boldsymbol{t}) = \sum_{i=0}^{t_1}\sum_{j=0}^{t_2}\sum_{k=0}^{t_3} k p_{ijk}, \quad \mu_h = h_0(L-1, L-1, L-1). \tag{14}$$

The optimal global threshold is

$$\boldsymbol{t}^* = (t_1^*, t_2^*, t_3^*) = \mathrm{Arg}\max_{\boldsymbol{t}\in\mathcal{L}^3}\Big\{\mathrm{tr}\big(S_B(\boldsymbol{t})\big)\Big\}, \tag{15}$$

which can be obtained by ALG-3D-EXAMPLE. Note that the segmentation process uses the thresholds t_1^*, t_2^*, t_3^*.

ALG-3D-EXAMPLE

1 **for** $\boldsymbol{t} \in \mathcal{L}^3$
2 Calculate $\nu_0(\boldsymbol{t}), f_0(\boldsymbol{t}), g_0(\boldsymbol{t}), h_0(\boldsymbol{t})$
3 $\mu_f = f_0(L-1, L-1, L-1)$
4 $\mu_g = g_0(L-1, L-1, L-1)$
5 $\mu_h = h_0(L-1, L-1, L-1)$
6 **for** $\boldsymbol{t} \in \mathcal{L}^3$
7 Calculate $\mathrm{tr}\big(S_B(\boldsymbol{t})\big)$ by Eq. (10)
8 Search for $\boldsymbol{t}^*$ by comparing $\mathrm{tr}\big(S_B(\boldsymbol{t})\big)$

We can still see $\nu_0(\boldsymbol{t})$, $h_0(\boldsymbol{t})$, $g_0(\boldsymbol{t})$ and $f_0(\boldsymbol{t})$ are all partial sums of index variables i, j, k. These partial sums form the objective function.

2.5. Thresholding segmentation and partial sums

Few thresholds may not be sufficient to support effective image segmentation, because many images always have rich information. The higher dimensional thresholding methods with more thresholds might be a good choice.

When we consider nD thresholding methods, higher dimensional histograms are needed firstly. Then we need one sensible function $F(t_1, t_2, \ldots, t_n)$ defined in $\mathcal{L}^n$. Finally we can segment an image $\mathcal{I}$ into two classes $\mathcal{C}_0$ and $\mathcal{C}_1$ by maximizing or minimizing the objective function $F(t_1, t_2, \ldots, t_n)$ based on the nD histogram of $\mathcal{I}$.

With more generalized neighborhood gray values in $\Delta_{d\times d}(x, y)$, we can obtain the higher dimensional histograms.

Example 5 (*4D Histogram*). Let $\tilde{G}_1(x, y) = f(x, y)$, $\tilde{G}_2(x, y) = g(x, y)$, $\tilde{G}_3(x, y) = h(x, y)$ and $\tilde{G}_4(x, y) = [(f_{\max}(x, y) + f_{\min}(x, y))/2]$, then we get a 4D histogram.

The trace of between-class scatter matrix $\mathrm{tr}\big(S_B(\boldsymbol{t})\big)$ in Eq. (10) can be generalized to the nD case, which can be employed as an objective function.

Example 6. Given a 4D histogram, let $F(t_1, t_2, t_3, t_4)$ be $\mathrm{tr}\big(S_B(t_1, t_2, t_3, t_4)\big)$ in the 4D case, then a 4D thresholding method is constructed.

We can see the higher dimensional thresholding method is more complex, thus it is important to find its bottleneck. As we have seen in procedures Alg-1D-Example, Alg-2D-Example and Alg-3D-Example, the processes of calculating the object function are direct, while the processes of calculating those partial sums cost more. In general, the optimal thresholds are obtained only after we know all the objective function values of the index variables in the given region. On the other hand, the structure of histogram decides many objective functions can be rewritten as a function combined with some partial sums, and one can find more examples from [22] to check it. Therefore, the key point of getting the optimal thresholds is to calculate all the partial sums in the objective function efficiently.

The presented thresholding methods [10,25,11,8] have noticed the importance of the partial sums and algorithmic efficiency, but they did not focus on the theoretical foundation of this problem. More importantly, when we treat the higher dimensional histogram, the time performance of calculating partial sums would be a barrier. Here our task is to calculate all the partial sums of index variables in a given region, which can be called the multidimensional All-Partial-Sums (APS) problem. Any algorithm $\mathcal{A}$ for solving the APS problem is named as an APS algorithm, and we should know how well the APS algorithm can do.

We will analyze the APS problem and propose a general APS algorithm in Sections 3 and 4. It should be pointed out that the universe of partial sum values is very important, since such partial problems usually concern the universe $\mathcal{U}$ and the algebraic operations in $\mathcal{U}$. The description of the APS problem here is not sufficient to get an in-depth analysis, thus we will present the strict definitions of the APS problem in different universes before the discussion of the corresponding APS algorithms.

3. The APS problem in $\mathbb{R}$

In this section, we describe the APS problem in $\mathcal{U} = \mathbb{R}$ strictly, and give the analysis of some APS algorithms with the new criterion. Furthermore, the worst case of an APS algorithm is discussed.

3.1. Problem abstraction

Here are the notations used throughout the following sections.

Definition 3 (*Lexicographic Order*). The lexicographic order $\preccurlyeq$ means

$$\begin{aligned}&(e_0, e_1, \ldots, e_{n-1}) \preccurlyeq (e'_0, e'_1, \ldots, e'_{n-1})\\ &\Leftrightarrow (e_0 \le e'_0) \wedge (e_1 \le e'_1) \wedge \cdots \wedge (e_{n-1} \le e'_{n-1}).\end{aligned} \tag{16}$$

Definition 4 (*Integer Grid Graph*). Let $\boldsymbol{e} = (e_0, e_1, \ldots, e_{n-1}) \in \mathbb{N}^n$. Given $\boldsymbol{e}_{\min} \in \mathbb{N}^n$ and $\boldsymbol{e}_{\max} \in \mathbb{N}^n$ ($\boldsymbol{e}_{\min} \preccurlyeq \boldsymbol{e}_{\max}$), the integer grid graph $\mathcal{N}$ is defined as

$$\mathcal{N} = \{\boldsymbol{e} | \boldsymbol{e}_{\min} \preccurlyeq \boldsymbol{e} \preccurlyeq \boldsymbol{e}_{\max}, \boldsymbol{e} \in \mathbb{N}^n\}. \tag{17}$$

We know $(\mathcal{N}, \preccurlyeq)$ is partially ordered and $\mathcal{N}$ is the super cube bounded with the lower bound $\boldsymbol{e}_{\min}$ and the upper bound $\boldsymbol{e}_{\max}$. Note that some elements of $\mathbb{N}^n$ cannot be selected as the bounds of $\mathcal{N}$, because $\boldsymbol{e}_{\min} \preccurlyeq \boldsymbol{e}_{\max}$ requires that they must be compared by lexicographic order $\preccurlyeq$.

When calculating the partial sums, we work with some quantitative values instead of the points in $\mathcal{N}$, then a quantitative function is needed.

Definition 5 (*Quantitative Function from Integer Grid Graph*). The quantitative function q of the APS problem in $\mathbb{R}$ is defined as $q : \mathcal{N} \to \mathbb{R}$.

The partial sums must be composed of the elements from $q(\mathcal{N})$, where $q(\mathcal{N})$ denotes the image of $\mathcal{N}$ under q.

Definition 6 (*Partial Sum*). The partial sum function $S_q : \mathcal{N} \to \mathbb{R}$ is defined as

$$S_q(\boldsymbol{e}) = \sum_{\boldsymbol{e}_{\min} \preccurlyeq \boldsymbol{e}' \preccurlyeq \boldsymbol{e}} q(\boldsymbol{e}'). \tag{18}$$

The reverse partial sum

$$S_q^{\leftarrow}(\boldsymbol{e}) = \sum_{\boldsymbol{e} \preccurlyeq \boldsymbol{e}' \preccurlyeq \boldsymbol{e}_{\max}} q(\boldsymbol{e}') \tag{19}$$

is also useful, which can be treated in a very similar way to the partial sum.

Example 7. Many 2D thresholding methods [5,10,25,20,21] employ a very important function $\nu_0(t_1, t_2)$, whose definition is

$$\nu_0(t_1, t_2) = \sum_{i=0}^{t_1} \sum_{j=0}^{t_2} p_{ij}. \tag{20}$$

Suppose $n = 2$, $\boldsymbol{e}_{\min} = (0, 0)$, $\boldsymbol{e}_{\max} = (L-1, L-1)$, and $q(i, j) = p_{ij}$, then $S_q(t_1, t_2) = \nu_0(t_1, t_2)$.

We have illustrated that the objective function of the nD thresholding method can be combined with some partial sum functions and reverse partial sum functions, so the key is how to obtain all the partial sums mapped from $\mathcal{N}$.

Problem 1 (*The Multidimensional APS Problem in* $\mathbb{R}$)**.** Given $\mathcal{N}$, q and the corresponding S_q, we need to calculate $S_q(\mathcal{N})$, where $S_q(\mathcal{N})$ denotes the image of $\mathcal{N}$ under S_q.

Since the APS problem in the thresholding method has some unique properties, it is important to give a sophisticated analysis of the running time of such algorithms rather than measure them with asymptotic analysis.

3.2. The performance measurement criterion

We first introduce the straight-line program computation model [2]. Under this model, the procedure of algorithm $\mathcal{A}$ can be represented as the sequence of operations

$$E_1, E_2, \ldots, E_\lambda, \ldots, E_{\tau(\mathcal{A})}, \tag{21}$$

where $\tau(\mathcal{A})$ is the sequence length of $\mathcal{A}$, and E_λ is the λth operation ($1 \le \lambda \le \tau(\mathcal{A})$). In general, the operation E_λ is defined as

$$T_\lambda = L_\lambda \circ_\lambda R_\lambda, \tag{22}$$

where T_λ, L_λ, R_λ and $\circ_\lambda$ are the result, the (left) operation number, the (right) operation number and the operation symbol of the λth operation respectively.

For any APS algorithm $\mathcal{A}$, we only know $q(\mathcal{N})$ when the algorithm $\mathcal{A}$ starts, then we know $T_{\lambda'}$ ($1 \le \lambda' < \lambda$) when the λth operation runs. It is required that the operation numbers L_λ and R_λ must be known before E_λ runs. If $T_\lambda \in S_q(\mathcal{N})$, we call T_λ a final result, otherwise a temporary result. When we get all the final results of $S_q(\mathcal{N})$, the algorithm $\mathcal{A}$ ends.

Since the $+$ and $-$ operations are common for solving the APS problem in $\mathbb{R}$, we assume that any operation E_λ can only be $T_\lambda = L_\lambda + R_\lambda$ or $T_\lambda = L_\lambda - R_\lambda$. Then it is feasible to assume each operation costs equally in the APS problem. Thus we know $\tau(\mathcal{A})$ is the c-length (see [6] for more detailed explanations) of $\mathcal{A}$, and the sequence length $\tau(\mathcal{A})$ can be used as the measurement criterion of the time performance of $\mathcal{A}$.

Since the APS problem is in the algebraic framework [6], the sequence length as a measurement criterion of time performance is more sophisticated than the asymptotic time complexity, especially the sequence $E_1, E_2, \ldots, E_{\tau(\mathcal{A})}$ are the major part in the practical program. Therefore, we will use $\tau(\mathcal{A})$ as the measurement criterion in the analysis of the APS algorithm.

Furthermore, the parameters of the APS problem are very useful. Let $\boldsymbol{e}'_{\min} = (0, \ldots, 0)$, $\boldsymbol{e}'_{\max} = \boldsymbol{e}_{\max} - \boldsymbol{e}_{\min}$, and let $\mathcal{N}'$ denote the super cube bounded with $\boldsymbol{e}'_{\min}$ and $\boldsymbol{e}'_{\max}$. We can define similar q' and S'_q, and it does not affect the analysis of time performance. In other words, the partial sum function satisfies shift invariance when $\mathcal{N}$ shifts. Without loss of generality, we let $\boldsymbol{e}_{\min} = (0, 0, \ldots, 0)$ and $\boldsymbol{e}_{\max} = (N_0 - 1, N_1 - 1, \ldots, N_{n-1} - 1)$ in this paper.

3.3. The analysis of some APS algorithms

Here we give the analysis of different APS algorithms in image thresholding segmentation with the new measurement criterion. Suppose the range of the image is still $\mathcal{L}$.

These algorithms are only different in the process of calculating all the partial sums, thus we only give the sequence length of one APS algorithm as the measurement criterion of the whole algorithm.

We first consider the 2D thresholding method and introduce some different 2D APS algorithms. We take the calculation of $\nu_0(t_1, t_2)$ in Eq. (20) as the example, and the whole algorithm is Alg-2D-Example.

Example 8 (*2D Memoization Algorithm*)**.** For any $t_1, t_2 > 0$, we can obtain $\nu_0(t_1, t_2)$ by

$$\nu_0(t_1, t_2) = \nu_0(t_1 - 1, t_2) + \nu_0(t_1, t_2 - 1) - \nu_0(t_1 - 1, t_2 - 1) + p_{t_1 t_2}. \tag{23}$$

One can find more details of calculating $\nu_0(t_1, t_2)$ from [10]. We know the essence of this algorithm is memoization, thus we call the APS algorithm in [10] as the 2D memoization algorithm.

Example 9 (*2D Piling Algorithm*)**.** The APS algorithm in [25] is a 2D piling algorithm. The discussion of piling algorithms will be presented in Section 4. Note that the piling algorithm in $\mathbb{R}$ is similar to that in the commutative semigroup.

Now we employ the sequence length to analyze the 2D memoization algorithm and the 2D piling algorithm. The 2D memoization algorithm in [10] needs $3(L-1)^2+2(L-1)$ operations, while the 2D piling algorithm in [25] needs $2L^2-2L$ operations. That shows the algorithm of [25] is faster than the algorithm of [10].

We can also consider the 3D thresholding methods. The APS algorithms in [11,8] are both 3D memoization algorithms, which need $7(L-1)^3+9(L-1)^2+3(L-1)$ operations. In the meantime, the APS algorithm in [23] is a 3D piling algorithm, which needs $3L^3-3L^2$ operations. Therefore the algorithm of [23] is faster than the algorithms of [11,8].

3.4. The worst case of APS algorithms

The worst case analysis is often of particular concern, and here the property of q is very important to the running time of calculating, thus we must know which property of q can lead to the worst case. The following examples in $\mathbb{R}$ can illustrate this problem clearly, and it can also help to obtain the optimality proof in commutative semigroups in Section 5.

Example 10. Suppose there is one quantitative function q and some $\boldsymbol{e}_c \in \mathcal{N}$ such that $q(\boldsymbol{e}) = q(\boldsymbol{e}_c)$ $(\forall \boldsymbol{e} \in \mathcal{N})$, we know

$$S_q(\boldsymbol{e}) = q(\boldsymbol{e}_c) \sum_{\boldsymbol{e}_{\min} \preccurlyeq \boldsymbol{e}' \preccurlyeq \boldsymbol{e}} (1). \tag{24}$$

Let $\boldsymbol{e} - \boldsymbol{e}_{\min} = (e_0, e_1, \ldots, e_{n-1})$, then each partial sum can be simplified to

$$S_q(\boldsymbol{e}) = q(\boldsymbol{e}_c) \prod_{i=0}^{n-1} (e_i + 1). \tag{25}$$

In general, elements with the same value may lead to less computations, thus the function q must satisfy

$$q(\boldsymbol{e}_1) \neq q(\boldsymbol{e}_2) \quad (\forall \boldsymbol{e}_1 \neq \boldsymbol{e}_2, \boldsymbol{e}_1, \boldsymbol{e}_2 \in \mathcal{N}); \tag{26}$$

it means the function q must be injective in the worst case.

Example 11. Suppose $q(\boldsymbol{e}) = 0$ $(\forall \boldsymbol{e} \in \mathcal{N})$, we do not need any operations at all since each $S_q(\boldsymbol{e}) = 0$.

We can see the partial sums with the same value may also lead to less computations, thus the function S_q must satisfy

$$S_q(\boldsymbol{e}_1) \neq S_q(\boldsymbol{e}_2) \quad (\forall \boldsymbol{e}_1 \neq \boldsymbol{e}_2, \boldsymbol{e}_1, \boldsymbol{e}_2 \in \mathcal{N}); \tag{27}$$

it means the function S_q also must be injective in the worst case.

However, the worst case does not only require the function q and S_q are both injective. In fact, it requires any result composed by some quantitative values is not the other result composed by some different quantitative values.

Example 12. $S_q(\boldsymbol{e}_{\max})$ is usually the most complex partial sum, because it contains all the function values mapped from $\mathcal{N}$. Suppose there are some $\boldsymbol{e}_s, \boldsymbol{e}_t \in \mathcal{N}$ such that $S_q(\boldsymbol{e}_{\max})$ equals $q(\boldsymbol{e}_s) - q(\boldsymbol{e}_t)$, the running time for calculating $S_q(\boldsymbol{e}_{\max})$ is almost 0 if we know this property.

Finally we can present the property of q in the worst case: For any two functions $\delta_1, \delta_2 : \mathcal{N} \to \mathbb{Z}$, the function q must satisfy

$$\begin{aligned} &\sum_{\boldsymbol{e} \in \mathcal{N}} \delta_1(\boldsymbol{e}) q(\boldsymbol{e}) = \sum_{\boldsymbol{e} \in \mathcal{N}} \delta_2(\boldsymbol{e}) q(\boldsymbol{e}) \\ &\Leftrightarrow \delta_1(\boldsymbol{e}) = \delta_2(\boldsymbol{e}) \ (\forall \boldsymbol{e} \in \mathcal{N}). \end{aligned} \tag{28}$$

The worst case means we can only calculate all the partial sums by their definitions and we cannot take advantage of the special properties of q.

4. The APS problem in a commutative semigroup

The APS Problem in $\mathbb{R}$ can be extended to a more general case.

Definition 7 (*Mapping from Integer Grid Graph*)**.** The mapping q of the APS problem in g is defined as $q : \mathcal{N} \to g$, where $\langle g, + \rangle$ is a commutative semigroup.

The notation q is similar with the case in $\mathbb{R}$, while it does not mean g has any quantitative properties.

The property of q in commutative semigroups in the worst case is a little different from that in $\mathbb{R}$, because we only use the $+$ operation in commutative semigroups. For any two functions $\delta_1, \delta_2 : \mathcal{N} \to \mathbb{N}$, the function q must satisfy

$$\begin{aligned} &\sum_{\boldsymbol{e} \in \mathcal{N}} \delta_1(\boldsymbol{e}) q(\boldsymbol{e}) = \sum_{\boldsymbol{e} \in \mathcal{N}} \delta_2(\boldsymbol{e}) q(\boldsymbol{e}) \\ &\Leftrightarrow \delta_1(\boldsymbol{e}) = \delta_2(\boldsymbol{e}) \ (\forall \boldsymbol{e} \in \mathcal{N}) \end{aligned} \tag{29}$$

in the worst case. Note that here the domain of δ_1, δ_2 is $\mathbb{N}$ instead of $\mathbb{Z}$. In fact, the worst case here means the corresponding commutative semigroup is also the "worst" commutative semigroup.

The definition of such a mapping q means any operation in commutative semigroups must be

$$T_\lambda = L_\lambda + R_\lambda. \tag{30}$$

The definition of partial sum S_q here is similar to its definition in $\mathbb{R}$.

Problem 2 (*The Multidimensional APS Problem in a Commutative Semigroup*)**.** Given $\mathcal{N}, g, q$ and the corresponding S_q, we need to calculate $S_q(\mathcal{N})$.

Since the running time of the operation $+$ in a commutative semigroup g may be longer, the sequence length is more important to the analysis of the APS algorithm in g than that in $\mathbb{R}$.

4.1. Optimal algorithm

If we only want to get one $S_q(\boldsymbol{e})$ ($\boldsymbol{e} \in \mathcal{N}$), we can use a certain partial sum algorithm. We know this partial sum algorithm can also be described as a sequence of operations. However, such sequences cannot be designed individually and then be combined to solve the APS problem, because the points that different partial sums concerned may have overlap. Therefore, memoization algorithms are not suitable to handle the APS problem, and we need an overall algorithm.

It is feasible to sum all elements from the lower dimensional case to the higher dimensional case. Actually, such processes are similar to the piling, thus we name the corresponding algorithm as a piling algorithm. The algorithm Piling-All is proposed for solving the nD APS problem in a commutative semigroup, whose optimality will be shown in Section 5. We also need some subprocedures: The procedure All-Loop-Assignment assigns $q(P[0], \ldots, P[n-1])$ to $T(P[0], \ldots, P[n-1])$, and the procedure All-Loop-Add sums elements under a certain rule. Combined with All-Loop-Assignment and All-Loop-Add, we can obtain the algorithm Piling-All.

```
All-Loop-Assignment(T, i)
1  if i < n
2      for P[i] = 0 to N_i − 1
3          All-Loop-Assignment(T, i + 1)
4  else
5      T(P[0], ..., P[n − 1]) = q(P[0], ..., P[n − 1])
```

The procedure All-Loop-Assignment initializes data. T is a working array, which storages all the elements of $q(\mathcal{N})$ for the later summing process.

```
All-Loop-Add(T, i, V, D)
1  if i < n
2      for P[D[i]] = 0 to N_{D[i]} − 1
3          All-Loop-Add(T, i + 1, V, D)
4  else
5      T(P[0], ..., P[n − 1]) += T(P[0] − V[0], ..., P[n − 1] − V[n − 1])
6      // Only V[k] equals 1
```

The procedure All-Loop-Add implements an n-deep loop for summing the elements in certain dimension k. All the elements of the array V equal 0 but one element $V[k]$ equals 1, and this non-zero element $V[k]$ indicates that we will do the piling processing on the dimension k.

```
Piling-All(T)
 1  All-Loop-Assignment(T, 0)
 2  for i = 0 to n − 1
 3      V[i] = 0
 4      D[i] = i
 5  for i = 0 to n − 1
 6      V[n − 1 − i] = 1
 7      // It means the non-zero element V[k] in All-Loop-Add is V[n − 1 − i]
 8      All-Loop-Add(T, 0, V, D)
 9      V[n − 1 − i] = 0
10      Shift D right circularly by 1 position
```

The procedure PILING-ALL is the main procedure, which completes the task by the piling processes in each dimension. The array D is the permutation of $(0, 1, \ldots, n-1)$, which is obtained by shifting $(0, 1, \ldots, n-1)$ right circularly. We use D to control the index variables order in the loop of ALL-LOOP-ADD. This order is in accordance with the dimension that the piling processing is on, and it can hold the memory locality.

Now we prove the correctness of the algorithm PILING-ALL. When it is running, the value of $T(e_0, e_1, \ldots, e_{n-1})$ changes correspondingly. Before the first ALL-LOOP-ADD in PILING-ALL runs, the array V is changed to $(0, \ldots, 0, 1)$, which means we will pile the elements in the dimension $P[n-1]$. Thus we know that

$$T(e_0, e_1, \ldots, e_{n-1}) = \sum_{e'_{n-1}=0}^{e_{n-1}} q(e_0, e_1, \ldots, e_{n-2}, e'_{n-1}) \tag{31}$$

after this ALL-LOOP-ADD ends. Before the kth ALL-LOOP-ADD in PILING-ALL runs, $V[n-k]$ is changed to 1 and the rest $V[i]$ $(i \neq n-k)$ is still 0, which means we will pile the elements in the dimension $P[n-k]$. Thus we know that

$$T(e_0, e_1, \ldots, e_{n-1}) = \sum_{e'_{n-k}=0}^{e_{n-k}} \sum_{e'_{n-k+1}=0}^{e_{n-k+1}} \cdots \sum_{e'_{n-1}=0}^{e_{n-1}} q(e_0, \ldots, e_{n-k-1}, e'_{n-k}, e'_{n-k+1}, \ldots, e'_{n-1}) \tag{32}$$

when the kth ALL-LOOP-ADD in PILING-ALL ends. Therefore, when the last ALL-LOOP-ADD in PILING-ALL ends,

$$T(e_0, e_1, \ldots, e_{n-1}) = \sum_{e'_0=0}^{e_0} \sum_{e'_1=0}^{e_1} \cdots \sum_{e'_{n-1}=0}^{e_{n-1}} q(e'_0, e'_1, \ldots, e'_{n-1}). \tag{33}$$

So we obtain $T(e_0, e_1, \ldots, e_{n-1}) = S_q(e_0, e_1, \ldots, e_{n-1})$ for each $(e_0, e_1, \ldots, e_{n-1}) \in \mathcal{N}$ finally.

Lemma 1. *The sequence length of* PILING-ALL *is*

$$\sum_{i=0}^{n-1} \left(\prod_{j=0}^{n-1} N_j - \prod_{\substack{0 \le j \le n-1 \\ j \neq i}} N_j \right) = n \prod_{j=0}^{n-1} N_j - \sum_{i=0}^{n-1} \prod_{\substack{0 \le j \le n-1 \\ j \neq i}} N_j. \tag{34}$$

Proof. By induction on the dimension n, this lemma can be shown. □

4.2. 3D optimal algorithm

Since PILING-ALL has a recursive form and uses the nD array, it should be rewritten in a direct form in practice. Now we present the algorithm 3D-PILING-ALL as an example. 3D-PILING-ALL is a special case of PILING-ALL when the dimension $n = 3$, with which we can illustrate the principle of PILING-ALL.

```
3D-Piling-All(T)
 1  // The following nest-loops initializes data
 2  for P[0] = 0 to N_0 − 1
 3      for P[1] = 0 to N_1 − 1
 4          for P[2] = 0 to N_2 − 1
 5              T(P[0], P[1], P[2]) = q(P[0], P[1], P[2])
 6  // The following nest-loops are the piling on the dimension P[2]
 7  for P[0] = 0 to N_0 − 1
 8      for P[1] = 0 to N_1 − 1
 9          for P[2] = 0 to N_2 − 1
10              T(P[0], P[1], P[2]) += T(P[0], P[1], P[2] − 1)
11  // The following nest-loops are the piling on the dimension P[1]
12  for P[2] = 0 to N_2 − 1
13      for P[0] = 0 to N_0 − 1
14          for P[1] = 0 to N_1 − 1
15              T(P[0], P[1], P[2]) += T(P[0], P[1] − 1, P[2])
16  // The following nest-loops are the piling on the dimension P[0]
17  for P[1] = 0 to N_1 − 1
18      for P[2] = 0 to N_2 − 1
19          for P[0] = 0 to N_0 − 1
20              T(P[0], P[1], P[2]) += T(P[0] − 1, P[1], P[2])
```

Fig. 1 shows the process of 3D-PILING-ALL, in which $P[0]$, $P[1]$, $P[2]$ are denoted as x, y, z respectively. The procedure 3D-PILING-ALL piles the elements in each dimension, and the propagation directions are indicated by the dashed arrow in Fig. 1.

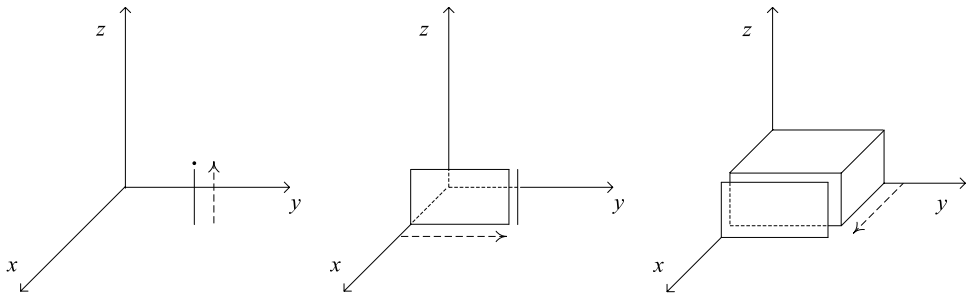

Fig. 1. 3D-Piling-All.

5. Optimality proof in commutative semigroups

The analysis of the APS algorithm in a commutative semigroup is also under the straight-line program computation model, and it is very similar to the analysis of the APS algorithm in $\mathbb{R}$. However, we want to know the minimum sequence length of the APS algorithm in commutative semigroups in the worst case since there is only a $+$ operation in commutative semigroups.

The structure of $\mathcal{N}$ is important to prove optimality, and we try to associate the operation with the points of $\mathcal{N}$. Our technique is to put all the operations to the points of $\mathcal{N}$, and each point can be viewed as a hole. Now all the operations are running in the holes, and the corresponding results are stored in the holes. We first prove that the operations of different holes are different, and we calculate the minimum time of the operations in each hole, thus we obtain the minimum sequence length of the APS algorithm in commutative semigroups. With this minimum sequence length, the algorithm Piling-All is shown to be optimal in the worst case.

5.1. Addition and merging

We will represent the operations of the APS algorithm in another perspective. For any APS algorithm in a commutative semigroup, $T_\lambda, L_\lambda, R_\lambda$ must be the sum of elements in $q(\mathcal{N})$. Let $nq(e)$ denote the sum of n $q(e)$, then

$$T_\lambda = \sum_{\boldsymbol{e}\in\mathcal{N}} \gamma_\lambda(\boldsymbol{e})q(\boldsymbol{e}), \tag{35}$$

$$L_\lambda = \sum_{\boldsymbol{e}\in\mathcal{N}} \beta_\lambda(\boldsymbol{e})q(\boldsymbol{e}), \tag{36}$$

$$R_\lambda = \sum_{\boldsymbol{e}\in\mathcal{N}} \alpha_\lambda(\boldsymbol{e})q(\boldsymbol{e}), \tag{37}$$

where $\gamma_\lambda(\boldsymbol{e}), \beta_\lambda(\boldsymbol{e}), \alpha_\lambda(\boldsymbol{e}) \in \mathbb{N}$.

If an operation obtains T_λ including a certain $\gamma_\lambda(\boldsymbol{e}) > 1$, we say T_λ useless. If an operation uses a useless T_λ as the operation number, it will lead to a new useless result $T_{\lambda'}$ ($\lambda' > \lambda$) with $\gamma_{\lambda'}(\boldsymbol{e}) \geq \gamma_\lambda(\boldsymbol{e}) > 1$. Since we discuss the worst case of the algorithm, these useless results cannot lead to the final results. If a useless result T_λ is a final result, it must satisfy $\gamma_\lambda(\boldsymbol{e}) = 1$ or $\gamma_\lambda(\boldsymbol{e}) = 0$ according to Eq. (29), and this leads to a contradiction. Hence, we do not need any useless results, then $T_\lambda, L_\lambda, R_\lambda$ must be

$$T_\lambda = \sum_{\boldsymbol{e}\in\mathcal{N}, \gamma_\lambda(\boldsymbol{e})=1} \gamma_\lambda(\boldsymbol{e})q(\boldsymbol{e}), \tag{38}$$

$$L_\lambda = \sum_{\boldsymbol{e}\in\mathcal{N}, \beta_\lambda(\boldsymbol{e})=1} \beta_\lambda(\boldsymbol{e})q(\boldsymbol{e}), \tag{39}$$

$$R_\lambda = \sum_{\boldsymbol{e}\in\mathcal{N}, \alpha_\lambda(\boldsymbol{e})=1} \alpha_\lambda(\boldsymbol{e})q(\boldsymbol{e}). \tag{40}$$

With these constraints, $T_\lambda, L_\lambda, R_\lambda$ can be viewed as sets.

Definition 8. The set forms of $T_\lambda, L_\lambda, R_\lambda$ are

$$\langle T_\lambda \rangle = \{\boldsymbol{e} | \boldsymbol{e} \in \mathcal{N}, \gamma_\lambda(\boldsymbol{e}) = 1\}, \tag{41}$$

$$\langle L_\lambda \rangle = \{\boldsymbol{e} | \boldsymbol{e} \in \mathcal{N}, \beta_\lambda(\boldsymbol{e}) = 1\}, \tag{42}$$

$$\langle R_\lambda \rangle = \{\boldsymbol{e} | \boldsymbol{e} \in \mathcal{N}, \alpha_\lambda(\boldsymbol{e}) = 1\}. \tag{43}$$

In the perspective of sets, we know that

$$\langle L_\lambda\rangle, \langle R_\lambda\rangle \subset \langle T_\lambda\rangle, \tag{44}$$

$$\langle L_\lambda\rangle \cap \langle R_\lambda\rangle = \varnothing, \tag{45}$$

$$\langle L_\lambda\rangle \cup \langle R_\lambda\rangle = \langle T_\lambda\rangle, \tag{46}$$

and E_λ means

$$\langle T_\lambda\rangle = \langle L_\lambda\rangle \cup \langle R_\lambda\rangle. \tag{47}$$

Now the operations of calculating $S_q(\boldsymbol{e})$ mean the merging of different elements in $\mathcal{N}$. Thus we will demonstrate the optimality with the structure of a grid graph.

5.2. Computation in holes

Our proof technique is similar to amortized analysis. We set the hole $\mathtt{h}(\boldsymbol{e})$ in any $\boldsymbol{e} \in \mathcal{N}$, and let all the operations run in the different holes. Thus the sequence length of the APS algorithm is the sum of the sequence length in all the holes.

Definition 9 (*Supremum*)**.** For any set $\Gamma \subset \mathcal{N}$, the supremum of Γ is

$$\sup \Gamma = \Big(\max_{\boldsymbol{e}\in\Gamma} e_0, \max_{\boldsymbol{e}\in\Gamma} e_1, \ldots, \max_{\boldsymbol{e}\in\Gamma} e_{n-1}\Big), \tag{48}$$

where $\boldsymbol{e} = (e_0, e_1, \ldots, e_{n-1})$.

The computation must satisfy the following requirements: Every $\{\boldsymbol{e}\}$ has been put in $\mathtt{h}(\boldsymbol{e})$ before the algorithm starts. The operation E_λ is set to run in $\mathtt{h}(\sup\langle T_\lambda\rangle)$, and the result $\langle T_\lambda\rangle$ will be put into $\mathtt{h}(\sup\langle T_\lambda\rangle)$.

Definition 10. For any $\boldsymbol{e} = (e_0, e_1, \ldots, e_{n-1}) \in \mathcal{N}$, let $\downarrow_i(\boldsymbol{e})$ be

$$\downarrow_i(\boldsymbol{e}) = (e_0, e_1, \ldots, e_i - 1, \ldots, e_{n-1}). \tag{49}$$

Note that $\downarrow_i(\boldsymbol{e})$ may not belong to $\mathcal{N}$.

Definition 11. Let $\boldsymbol{e} \in \mathcal{N}$, the core subset $C(\boldsymbol{e})$ of $\{\boldsymbol{e}' | \boldsymbol{e}_{\min} \preccurlyeq \boldsymbol{e}' \preccurlyeq \boldsymbol{e}\}$ is defined as

$$C(\boldsymbol{e}) = \{\boldsymbol{e}\} \cup \{\downarrow_i(\boldsymbol{e}) | 0 \le i \le n-1, \downarrow_i(\boldsymbol{e}) \in \mathcal{N}\}. \tag{50}$$

Lemma 2. *For any set $U \subset C(\boldsymbol{e})$ with $|U| > 1$, $\sup U = \boldsymbol{e}$.*

Proof. Since $U \subset C(\boldsymbol{e})$, we can obtain $\sup U \preccurlyeq \sup C(\boldsymbol{e}) = \boldsymbol{e}$ according to the definition of sup, then we discuss the different cases:

(1) Suppose $\boldsymbol{e} \in U$, then $\sup U = \boldsymbol{e}$.

(2) Suppose $\boldsymbol{e} \notin U$, we can take any two different elements from $U \subset C(\boldsymbol{e})$, then they must be $\downarrow_{i_1}(\boldsymbol{e})$ and $\downarrow_{i_2}(\boldsymbol{e})$ $(i_1 \neq i_2)$, thus $\sup U = \boldsymbol{e}$.

We have thus proved this lemma. □

Lemma 3. *For any set $U \subset \mathcal{N}$, $\sup U \preccurlyeq \boldsymbol{e}$ and $|U \cap C(\boldsymbol{e})| > 1$, then $\sup U = \boldsymbol{e}$.*

Proof. Since $U \cap C(\boldsymbol{e}) \subset C(\boldsymbol{e})$ and $|U \cap C(\boldsymbol{e})| > 1$, then $\sup(U \cap C(\boldsymbol{e})) = \boldsymbol{e}$. We know $\sup(U \cap C(\boldsymbol{e})) \preccurlyeq \sup U \preccurlyeq \boldsymbol{e}$, therefore $\sup U = \boldsymbol{e}$. □

5.3. The minimum sequence length

With the concept of holes, we will show the minimum sequence length of the APS algorithm in commutative semigroups in the worst case.

Lemma 4. *For any $U \subset C(\boldsymbol{e})$ with $|U| > 1$, we need at least $|U| - 1$ operations in $\mathtt{h}(\boldsymbol{e})$ to obtain U.*

Proof. There is only $\{\boldsymbol{e}\}$ in $\mathtt{h}(\boldsymbol{e})$ before any operation in $\mathtt{h}(\boldsymbol{e})$. Suppose the ith operation in $\mathtt{h}(\boldsymbol{e})$ is the ω_ith operation E_{ω_i} of the algorithm:

$$\langle T_{\omega_i}\rangle = \langle L_{\omega_i}\rangle + \langle R_{\omega_i}\rangle. \tag{51}$$

For any APS algorithm $\mathcal{A}$, we will prove the lemma by induction on $\tau(\mathcal{A})$, i.e. the lemma holds when the total sequence length is $\tau(\mathcal{A})$.

(1) E_{ω_1} is the first operation in $\mathtt{h}(\boldsymbol{e})$. We will prove the lemma holds when there is only one operation $(\tau(\mathcal{A}) = 1)$.

The result of E_{ω_1} must be in $\mathtt{h}(\boldsymbol{e})$, then $\sup\langle T_{\omega_1}\rangle = \boldsymbol{e}$, thus

$$|\langle T_{\omega_1}\rangle \cap C(\boldsymbol{e})| \ge 1. \tag{52}$$

Assuming that $|\langle L_{\omega_1}\rangle \cap C(\boldsymbol{e})| > 1$, we can obtain $\sup\langle L_{\omega_1}\rangle = \boldsymbol{e}$ because $\sup\langle L_{\omega_1}\rangle \preccurlyeq \sup\langle T_{\omega_1}\rangle = \boldsymbol{e}$. That means $\sup\langle L_{\omega_1}\rangle$ has been in $\mathtt{h}(\boldsymbol{e})$ before E_{ω_1}. Since there is only $\{\boldsymbol{e}\}$ in $\mathtt{h}(\boldsymbol{e})$ before E_{ω_1}, and we know $|\{\boldsymbol{e}\} \cap C(\boldsymbol{e})| = 1$, that means our assumption is wrong. Therefore, $|\langle L_{\omega_1}\rangle \cap C(\boldsymbol{e})| \leq 1$.

It is similar to prove $|\langle R_{\omega_1}\rangle \cap C(\boldsymbol{e})| \leq 1$. Since $\langle T_{\omega_1}\rangle = \langle L_{\omega_1}\rangle \cup \langle R_{\omega_1}\rangle$, then

$$|\langle T_{\omega_1}\rangle \cap C(\boldsymbol{e})| \leq 2, \tag{53}$$

so we obtain

$$1 \leq |\langle T_{\omega_1}\rangle \cap C(\boldsymbol{e})| \leq 2. \tag{54}$$

There is only one operation because $\tau(\mathcal{A}) = 1$, then $\langle T_{\omega_1}\rangle = U \subset C(\boldsymbol{e})$. Since $|U| > 1$, thus

$$|\langle T_{\omega_1}\rangle \cap C(\boldsymbol{e})| = 2 = |U|. \tag{55}$$

There is only $\{\boldsymbol{e}\}$ in $\mathtt{h}(\boldsymbol{e})$, so we need the operation E_{ω_1}. Therefore, we need at least $|U| - 1 = 1$ operation.

(2) For any $U \subset C(\boldsymbol{e})$ with $|U| > 1$, suppose we need at least $|U| - 1$ operations in the $\mathtt{h}(\boldsymbol{e})$ to store U into $\mathtt{h}(\boldsymbol{e})$ when the sequence length $\tau(\mathcal{A}) = \tau$.

The $(\tau + 1)$th operation only obtains $\langle T_{\omega_{\tau+1}}\rangle$. We need $|\langle L_{\omega_{\tau+1}}\rangle| - 1$ and $|\langle R_{\omega_{\tau+1}}\rangle| - 1$ operations respectively to get $\langle L_{\omega_{\tau+1}}\rangle$ and $\langle R_{\omega_{\tau+1}}\rangle$ in $\mathtt{h}(\boldsymbol{e})$. Since the essence of the operation is the union of sets, we know $\langle L_{\omega_{\tau+1}}\rangle \cap \langle R_{\omega_{\tau+1}}\rangle = \varnothing$, then the operations for calculating $\langle L_{\omega_{\tau+1}}\rangle$ and $\langle R_{\omega_{\tau+1}}\rangle$ are independent. The final operation for calculating $\langle T_{\omega_{\tau+1}}\rangle$ is $E_{\omega_{\tau+1}}$, so the whole sequence lengths for calculating $\langle T_{\omega_{\tau+1}}\rangle$ are

$$|\langle L_{\omega_{\tau+1}}\rangle| - 1 + |\langle R_{\omega_{\tau+1}}\rangle| - 1 + 1 = |\langle T_{\omega_{\tau+1}}\rangle| - 1. \tag{56}$$

Therefore, the lemma holds when the sequence length is $\tau + 1$. This completes the proof. □

Theorem 1. *For any $U \subset \mathcal{N}$ with $\sup U \preccurlyeq \boldsymbol{e}$ and $|U \cap C(\boldsymbol{e})| > 1$, we need at least $|U \cap C(\boldsymbol{e})| - 1$ operations in $\mathtt{h}(\boldsymbol{e})$ to obtain U.*

Proof. We can observe

$$\big(\langle T_{\omega_i}\rangle \cap C(\boldsymbol{e})\big) = \big(\langle L_{\omega_i}\rangle \cap C(\boldsymbol{e})\big) \cup \big(\langle R_{\omega_i}\rangle \cap C(\boldsymbol{e})\big) \tag{57}$$

to analyze corresponding E_{ω_i}

$$\langle T_{\omega_i}\rangle = \langle L_{\omega_i}\rangle \cup \langle R_{\omega_i}\rangle. \tag{58}$$

Since $U \cap C(\boldsymbol{e}) \subset C(\boldsymbol{e})$, we need at least $|U \cap C(\boldsymbol{e})| - 1$ operations in $\mathtt{h}(\boldsymbol{e})$ to obtain $U \cap C(\boldsymbol{e}) \subset C(\boldsymbol{e})$.

If this theorem were false, it would lead to one algorithm for getting $U \cap C(\boldsymbol{e})$ with sequence length less than $|U \cap C(\boldsymbol{e})| - 1$, so this leads to a contradiction. Therefore, we need at least $|U \cap C(\boldsymbol{e})| - 1$ operations in $\mathtt{h}(\boldsymbol{e})$ to obtain U. □

Theorem 2. *It takes at least $|C(\boldsymbol{e})| - 1$ operations in commutative semigroups to get $S_q(\boldsymbol{e})$.*

Proof. We can see $|S_q(\boldsymbol{e})| \geq 1$ according to its definition, then we discuss the problem in the different cases.

Suppose $|S_q(\boldsymbol{e}) \cap C(\boldsymbol{e})| > 1$, we need at least $|C(\boldsymbol{e})| - 1$ operations in $\mathtt{h}(\boldsymbol{e})$ according to Theorem 1.

Suppose $|S_q(\boldsymbol{e}) \cap C(\boldsymbol{e})| = 1$, it means $\boldsymbol{e} = \boldsymbol{e}_{\min}$ and we need no operations.

Therefore, we need at least $|C(\boldsymbol{e})| - 1$ operations in $\mathtt{h}(\boldsymbol{e})$ to get $S_q(\boldsymbol{e})$. □

Definition 12. The core degree $\Lambda(\boldsymbol{e})$ of $\boldsymbol{e}$ is defined as

$$\Lambda(\boldsymbol{e}) = |C(\boldsymbol{e})| - 1. \tag{59}$$

Theorem 3. *It takes at least*

$$\sum_{\boldsymbol{e}\in\mathcal{N}} \Lambda(\boldsymbol{e}) \tag{60}$$

operations in commutative semigroups to get all the elements of $S_q(\mathcal{N})$.

Proof. We need at least $|C(\boldsymbol{e})| - 1$ operations in commutative semigroups to obtain $\langle S_q(\boldsymbol{e})\rangle$ in $\mathtt{h}(\boldsymbol{e})$. The operations in different holes are different, therefore the minimum sequence length is equal to the sum of the minimum sequence lengths in all the holes. □

Lemma 5. *It takes at least $N_0 - 1$ operations in commutative semigroups for a 1D APS problem.*

Theorem 4. *The sum of all the core degrees in $\mathcal{N}$ is*

$$\sum_{\boldsymbol{e}\in\mathcal{N}} \Lambda(\boldsymbol{e}) = n\prod_{j=0}^{n-1} N_j - \sum_{i=0}^{n-1} \prod_{\substack{0\leq j\leq n-1\\ j\neq i}} N_j. \tag{61}$$

Proof. We will prove this theorem by induction on the dimension n. Let $\boldsymbol{e}(n)$ and $\mathcal{N}(n)$ denote the element of $\mathbb{N}^n$ and the integer grid graph in the nD case.

When $n = 1$, the theorem holds. When $n = k$, suppose

$$\sum_{\boldsymbol{e}(k)\in\mathcal{N}(k)} \Lambda\big(\boldsymbol{e}(k)\big) = k\prod_{j=0}^{k-1} N_j - \sum_{i=0}^{k-1}\prod_{\substack{0\le j\le k-1\\ j\neq i}} N_j. \tag{62}$$

When $n = k+1$, the sum of all the core degrees are composed of two parts.

One part consists of all the $\downarrow_k (\boldsymbol{e}(k+1))$, whose sum is

$$(N_k - 1)\prod_{j=0}^{k-1} N_j. \tag{63}$$

The other part consists of all the $\downarrow_i (\boldsymbol{e}(k+1))$ $(0 \le i \le k-1)$, whose sum is

$$N_k\Bigg(\sum_{\boldsymbol{e}(k)\in\mathcal{N}(k)} \Big|C\big(\boldsymbol{e}(k)\big)\Big|\Bigg) = N_k\Bigg(\sum_{\boldsymbol{e}(k)\in\mathcal{N}(k)} \Big(\Lambda\big(\boldsymbol{e}(k)\big) + 1\Big)\Bigg). \tag{64}$$

According to the assumption, this value is equal to

$$N_k\Bigg(k\prod_{j=0}^{k-1} N_j - \sum_{i=0}^{k-1}\prod_{\substack{0\le j\le k-1\\ j\neq i}} N_j\Bigg) + N_k\prod_{j=0}^{k-1} N_j. \tag{65}$$

Then the sum of all the core degrees in $(k+1)$D case is

$$\sum_{\boldsymbol{e}(k+1)\in\mathcal{N}(k+1)} \Lambda\big(\boldsymbol{e}(k+1)\big) = (k+1)\prod_{j=0}^{k} N_j - \sum_{i=0}^{k}\prod_{\substack{0\le j\le k\\ j\neq i}} N_j, \tag{66}$$

therefore we have proved the theorem. □

5.4. Optimality of some APS algorithms

Here we show the optimality of Piling-All, the APS algorithm in [25], and the APS algorithm in [23].

Theorem 5. Piling-All *is the optimal APS algorithm in commutative semigroups in the worst case.*

Proof. According to Lemma 1, Theorems 3 and 4, we can see the sequence length of Piling-All is exactly the minimum sequence length to get $S_q(\mathcal{N})$ in commutative semigroups, therefore Piling-All is optimal.

Corollary 1. *It takes at least $2N_0N_1 - N_0 - N_1$ operations in commutative semigroups for a 2D APS problem.*

We can see the minimum sequence length is $2L^2 - 2L$ when $N_0 = N_1 = L$, which shows that the APS algorithm in [25] is optimal in commutative semigroups. Actually, the APS algorithm in [25] is the special case of Piling-All when the dimension is 2.

Corollary 2. *It takes at least $3N_0N_1N_2 - N_0N_1 - N_1N_2 - N_2N_0$ operations in commutative semigroups for a 3D APS problem.*

We can see the minimum sequence length is $3L^3 - 3L^2$ when $N_0 = N_1 = N_2 = L$, which shows that the APS algorithm 3D-Piling-All and the two APS algorithms in [23] are optimal in commutative semigroups. However, the analysis in the original paper [23], which claims the time complexity is only $O(L^2)$, is wrong. Furthermore, 3D-Piling-All is equivalent to two APS algorithms in [23].

Corollary 3. *For any image with range $[0, L-1]$, the lower bound of the nD thresholding method based on the APS algorithm is $\Omega(L^n)$.*

6. Experimental results

We use some simulation experiments to compare the efficiency of different APS algorithms and check the effectiveness of the sequence length as the measurement criterion of time performance of the thresholding methods. The following algorithms are all implemented in MATLAB, and the hardware is a PC with an Intel Core 2 Duo E6320 (1.86 GHz) and 2 GB memory.

We have implemented the 2D Otsu's method, the 2D entropic method and the 3D Otsu's method with different APS algorithms, and list their results in Table 1, 2 and 3 respectively. We focus on the running time of the APS algorithm (denoted as APS Time) and the total running time of the whole method (denoted as Total Time), and we also calculate the ratio of APS Time to Total Time (denoted as Ratio). It should be pointed out that different APS algorithms here only lead to different running time, and they do not change segmentation result of each thresholding method.

As shown in Tables 1–3, the APS time of the piling algorithm is less than that of the memoization algorithm when segmenting the same image, which means the piling algorithm is faster than the memoization APS algorithm. Especially,

Table 1
Time performance of the 2D Otsu's method.

Image	Algorithm	APS time (s)	Total time (s)	Ratio (%)
Lena	Memoization	0.0078	0.0889	9
Lena	Piling	0.0056	0.0873	6
Camera man	Memoization	0.0077	0.0358	22
Camera man	Piling	0.0055	0.0306	18
Air plane	Memoization	0.0081	0.0895	9
Air plane	Piling	0.0056	0.0846	6

Table 2
Time performance of the 2D entropic method.

Image	Algorithm	APS time (s)	Total time (s)	Ratio (%)
Lena	Memoization	0.0053	0.0968	5
Lena	Piling	0.0035	0.0964	4
Camera man	Memoization	0.0052	0.0441	12
Camera man	Piling	0.0038	0.0437	9
Air plane	Memoization	0.0052	0.1009	5
Air plane	Piling	0.0037	0.0965	4

Table 3
Time performance of the 3D Otsu's method.

Image	Algorithm	APS time (s)	Total time (s)	Ratio (%)
Lena	Memoization	7.38	9.92	74
Lena	Piling	3.42	5.32	64
Camera man	Memoization	7.27	10.13	72
Camera man	Piling	3.54	5.51	64
Air plane	Memoization	7.48	10.07	74
Air plane	Piling	3.49	5.69	61

Table 3 shows that the memoization algorithm [23] can reduce the running time by about half ($\approx$46%) compared with the piling algorithm in [8] as we mentioned in Section 1. Actually, the piling algorithm is the fastest APS algorithm in $\mathbb{R}$ up till now. In the meantime, the total time of the whole method based on the piling algorithm is less than that based on the memoization algorithm. These results agree with the theoretical discussion in Section 3.3, thus the sequence length is a feasible criterion for measuring the time performance of different thresholding methods.

The ratio can show the importance of the APS algorithm in the thresholding method. All the ratio values in Table 3 are $>$60%, which means the running time of the APS algorithm is indeed the bottleneck of the thresholding method in the higher dimensional case. Furthermore, the running time of the piling algorithm in the 3D case is much longer than that in the 2D case, which means we may face more complex problems in the higher dimensional cases.

It should be pointed out that the preprocessing procedure (e.g. reading the image) costs much more than the procedure of the APS algorithm in the 2D case, which leads to the lower ratio values in Tables 1 and 2.

7. Conclusion

The partial sum calculation is the major part of thresholding methods in image thresholding segmentation, which has not been noticed before. It is necessary to calculate all the partial sums efficiently for optimal threshold searching. Therefore, we need to design better APS algorithms to enhance the time performance of segmentation, and the theoretical analysis and experimental results both show the APS problem is the bottleneck of the thresholding method.

The lower bound of the nD thresholding method based on the APS algorithm is $\Omega(L^n)$ when the dimension is n, while the running time of various $O(L^n)$ APS algorithms are different. We propose the piling algorithm for solving the APS problem, and show the optimality in commutative semigroups in the worst case under the straight-line program computation model.

The piling algorithm is general in any dimensional cases, we plan to apply this optimal APS algorithm to higher dimensional threshold methods in the future. The feasible generalized neighborhood gray values are needed first, which can form the higher dimensional histogram. We also need to find sound objective functions for more effective methods.

The APS algorithm in the higher dimensional cases still costs a little more, thus another further work is to construct a more practical APS algorithm. One possible way is to consider the parallel implementation of our algorithm, and the current results of the prefix-sum operation in parallel computations would help to modify the APS algorithm in the nD case. Moreover, the parallel complexity of the multidimensional APS problem is also an interesting problem.

Acknowledgements

The authors thank the three anonymous referees for their valuable comments and constructive suggestions.

This work was supported in part by the National Natural Science Foundation of China (Grant No. 60572133), the Natural Science Basic Research Plan in Shaanxi Province of China (Grant No. SJ08F24) and the Special Research Project of Shaanxi Department of Education (Grant No. 09JK731 and Grant No. 2010JK820).

References

[1] A.S. Abutaleb, Automatic thresholding of gray-level pictures using two-dimensional entropy, Computer Vision, Graphics, and Image Processing 47 (1989) 22–32.
[2] A.V. Aho, J.E. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.
[3] M. Portes de Albuquerque, I.A. Esquef, A.R. Gesualdi Melloa, M. Portes de Albuquerque, Image thresholding using Tsallis entropy, Pattern Recognition Letters 25 (9) (2004) 1059–1065.
[4] G.E. Blelloch, Prefix sums and their applications, Technical Report CMU-CS-90-190, School of Computer Science, Carnegie Mellon University, 1990.
[5] A.D. Brink, Thresholding of digital images using two-dimensional entropies, Pattern Recognition 25 (8) (1992) 803–808.
[6] P. Burgisser, M. Clausen, M.A. Shokrollahi, T. Lickteig, Algebraic Complexity Theory, Springer-Verlag, 1997.
[7] B. Chazelle, B. Rosenberg, The complexity of computing partial sums off-line, International Journal of Computational Geometry and Applications 1 (1) (1991) 33–45.
[8] J.L. Fan, F. Zhao, X.F. Zhang, Recursive algorithm for three-dimensional Otsu's thresholding segmentation method, Acta Electronica Sinica 35 (7) (2007) 1398–1402 (in Chinese).
[9] M.L. Fredman, The complexity of maintaining an array and computing its partial sums, Journal of the ACM 29 (1) (1982) 250–260.
[10] J. Gong, L.Y. Li, W.N. Chen, Fast recursive algorithm for two-dimensional thresholding, Pattern Recognition 31 (3) (1998) 295–300.
[11] X.J. Jing, J.F. Li, Y.L. Liu, Image segmentation based on 3-D maximum between-cluster variance, Acta Electronica Sinica 31 (9) (2003) 1281–1285 (in Chinese).
[12] J.N. Kapur, P.K. Sahoo, A.K.C. Wong, A new method for gray-level picture thresholding using the entropy of the histogram, Computer Vision, Graphics, and Image Processing 29 (1985) 273–285.
[13] S.U. Lee, S.Y. Chung, R.H. Park, A comparative performance study of several global thresholding techniques for segmentation, Computer Graphics and Image Processing 52 (1990) 171–190.
[14] J.Z. Liu, W.Q. Li, The automatic thresholding of gray-level pictures via two-dimensional Otsu method, Acta Automatica Sinica 19 (1) (1993) 101–105 (in Chinese).
[15] N. Otsu, A threshold selection method from gray-level histograms, IEEE Transactions on Systems, Man, and Cybernetics 9 (1) (1979) 62–66.
[16] N.R. Pal, S.K. Pal, Entropic thresholding, Signal Process 16 (1989) 97–108.
[17] M. Pătraşcu, E.D. Demaine, Tight bounds for the partial-sums problem, in: Proceedings of the 15th Annual ACM–SIAM Symposium on Discrete Algorithms, 2004, pp. 20–29.
[18] T. PUN, Entropic thresholding, a new approach, Computer Graphics and Image Processing 16 (1981) 210–239.
[19] P.K. Sahoo, S. Soltani, A.K.C. Wong, Y.C. Chen, A survey of thresholding techniques, Computer Vision, Graphics and Image Process 41 (1988) 233–260.
[20] P.K. Sahoo, G. Arora, A thresholding method based on two-dimensional Renyi's entropy, Pattern Recognition 37 (6) (2004) 1149–1161.
[21] P.K. Sahoo, G. Arora, Image thresholding using two-dimensional Tsallis-Havrda-Charvát entropy, Pattern Recognition 27 (6) (2006) 520–528.
[22] M. Sezgin, B. Sankur, Survey over image thresholding techniques and quantitative performance evaluation, Journal of Electronic Imaging 13 (1) (2004) 146–156.
[23] L. Wang, H.C. Duan, J.L. Wang, A fast algorithm for three-dimensional Otsu's thresholding method, in: IEEE International Symposium on IT in Medicine and Education, 2008, pp. 136–140.
[24] N. Wang, X. Li, X.H. Chen, Fast three-dimensional Otsu thresholding with shuffled frog-leaping algorithm, Pattern Recognition Letters 31 (13) (2010) 1809–1815.
[25] X.J. Wu, Y.J. Zhang, L.Z. Xia, A fast recurring two-dimensional entropic thresholding algorithm, Pattern Recognition 32 (1999) 2055–2061.
[26] A.C. Yao, On the complexity of maintaining partial sums, SIAM Journal on Computing 14 (2) (1985) 277–288.