Pattern Recognition Letters 80 (2016) 216–223

Contents lists available at ScienceDirect

Pattern Recognition Letters

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

Fast binary image set operations on a run-based representation☆

Siyu Guo*, Weifang Zhou, He Wen, Mengxia Liang

College of Electrical and Information Engineering, Hunan University, Changsha 410082, China

ARTICLE INFO

Article history:
Received 31 December 2015
Available online 14 July 2016

Keyword:
Set operation
Logical operation
Run
Binary image

ABSTRACT

Set operations are common processing of binary images. Though set operations implemented through naïve pixel-by-pixel logical operations are usually efficient, applications exist where the number of required set operations is large and faster set operations are needed. For such applications, a run-based representation, run forest, of binary images is proposed, and commonly used set operations of intersection, union, complementation, symmetric difference and set difference are realized on it. Run forests are lists of columns, which are also lists of runs in an image column. Spatial relations of two runs are exhaustively enumerated. A data structure called run iterator is designed to elegantly handle the operations of two runs. Run operations themselves consist of logical operations and assignments of integers, and can thus be very fast. Taking advantages of the simplicity of run operations, as well as the nature of run forest being compressive and well ordered, the set operations are efficiently realized. Experimental results show that although conversions of binary images to and from run forests cause computational overheads, this can be quite compensated during the computation of enough set operations, making the proposed method a suitable choice for applications with many set operations among largely fixed binary images, or applications using the run forest as the base representation throughout.

1. Introduction

Set operations of binary images are commonly used low level image processing operations. Since they can equivalently be expressed in logical terms, they are sometimes called as logical operations as well. Their applications are abundant in a wide range of fields such as region-of-interest manipulation, shape representation and reconstruction, and object recognition.

The most commonly used implementation of binary image set operations is to use pixel-by-pixel logical operations. Other approaches also exist, usually based on a specific binary image representation data structure. Chain codes [8], runs [15] and quadtrees [17] are examples of the elementary data structures for binary images widely used in image processing tasks, while various representations, though not frequently appearing in the recent literature, are also available. Some authors categorized the binary image representations into three types: trees, strings, and sets of codes [18, 25]. Besides quadtrees, binary trees are also used to encode binary images, which are constructed by recursively splitting an image into two halves along the horizontal and vertical directions alternately [13]. Tree structures suffer from memory costs on internal node storage and the pointers linking the nodes together, thus for the purpose of compression, leaf nodes of a tree are numerically encoded and consequently result in a set of codes denoting the original binary image. Quadtrees, octrees, binary trees can be converted into sets of codes following this idea [9,20,21]. Of course, codes can be generated not only from trees. For examples, Sarkar [18] and Wu and Chung [25] treat binary images as truth tables to derive codes from Boolean expressions. As for strings, other than the widely used Freeman chain codes, there are strings constructed from S-trees or pixel trees such as quadtrees [4,6,24]. It should be noticed that the above mentioned tree- and run-based representations all make use of a certain decomposition of the binary image. Other decomposition ways can be adopted, and new representations can be derived, e.g., Spiliotis and Mertzios [22] utilized a block representation of binary images. Suk et al. [23] surveyed a number of binary image decomposition methods. Though mathematical morphological operation results such as skeletons and analytical boundary representations such as splines can also be used to describe regions (shapes) in binary images, the set operations concerned in this paper are not easily performed on these representations, and we thus omit the introduction to the related literature.

Efficient binary image processing and analysis methods have been built upon these representations, such as geometric features, e.g., area and centroid, computation [19, 20, 22], geometric transformations, e.g., translation, scaling, rotation, and mirroring [3,20], connected component labeling [16], and so on. Several papers are

☆ This paper has been recommended for acceptance by Punam Kumar Saha.

* Corresponding author.
E-mail address: syguo75@163.com (S. Guo).

http://dx.doi.org/10.1016/j.patrec.2016.06.029
0167-8655/

directly related to our topic. In [22], the set difference operation, AND, OR, XOR, and NOT operations on block-represented binary images were given. AND, OR, and NOT operations on Interpolation-Based Bintree-based and Boolean function-encoded binary images were also proposed in [11] and [19], respectively. In [7], a run-based XOR operation implementation was given, whose basic idea is similar to ours, though in this paper we present a more comprehensive and elegant way to handle all the five commonly used set operations of binary image, i.e., AND, OR, XOR, NOT, and set difference.

During the search of related literature, we have noticed that the articles retrieved are relatively old. We give a possible explanation as follows. The set operations of binary images themselves can usually be efficiently done through the naïve pixel-by-pixel logical operations. Some operations might be speeded up by using one of the mentioned more sophisticated representations, but if the required representations of the binary images under concern are not readily available, the conversion from the binary images to the specific representation can be time consuming, and the speedup is not sufficient to compensate this cost. So the related researches are rare in the recent literature.

However, there exist applications where the compressed versions of the binary images involved in the set operations are at hand. One such application is given in the paper as an application example, which tries to find out the medial axes of plant leaves through leaf region reconstruction using skeletal information. In these applications, a large number of set operations are required, and during the whole computation, the elementary regions remain unchanged and can be converted into the desirable representation before the computation begins. In this scenario, the conversions may be worthwhile.

In this paper, we proposed the common set operations based on a run representation of binary images. This representation, called run forest, is essentially a list of image columns, any of which in turn is also a list of runs in the image column. The possible spatial relations of two runs are exhaustively summarized, and the run operations corresponding to the interesting set operations are defined. By considering the cases of every spatial relations of runs and introducing a run iterator data structure, the set operations of two columns are elegantly realized. Since run representation is a compact representation of the original binary images, the set operations implemented upon run forests can be significantly faster than the pixel-by-pixel logical operation implementation.

The paper is organized as follows. In Section 2 we describe run forest structure, the run-based representation of binary images used to accelerate the set operations. An exhaustive enumeration of spatial relations of two runs is presented in Section 3. Details of operations of AND, OR, XOR, NOT, and set difference implemented upon run forests are given in Section 4. In Section 5, synthetic and application experiments are introduced and conducted, with results shown and discussed. Finally, conclusions are drawn in Section 6.

2. Run forest representation of binary images

Runs are maximal sequences of logic 1 pixels (feature pixels) in a column (or row) in a binary image. For a given column, we characterize each run by two integers: i_S, the row index of the starting pixel, and i_E, the row index of the logic 0 pixel (background pixel) following the ending pixel of the run. The runs in a column are linked in ascending order of i_S into a list L_R. L_R and the column index, j_C, form the *column* structure. And the column structures in a binary image where runs exist are again linked in ascending order of j_C into a list L_C. In order to reconstruct the binary image, L_C and the height and width of the image, H and W, respectively, are encapsulated into a structure that we henceforth call the *run forest* (RF) of the binary image. The structure of RF is illustrated with an example in Fig. 1. The conversions of binary images to and from RFs are easy, so we omit the algorithm steps for these conversions in this paper.

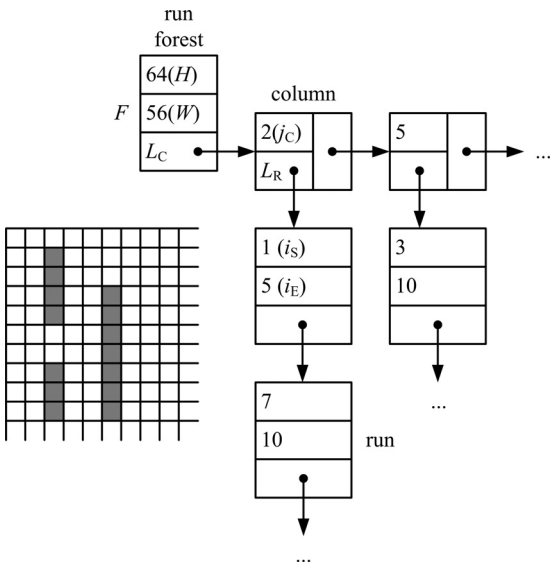

Fig. 1. Illustration of the run forest structure with an example. A run forest is a linked list, L_C, of columns in ascending order of column index, combined with the height H and width W of the image. A column is also a linked list, L_R, of runs in the image column, with the column index j_C stored together. Runs are characterized by i_S, its first pixel index, and i_E, the index of the pixel following the last pixel. Dark squares denote feature pixels.

During the run operations, it is convenient to define a "null" status of a run, which means the run does not contain any valid information. We set i_E as 0 to indicate that the run be null.

3. Spatial relations between two runs

In this section we consider the spatial relations of two runs in any same given columns from two binary images. These relations give cases exclusively treated in the run operations. Thirteen spatial relations are defined:

Let r_A and r_B be two runs. The relations $<_S$ (*strictly above*), $<_A$ (*adjacently above*), $<_O$ (*overlapped above*), $<_T$ (*top-aligned above*), $<_B$ (*bottom-aligned above*), $<_I$ (*strictly including*), $=$ (*equal*), $>_I$ (*strictly included*), $>_B$ (*bottom-aligned below*), $>_T$ (*top-aligned below*), $>_O$ (*overlapped below*), $>_A$ (*adjacently below*), and $>_S$ (*strictly below*) are defined as follows.

If r_A is null,	$r_A >_S r_B$;
otherwise, if r_B is null,	$r_A <_S r_B$;
otherwise, if $r_A.i_E < r_B.i_S$,	$r_A <_S r_B$;
otherwise, if $r_A.i_E = r_B.i_S$,	$r_A <_A r_B$;
otherwise, if $r_A.i_S < r_B.i_S$ and $r_A.i_E < r_B.i_E$,	$r_A <_O r_B$;
otherwise, if $r_A.i_S < r_B.i_S$ and $r_A.i_E = r_B.i_E$,	$r_A <_B r_B$;
otherwise, if $r_A.i_S < r_B.i_S$ and $r_A.i_E > r_B.i_E$,	$r_A <_I r_B$;
otherwise, if $r_A.i_S = r_B.i_S$ and $r_A.i_E < r_B.i_E$,	$r_A <_T r_B$;
otherwise, if $r_A.i_S = r_B.i_S$ and $r_A.i_E = r_B.i_E$,	$r_A = r_B$;
otherwise, if $r_A.i_S = r_B.i_S$ and $r_A.i_E > r_B.i_E$,	$r_A >_T r_B$;
otherwise, if $r_A.i_E < r_B.i_E$,	$r_A >_I r_B$;
otherwise, if $r_A.i_E = r_B.i_E$,	$r_A >_B r_B$;
otherwise, if $r_A.i_S < r_B.i_E$ and $r_A.i_E > r_B.i_E$,	$r_A >_O r_B$;
otherwise, if $r_A.i_S = r_B.i_E$,	$r_A >_A r_B$;
otherwise,	$r_A >_S r_B$.

These relations are also illustrated in Fig. 2. It can be seen in Fig. 2 that when we say $r_A <_* r_B$, the symbol "$<$" means that run r_A is somehow visually "above" run r_B. On the contrary, "$>$" implies